Simplified EVLA OTF Interferometry

W. D. Cotton (NRAO) November 11, 2016

***Abstract*—Snapshot imaging of "On-the-fly" (OTF) interferometry with short delay updates such that negligible rotation in parallactic angle occurs can ignore pointing variations and incorporate them into the beam used in mosaicing. This technique is explored and scripts developed for its implementation and are applied to test EVLA VLASS data in Obit. RMS values in the images of the order of 150 μJy/bm are typical in test data at 40° declination and 180 μJy/bm at 0° declination which are more strongly affected by geosynchronous satellites. The sampling of the antenna pattern by the OTF scanning pattern may adequately reduce the off–axis instrumental polarization allowing it to be ignored although accurate on-axis polarization calibration is critical. Timing tests leads to estimates of the imaging and mosaicing times for Stokes I, Q, and U of $\approx 35\times$ observe time on a single well equipped workstation using RAM disk and should scale to a diskless cluster. VLASS S band data could possibly be processed in near real time on a modest cluster and commensal VLITE observations may be tractable on a single workstation.**

***Index Terms*—Radio Interferometry**

I. Introduction

RAPID imaging using radio interferometers can be performed in "On the Fly" (OTF) mode in which the antennas are smoothly driven in pointing and the delay and phase tracking centers are periodically stepped. This mode is preferable over the "point and click" mode if the desired integration time per pointing is not significantly longer than the time for the antennas to move to a new tracking position and for oscillations to damp out. Imaging with the inclusion of pointing corrections can be quite expensive in computing resources. However, if all antennas observe the same positions and if the delay center updates are sufficiently often that time smearing during that interval is negligible, a more efficient approximate imaging technique can be employed. The technique discussed here is implemented in the Obit package ([1], http://www.cv.nrao.edu/∼bcotton/Obit.html).

II. "Aussie mode" OTF Interferometry

Consider the (simplified) response of an interferometer to a sky composed of n point sources:

$$v_{k,t} \;=\; \sum_{i=0}^{n} b_{i,k,t}\ s_i\ e^{-2\pi j(u_k*x_i+v_k*y_i)} \qquad (1)$$

where $b_{i,k,}$ is the antenna gain of the antennas at time t and in the direction of source i; s_i, x_i and y_i are the flux density and position of source i, and u_k and v_k are the baseline coordinates. If the antenna beams are moving on the sky, an integration over a finite interval will result in the integral of Eq. 1 over that interval. In general, multiple integrations will be recorded during the period over which the delay tracking center is held constant.

W. Cotton is with National Radio Astronomy Observatory, 520 Edgemont Rd., Charlottesville, VA, 22903 USA email: bcotton@nrao.edu

If the phase of the response of a source anywhere in the field of view on a given baseline and frequency varies by no more than an acceptable amount during the observations of a given delay tracking center, only the amplitudes are changing due to the moving antenna pointing. This condition is that:

$$(u_k*x_i+v_k*y_i) \;\approx\; constant \qquad (2)$$

for any source i and visibility $v_{k,t}$.

Under this condition and assuming all antennas point the same, a dirty image formed from the visibilities with a given delay tracking center will be the "true" sky dirty image multiplied by the time averaged primary beam. This can be seen from the case of a sky model with one component. Averaging over time, the dirty image derived from the Fourier transform of Eq. 1 is

$$D_i(x,y) \;=\; \int\int s_i\ e^{-2\pi j(u*x_i+v*y_i)}\ \frac{\int b_{i,t}dt}{\Delta t}du\ dv \qquad (3)$$

where $\frac{\int b_{i,t}dt}{\Delta t}$ is the average antenna gain over interval Δt in the direction of source i. Since the image formation given by Eq. 3 is linear, a similar expression holds for the sum of all sources in the sky model. The deconvolution of such an image will give the "true" CLEAN image multiplied by the time averaged primary beam which needs to be used in the formation of mosaics.

The condition given by Eq. 2 also implies that the u-v coverage and thus the dirty beam, during the time of a given delay tracking center will be nearly constant. In the CLEAN deconvolution, the flux density of a given component will be multiplied by the average of the antenna gain over the entire period whereas the effect of the component in a given integration will be multiplied by the average beam gain during that integration. Thus, the subtraction of the CLEAN component will be in error for each integration but, given the assumption of a constant dirty beam, this error will average to zero. In this case, the motion of the antenna beam can be ignored in the imaging and deconvolution and corrected when making the primary beam correction. Thus, more-or-less standard wideband, widefield snapshot images can be made in all desired Stokes parameters and combined into a linear mosaic.

Editing of the data must be done with care. If some samples of a given channel and polarization are removed in a given delay setting while other are not, the averaging of error to zero will not occur. Likewise the EVLA data at S band has beam squint in which the RR and LL data have effectively different pointing positions. Imaging using "formal I" (requiring both RR and LL) is needed.

The purpose of OTF interferometry is to obtain images over an extended region so a large number of individual delay pointing images are combined into a mosaic. For purposes of this combination, the effective primary beam is the convolution of the instantaneous beam with its trace on the sky.

I call this “Aussie mode” OTF interferometry as this is the technique used for the ATCA (although I have found no documentation of its use).

III. Examples

A sample region of an image derived by this technique is shown in Figure 1. This is an EVLA S Band image from near 40° declination where the image RMS is typically around 150 μJy/bm. The wideband imaging used multiple frequency planes (spectral windows) CLEANed jointly as described in [2], [3] and [4].

An example of a polarized, extended source is given in Figure 2. This region is near 0° declination and the I,Q and U RMSes are about 180μJy/bm due to the stronger RFI environment. In particular, 6 of the 16 spectral windows are completely lost including the 3 at the top of the band. Much of the emission in this source may be resolved out as the NVSS (1.4 GHz) catalog gives 635 mJy flux density (372 mJy extrapolated to 3 GHz) whereas the emission in Figure 2 amounts to 180 mJy.

An example of a polarized source with detectable polarized flux density in the individual frequency images is shown in Figure 3. Sources with sufficiently strong polarized emission for this analysis are relatively rare; 14 such cases were identified in a 20$^{\circ 2}$ search area.

A set of sample FITS images of the broadband brightness in Stokes I, Q and U are given in ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/EVLAOTF/. Rotation measure images are also given (name includes ’RM’) in which the first plane is the RM (rad/m^2), the second is the angle at zero wavelength (rad), the third is the error estimate of RM, the fourth the error estimate of the polarization angle and the fifth the χ^2 of the fit. Plots of EVPA vs. λ^2 are also given as gzipped postscript files.

IV. Limitations

The technique described above is an approximation and some level of artifact is unavoidable; in particular residual pointing errors may cause problems. Atmospheric phase fluctuations can be removed using phase self calibration with a solution interval of the entire scan. Some amount of the residual pointing errors can be removed using amplitude and phase self calibration when the source is strong enough - again, with a solution interval of the entire scan. An example of this is shown in Figure 4 for EVLA S band observations. The residual artifacts are clearly related to the dirty beam and would be reduced by a variation in parallactic angle. In a given observation, the same position is observed multiple times but over a short time interval minimizing the range of parallactic angle. Amplitude and phase self calibration reduced the artifacts to of order a few parts per thousand or less. Artifacts around complex regions will appear due to the limited uv coverage.

Another limitation is the limited depth of the CLEAN required by the snapshot imaging. Since each location on the sky is sampled many times in various places in the antenna pattern, the final mosaiced image will be more sensitive that the individual snapshot images; thus snapshot deconvolution will not remove sidelobes as effectively as a joint deconvolution. The good wideband snapshot coverage of the EVLA helps with this although it may be a problem with the poorer uv coverage of the VLITE array.

V. Off–axis Beam Effects

Position dependent beam effects have the potential to adversely affect wide–field imaging. With alt–az mounts such as the VLA has, the beam pattern will rotate on the sky with parallactic angle; the resultant averaging will reduce these effects. Snapshot imaging uses a single parallactic angle and off–axis effects are not reduced. OTF imaging consists of multiple snapshots but essentially all at the same parallactic angle.

One of the more serious off–axis beam effects is position dependent instrumental polarization which can be several percent in the usable part of the antenna pattern. This is frequently comparable to, or larger than, the source polarization at 3 GHz and has the potential to totally corrupt polarization measurements. A narrow band study made during the final days of the old VLA correlator [5] demonstrates this effect and a method to correct it. Unfortunately, the more recent wide–band beam holography measurements have yielded polarization results which are yet to be understood.

The scanning pattern of OTF observing, while wide–band and at a single parallactic angle, still allows relatively widespread sampling of the antenna pattern, hence averaging of the off–axis beam effects. Moreover, data obtained when a given source is closer to the pointing center (where beam effects are less) is given higher weight. As part of the VLASS test observations, a set of standard observations pointed towards the brighter sources in one of the Stripe 82 and Cepheus fields. These measurements of a source on–axis, and with longer integration can be compared against OTF images of the same sources to look for residual beam effects. Such a comparison is shown in Figure 5.

The two sets of data points basically follow each other except for two outliers in Q. For one of these, the difference amounts to 0.3% of Stokes I which may just represent the limitations of the technique. The difference for the other is about 1% of Stokes I and is cause for concern. The polarization for this source observed in the two modes is shown in Figure 6. The two measures of the rotation measure (RM) are essentially identical but there are small ($\sim 10^\circ$ differences in the phases. Differences in RFI editing lead to slightly different λ^2 coverages.

The offset seen in Figure 5 appears to be more of a difference in calibration than a residual instrumental polarization problem. The instrumental polarization is a strong function of distance from the pointing center and the 100% bandpass of the data leads to strong wavelength dependent beam location for a given source. Strong wavelength dependent effects as

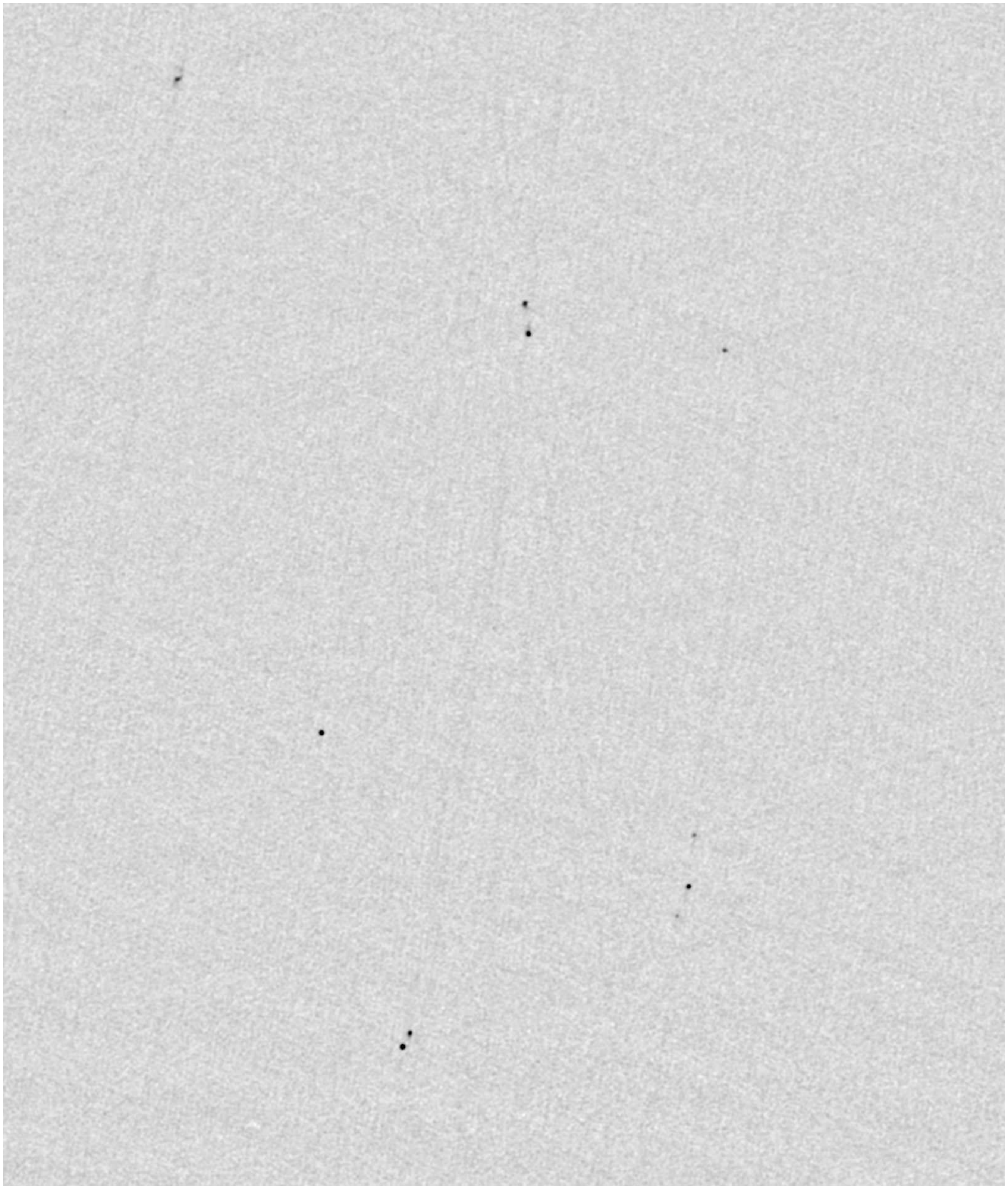

Fig. 1. Reverse grayscale of a particularly populous region of an OTF image made with the EVLA at S band. The displayed pixel range is -2 to +5 mJy/bm and the RMS is 150 μJy/bm. The region displayed is 13.'5 x 15.'9. The brightest source in the field is 11 mJy.

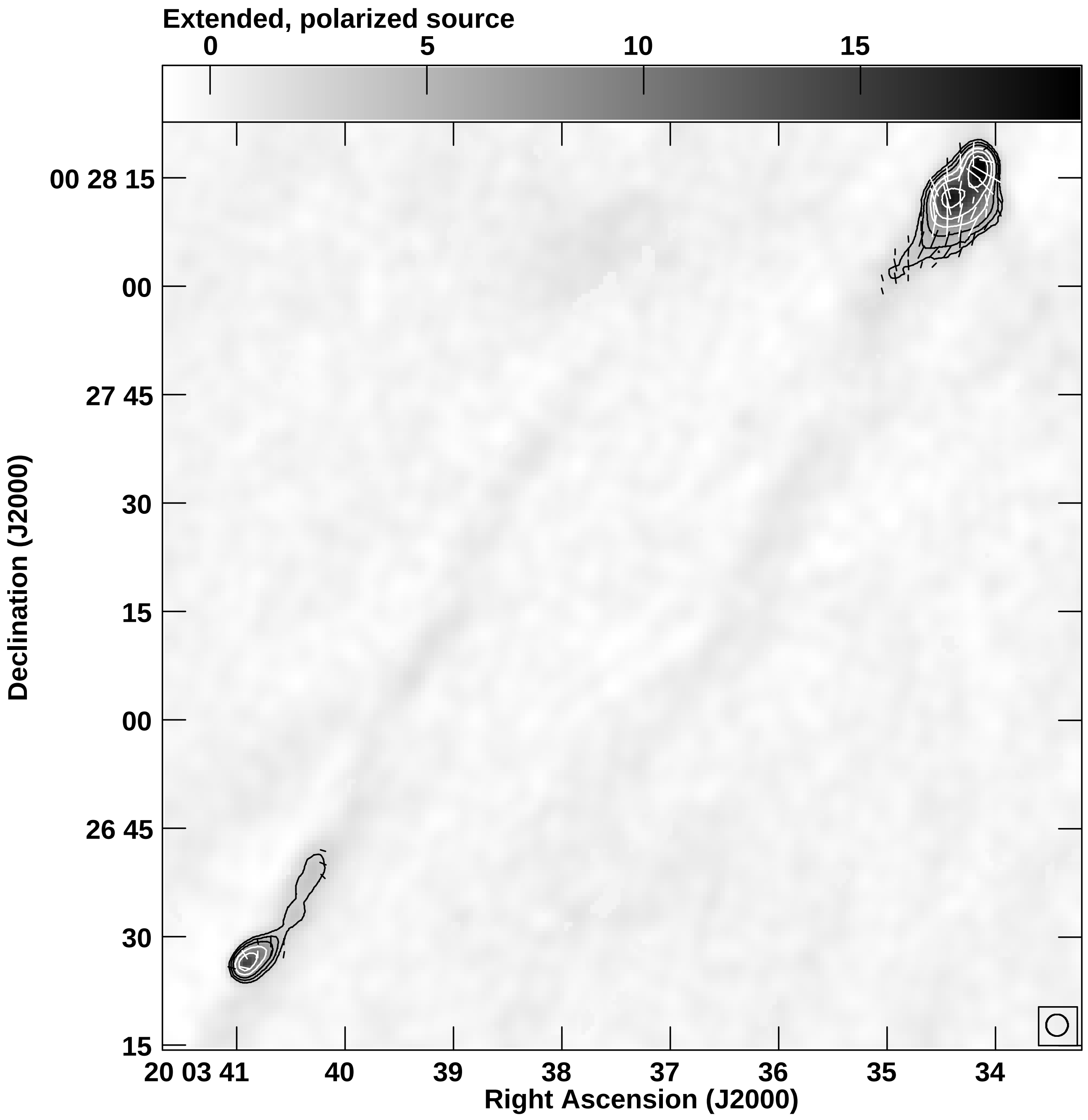

Fig. 2. Reverse grayscale of a extended polarized source from test VLASS data. The displayed pixel range is -2 to +20 mJy/bm (scale bar at top) and the RMS is 180 μJy/bm. Contours are of Stokes I and are powers of two from 2 mJy/bm. Vectors show the orientation of the polarization "E-vectors" with amplitudes proportional to the polarized flux density. The 3" restoring beam is shown in the lower right corner.

expected from off–axis instrumental polarization are not apparent in Figure 6.

VI. Difficult Areas

Complex, extended regions cause problems for snapshot imaging if the sky is more complex than is adequately sampled in the data. A particularly troublesome region from the Cepheus region is compared with the NVSS image in Figure 7. In the NVSS image (45" resolution, 1.4 GHz) the integrated flux density is ∼1650 Jy whereas in the VLASS test image (3" resolution, 3 GHz) the sum of the CLEAN components is about 4 Jy. The vast majority of the emission is resolved out leaving serious artifacts. In the default imaging, the amplitude and phase selfcalibration caused a serious distortion of the data. The VLASS image in Figure 7 was processed using 600 CLEAN components (standard 150 components) with ccfLim=0.65 (standard 0.35) and only phase self calibration. Only visibilities with amplitudes in excess of 8 Jy were flagged. The standard processing is described in Section VII. Automated windowing in Obit gave results comparable to hand boxing.

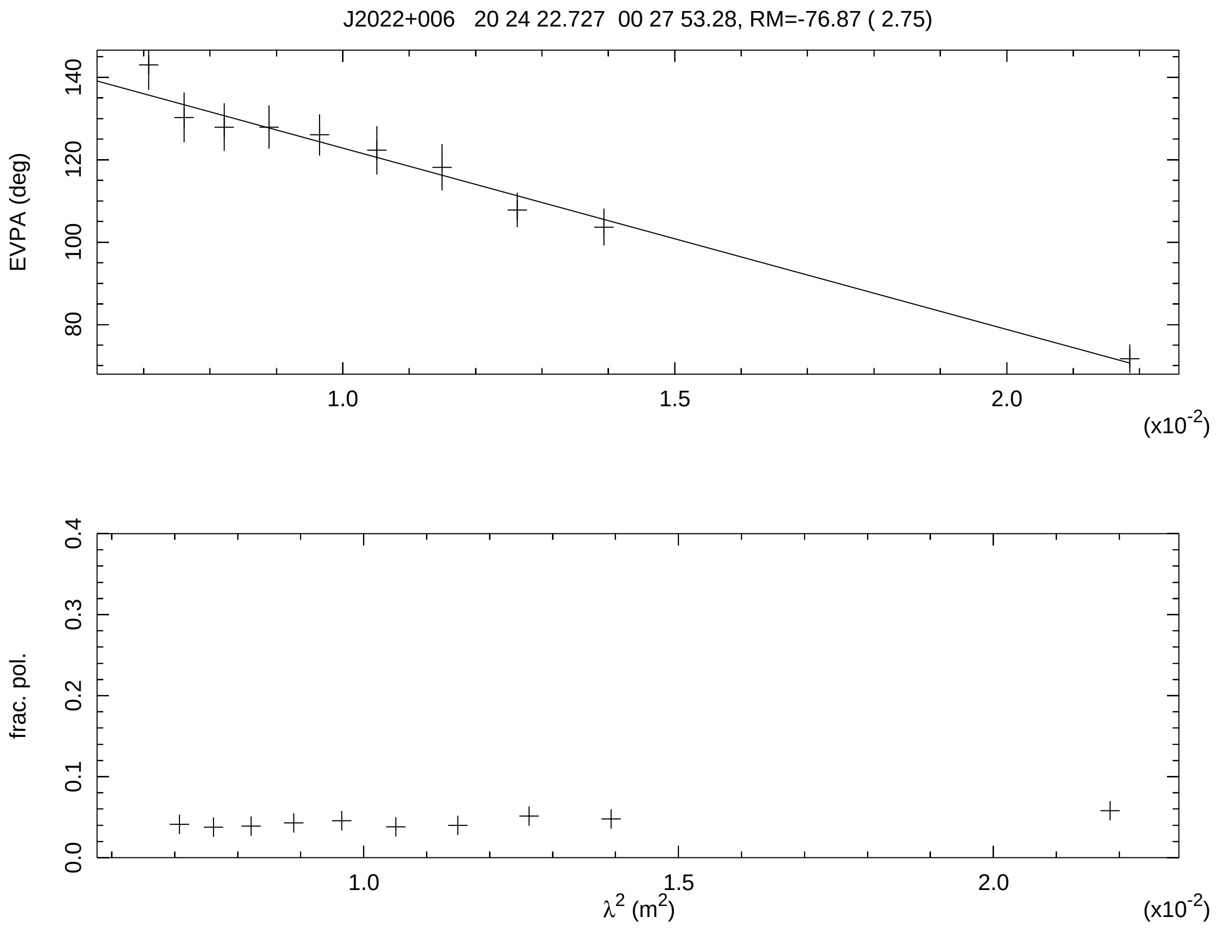

Fig. 3. **Top** gives the polarization angle v. λ^2 of a source. Solid line represents fitted RM whose value (rad/m^2) and error are given in the title. Error bars are derived from the image RMS and do not include calibration errors.
Bottom The fractional polarization as a function of λ^2. The Stokes I flux density is 145 mJy/bm and the I, Q and U RMSes 180 μJy/bm.

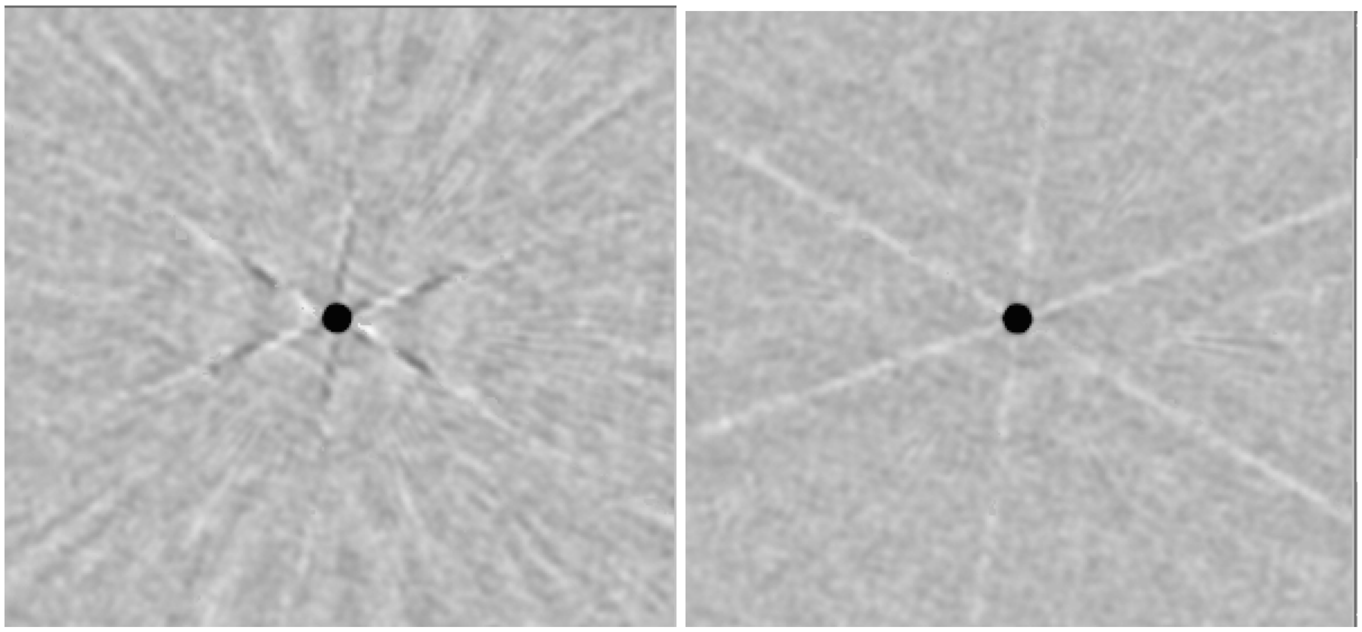

Fig. 4. Region around 300 mJy calibrator source in negative grayscale showing residual artifacts. Image noise is about 150 μJy/bm. Multiple overlapping images have been combined. Left has phase self calibration and right amplitude and phase self calibration. The pixel range displayed is -2 to +5 mJy/bm.

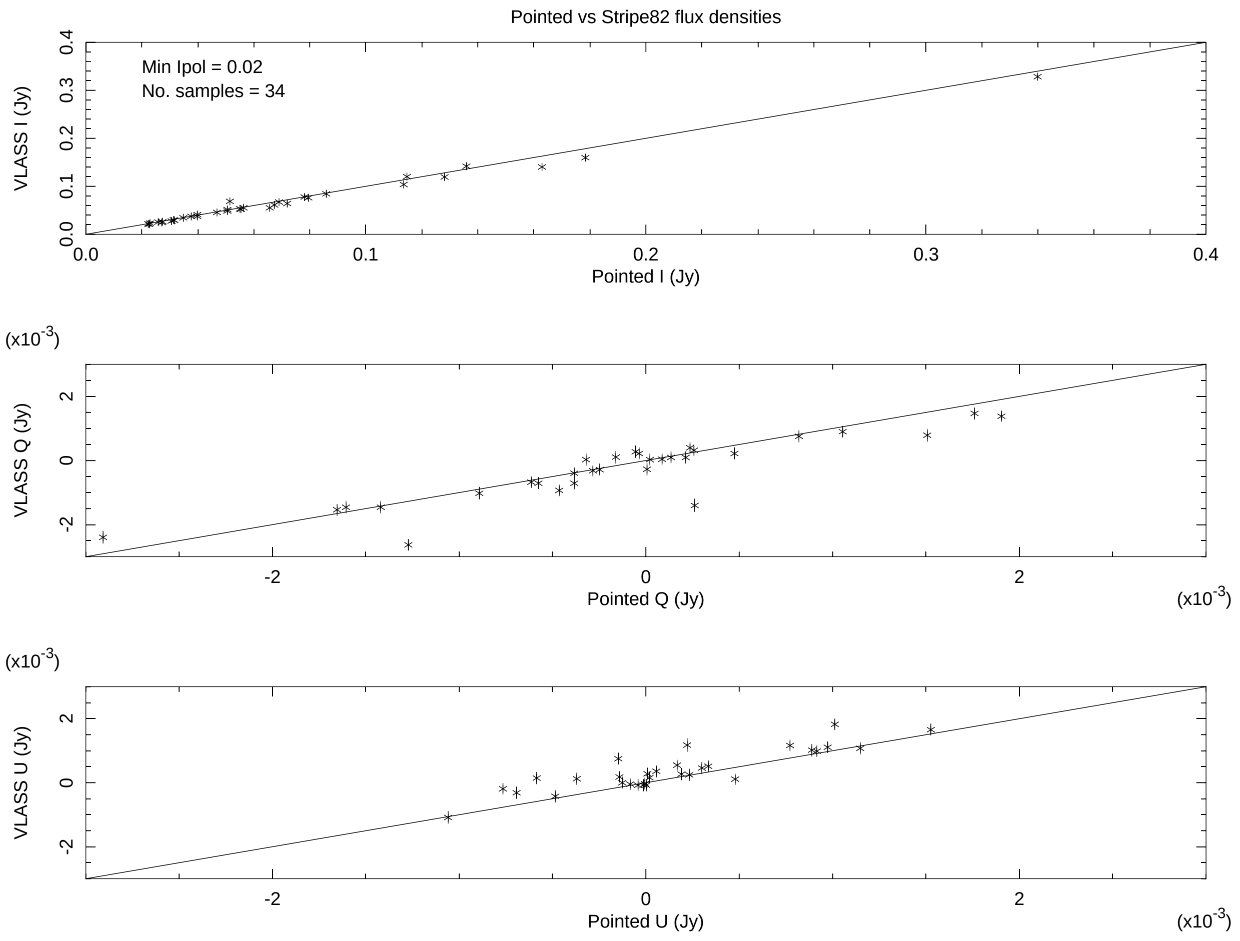

Fig. 5. A comparison of I, Q, and U peak flux densities for a number of sources in the OTF Stripe 82 field with pointed observations.

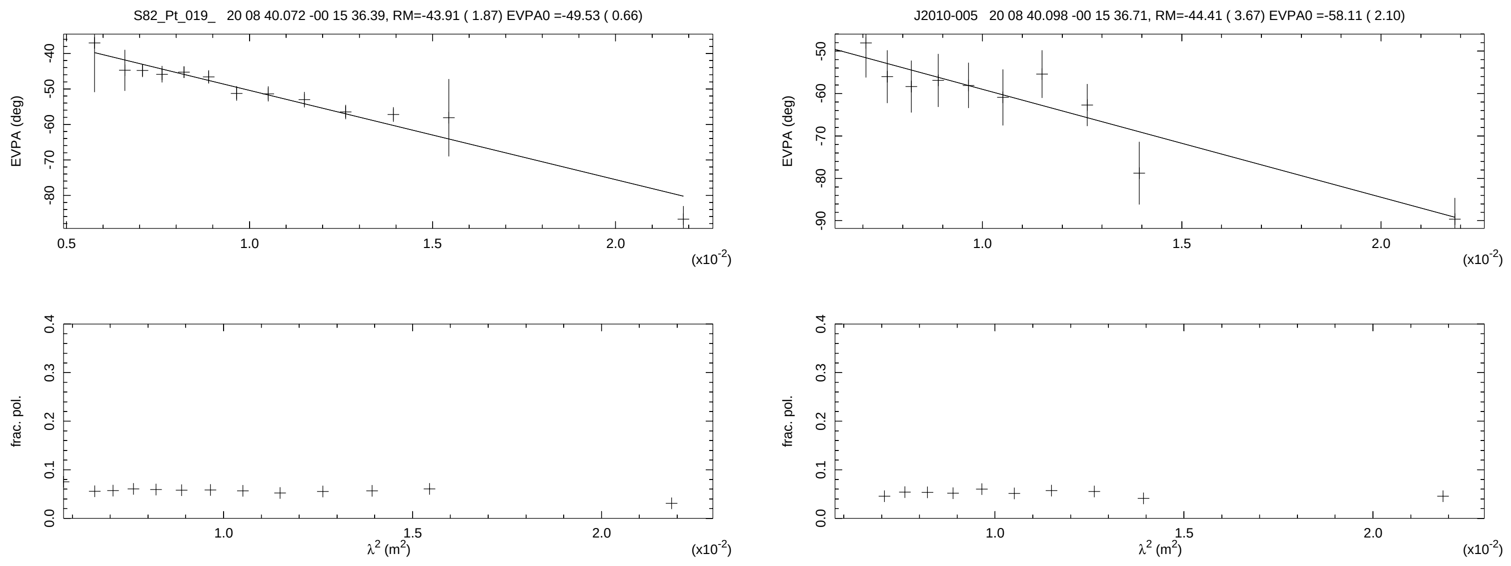

Fig. 6. Source polarization as a function of wavelength squared for a source observed in pointed and OTF mode. In each panel, the upper plot is the polarization angle and the lower plot the fractional polarization. The left panel is the pointed observation and the right the OTF image. The pointed observation had about 30 sec on source whereas the OTF imaging had the equivalent of 5 sec.

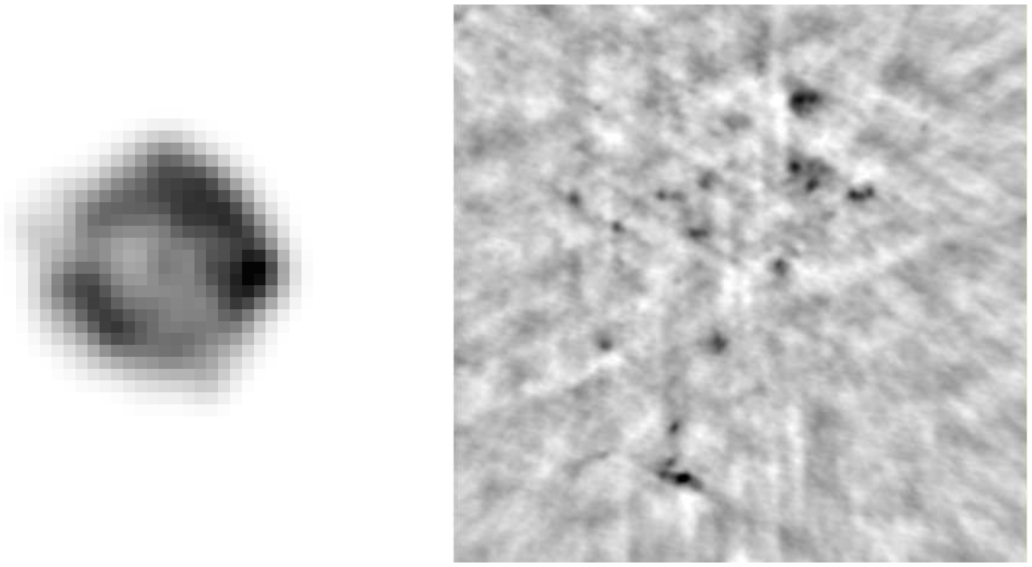

Fig. 7. Negative gray scale images of a region in Cepheus observed in NVSS (left, 45" resolution, 1.4 GHz) and VLASS (right, 3" resolution, 3 GHz). The size of the region shown in the left figure is 12.5' on a side and in the right image 4.5' and is centered on the brighter part of the left image.

VII. VLASS Data Processing

A scheme for processing VLASS using multiple processing streams is described in the following.

A. Calibration

Calibration used the standard Obit EVLA calibration scripts [7] including polarization but deferring imaging target fields. Task OTFFlag is used on the resultant UV data to assure that data within a given delay update (scan) are consistently edited.

B. Imaging

Imaging is done using scripts that allow multiple, parallel processing streams on the same host. The broadband nature of the data requires wideband-widefield imaging, especially in Stokes Q and U where variations with frequency are the primary observable. This is done in Obit task MFImage which images and deconvolves jointly in a number of frequency bins, here consisting of spectral windows. The details are given in [2], [3] and [4]. The scripts implementing these steps are given in an Appendix. Multiple processing streams may implement these scripts as limited by the resources on the host. Each processing script is driven by a list of pointings which is sorted to give processing in RA order.

Intermediate (delay pointing) images are stored in FITS images quantized to 50 μJy/bm. Zeroing the spectral index plane and recalculating the image max and min allows most images to be stored as 16 bit scaled integers. Another factor of $\sim$2 can be had by gzipping[1] the files, these can be read directly into the mosaicing software.

The imaging proceeds as follows:

1) **Quick Imaging** using task MFImage to get an estimate of the flux density in the field. A narrow field with outliers from the NVSS catalog is used with a shallow CLEAN. This uses the "doPS" option to avoid reprocessing pointings already done.
2) **Clip UV data.** The sum of the CLEAN flux densities from the previous step is used to flag high flux densities in the output UV data from the previous MFImage step. This uses task AutoFlag with a clip level of the greater of 0.5 and 3$\times$ the sum of the CLEAN flux from the previous step.
3) **OTF Editing.** Task OTFFlag is used to assure that the output data from the previous CLEAN are consistently edited.
4) **Deep Image.** The edited data from the previous CLEAN is used to perform a deeper CLEAN on the bulk of the primary beam. This uses task MFImage with self calibration in phase for fields with more than 25 mJy/bm peak and amplitude and phase for fields with peaks more than 100 mJy/bm. Imaging is done using spectral windows as the frequency binning. If parameter doStokes = True, I, Q and U images are derived else only I. The Stokes I images require "Formal I" or both RR and LL visibilities.
5) **Blank spectral index plane.** The spectral index plane in the output image cube was blanked to allow more efficient quantization of the pointing image. Spectral window planes with excessive RMSes due to heavy RFI flagging were also blanked.
6) **Reset Image max/min.** The output image cube has the actual range of pixel values determined and entered into the image header.
7) **Quantized FITS.** In order to reduce the volume of the intermediate images, they are written to FITS format as scaled integers quantized at 50 μJy/bm. These images were further reduced in volume using gzip.

[1]A multithreaded version of gzip, pigz, is available at http://zlib.net/pigz/.

C. Mosaic Formation

Mosaic images are formed for each plane of 1.3°^{2} images spaced by 1°; these are 6000×6000×18 pixels. Mosaics are formed by:

$$M(x,y) \;=\; \frac{\sum_{i=1}^{n} A_i(x,y)\; I_i(x,y)}{\sum_{i=1}^{n} A_i^2(x,y)}$$

where $A_i(x,y)$ is the effective antenna gain of pointing i in direction (x,y) and $I_i(x,y)$ is the pointing i pixel value interpolated to direction (x,y) and M is the mosaiced cube. Note, the pointing images have already been multiplied by the antenna gain. The division in the above equation needs to be restricted to pixels above a value of $\sum_{i=1}^{n} A_i^2(x,y)$, $minWt$, that keeps from excessively amplifying the noise at the edges of the spatial coverage. Planes in the input pointing images with RMS values in excess of 10× the RMS in the combined plane are ignored. This reduces the effect of RFI. After the image is normalized, the spectrum is refitted in each pixel giving the flux density at the reference frequency (first plane) and the spectral index (second plane). Spectral window images are in subsequent planes. The mosaic formation script using the Jinc function expected for a 25 m antenna is given as an appendix.

VIII. Timing and Disk Requirements

An initial imaging timing test was conducted on zuul05 which has 16 × 2 GHz cores, 64 GByte of memory, a very fast RAID disk and a (minimal) SSD. Data from a VLASS test observation were used and the first third of a 4 hour session including 4200 delay centers was imaged in Stokes I, Q and U using four parallel processing streams with 4 cores per stream. The elapsed time until the final stream finished was 5.6 days for an average of 1.9 min per delay center. This extrapolates to 16.8 days for the full 4 hour observation. Each delay center image must be kept until it can be accumulated into a mosaic. As imaged in this test, each delay pointing and polarization produces a 2400×2400×18 image cube or 0.3862 GByte with pixels as floats or 0.1931 as scaled 16 bit integer FITS images. Gzip compression reduces the scaled integer images to ≈0.1 GByte. Each 4 hour session has 12600 delay centers and imaging Stokes I,Q, and U gives 14.3 TByte if images are stored as floats, 7.1 TByte as 16 bit integers and ≈3.7 TByte compressed. However, the mosaicing need not be delayed until the imaging is complete. The four hour blocks are logically organized into 3 patterns covering 2°×10°. If each pattern is processed in RA order, the mosaics can be formed as the data are processed and the disk requirements reduced.

Processing of the second two thirds of a four hour observation producing only Stokes I took 123.5 hours = 5.1 days using 4 processing streams. This scales to 7.7 days for the full four hour session.

Mosaic formation was also tested on zuul05. Using 2 processing streams seems to give the most robust timings. Thirty 1.3°× 1.3° mosaics were formed using 2 parallel streams with an average of 4.6 hours each (or 2.3 hr per mosaic). This scales to 138 hr (=5.75 days) per polarization for the 60 mosaics produced by a 4 hour observation.

The estimated imaging plus mosaicing time for a four hour session for Stokes I, Q and U is 34 days or 204× observe time. Processing only Stokes I is estimated at 13.4 days per four hour session or 81 × observe time. These are the times for single good workstation and do not include calibration and editing and image analysis.

As processed here, the mosaics are 1.3°×1.3° (6000×6000) pixels spaced every 1°; there will be 60 of these per polarization covering the region surveyed in the 4 hour session. If these are to be combined with subsequent observations of the same region, the full (18 channel) resolution needs to be kept. Stokes I images can be decomposed into flux density and spectral index and Q+U to rotation measure, polarized flux density and polarization angle. At full resolution the 60×3 (I,Q,U) mosaic images take 0.75 TByte; for only Stokes I 0.25 TByte.

IX. SSD/RAM Disk

The fundamental limitation on the timing results given in the previous section is I/O speed. The volume of visibility data used in any given pointing is modest but due to the resolution, the volume of image data is more substantial. The multiple processing streams needed collide in disk access. This is even true using RAID or SSD disk systems as in the previous discussion. The linux file system tries to cache as much of the sections of the file system being used as it can in memory which can reduce the actual I/O but this technique has limits. Since the processing consists of a large number of relatively modest jobs; higher performance I/O solutions with modest storage capacity are possible.

One of these is solid state disk (SSD) which uses semiconductor memory rather than spinning oxide platters. The seek time of an I/O operation is greatly reduced and the transfer rate increased. These devices cost a few hundred dollars for units of a quarter to one terabyte capacity.

Even faster but with lower capacity is RAM disk in which a portion of the machine's memory is configured as a disk. Data on RAM disk resides in memory so there is no seek or transfer to memory time. However, memory allocated to RAM disk is not available for program memory or the linux disk cache and disappears on reboot.

In the following, timing tests of snapshot imaging and mosaicing are reported in which the intermediate files are all 1) on RAID disk, 2) in SSD or 3) in RAM disk. These tests were performed on smeagle which had a software RAID-5 disk system, 1/2 TByte SSD, 64 GByte of memory and 16 × 3.1 GHz cores. Creating RAM disk requires super human powers but is done by:

```
sudo mkdir /mnt/ramdisk;
sudo chmod 777 /mnt/ramdisk
# to create 8 GByte RAM disk
sudo mount -t tmpfs -o size=8G tmpfs \
    /mnt/ramdisk
# Set up AIPS directories, etc
mkdir /mnt/ramdisk/RAMDISK
touch /mnt/ramdisk/RAMDISK/SPACE
# to delete
```

```
sudo umount /mnt/ramdisk
```

A. Snapshot Imaging 6 Stream

The first timing test consisted of six parallel imaging streams, each imaging 42 delay centers or in aggregate 1/50 of those in a 4 hour observing block. Each stream was given a separate "AIPS disk" directory to prevent conflicts in the file system and was allocated a maximum of 2 processing cores and 30 of the 64 GByte of memory was allocated to RAM disk for the RAM disk test. Final images were written as scaled integer FITS files and gzipped using pigz. The results are given in Table I. Use of RAM disk ran over 4 times faster than the RAID disk and more than twice as fast as SSD. This imaging represents a factor of 2 improvement over the results quoted in Section VIII.

B. Mosaic Formation 4 Stream

This timing test consisted of four parallel streams, each forming a single mosaic $1.3^\circ \times 1.3^\circ$ in a single Stokes parameter. Each stream was allocated a maximum of 4 processing cores and 30 of the 64 GByte of memory was allocated to RAM disk for the RAM disk test. The results are given in Table II. Using the RAM disk produces run times a factor of nearly 6 shorter than using SSD and nearly 8 shorter than using the RAID disk. The I/O system was simply not up to handling the traffic using either RAID or SSD disk and showed serious I/O conflicts amoung the processing streams.

The results for the RAM disk test represents a factor of 2.9 improvement over the results quoted in Section VIII. When scaled to a four hour observing block, the RAM disk I, Q, U imaging and mosaicing tests would predict a total processing time of 342 hours or 86 times the observe time.

C. Snapshot Imaging 12 Stream

Previous tests were limited by the amount of memory available in smeagle (64 GByte). Smeagle's memory was upgraded to 256 GByte and the RAM disk timing tests repeated with twice the number of processing streams. 100 GByte was allocated to RAM disk and 12 imaging streams each given 100 delay pointings were run in parallel. Each stream was allocated a maximum of 2 (of 16) processing cores. In agregate, this is nearly 1/10 of the pointings in a 4 hour VLASS session. The average for the 12 streams was 11.0 hours or scaled to the full data set 29.2$\times$ the observe time. The intermediate snapshot images were quantized at 200 μJy/beam, written as scaled short integer FITS images and gzip compressed. The total volume produced was 147 GByte (average file size 42.6 MByte). Scaled to a full VLASS session, this would be 1.4 TByte of temporary data.

D. Mosaic Formation 8 Stream

Similarly, the timing test for the mosaic formation was repeated using 8 parallel streams, each forming a single mosaic $1.15^\circ \times 1.15^\circ$ (5500×5500 pixels) in a single Stokes parameter. The eight mosaics represent 4.4% of the 180 produced by a 4 hour VLASS session. Each stream was allocated a maximum of 2 processing cores. The average time for mosaic formation was 3.63 hours; scaled to the full 4 hour VLASS observing session this is 20.4$\times$ the observe time. With the 12 stream imaging this gives a total of 49.6$\times$ the observe time for I, Q and U mosaics.

E. GZip image compression

The bulk of the intermediate delay pointing snapshot images are greatly reduced using lossless, gzip (or pigz) compression. However, when a small number of cores are used in each stream, the computing cost of this compression becomes a significant part of the total processing. It is therefore worth exploring the tradeoff between speed and compression ratio allowed by the various levels of compression allowed by gzip/pigz. The speed and compression depend on the statistics of the contents of the file being compressed and are thus best evaluated using a sample data image.

A random delay snapshot FITS image (48 MByte) was compressed with different levels of compression using pigz with one core (gzip gives similar results). The results are given in Table III; column "cl" is the gzip compression level, "run time" is the wall clock time as determined by unix utility time, "time_9" is the ratio of the run time with that for compression level 9, size_9 is the ratio of the resultant file size with that for compression level 9 (maximum), and "compression" is the overall compression ratio of the image file. The higher compression levels take considerably longer for negligible reduction of file size. A compression level of 4 seems a good compromise and is over an order of magnitude faster that compression level 9 used in previous tests. Compression level 4 is used in the folowing tests.

TABLE I
IMAGING TIMING WITH 6 STREAMS

Disk	St. 1 hr	St. 2 hr	St. 3 hr	St. 4 hr	St. 5 hr	St. 6 hr	Avg hr
RAID	16.2	16.6	16.3	16.1	16.5	16.2	16.3
SSD	9.0	9.1	9.0	8.9	9.0	9.0	9.0
RAM	3.90	4.04	4.02	4.02	4.02	3.99	4.00

TABLE II
MOSAIC TIMING WITH 4 STREAMS

Disk	St. 1 hr	St. 2 hr	St. 3 hr	St. 4 hr	Avg hr
RAID	24.5	24.5	24.7	24.8	24.6
SSD	18.5	18.6	18.9	18.9	18.7
RAM	2.98	3.03	3.28	3.30	3.15

F. Snapshot Imaging 16 Stream

The previous imaging tests did not completely use the computational power available as much of the processing is scalar. A final set of tests was performed using 16 streams allocated one core each. The imaging test used 8 lists of 100 pointings each with one stream starting from each end. The depth of the Stokes Q and U CLEANs were reduced from the Stokes I CLEAN. The average stream time was 5.52 hours which scaled to a full VLASS session predicts 62.1 hours or 15.5 times the observe time.

G. Mosaic Formation 16 Stream

Sixteen parallel streams were used in the mosaic testing, each forming a single $1.15^\circ \times 1.15^\circ$ in a single Stokes parameter. Each stream used a single core. The average time for mosaic formation was 5.52 hours; scaled to the full 4 hour VLASS observing session this is $15.5\times$ the observe time. With the 16 stream imaging this gives a total of $33.8\times$ the observe time.

X. Discussion

The short delay center update cycle of the EVLA OTF integration enables a simplified approximation in the processing allowing something close to a tradition snapshot imaging and mosaic with the major modification being the effective primary beam. This is called “Aussie mode” as this is how the ATCA OTF is done. Test EVLA data in VLASS mode has been calibrated, imaged and mosaiced in Obit. RMS values in the images of the order of 150 μJy/bm are typical in test data at 40° declination and 180 μJy/bm at 0° declination which are more strongly affected by geosynchronous satellites. This noise level could possibly be improved somewhat by more clever RFI flagging as of order half the data is lost to RFI.

It is worth noting that while the actual beam pattern of real antennas are not completely azimutually symmetric and have off axis variations in the instrumental polarization, these effects are greatly reduced by the observing pattern which samples a given source in many widely spaced locations in the beam. The sum of the weights in the imaging mosaicing step for the VLASS data imaged is about 5; meaning that the total observation was the equivalent of 5 snapshots centered on a given position; this value is very constant across a given mosaic indicating very uniform sensitivity. This value also indicates that the snapshot noise RMS is a bit more than twice the combined mosaic value so the limited depth of the snapshot CLEAN is not a serious problem.

Timing tests of test VLASS data leads to estimates of the imaging and mosaicing times for Stokes I, Q, and U of $\approx 35\times$ observe time for a single well equipped workstation using RAM disk. The improved I/O performance of RAM disk allowed more simultaneous processing streams. RAM disk should also allow usage of diskless cluster nodes using the lustre system for distribution of data and collection of results. A modest cluster should allow processing times keeping up with observing. Scaling to multiple cluster nodes has not been tested but using RAM disk confines most I/O in each node so the residual I/O contention may not swamp the distributed file system.

The survey as observed by the commensal VLITE system which will have a factor of about 40 fewer delay centers and only Stokes I should be feasible on a single workstation. This testing used an effective primary beam calculated from the Jinc function expected for a 25 m antenna which is not quite correct but will not affect the results given here. The very limited uv coverage for the VLITE array may cause trouble for the technique described in this memo.

The density of sources with polarised intensity sufficiently strong to determine a rotation measure is of the order of 0.7 source or component every $1^{\circ 2}$. Proper test data is needed to determine if imaging with VLITE on 30 sec intervals gives usable results. Stokes I, Q and U results from $120^{\circ 2}$ of test data from the Stripe 82 field are given in ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/EVLAOTF/. Another $80^{\circ 2}$ of Stokes I in the Cepheus field is also given.

TABLE III
PIGZ COMPRESSION TIMING

cl	run time sec	time_9	size_9	compression
1	2.81	0.051	1.215	0.276
2	3.12	0.057	1.179	0.268
3	4.83	0.088	1.086	0.247
4	4.39	0.080	1.072	0.244
5	8.25	0.151	1.048	0.238
6	15.07	0.275	1.011	0.230
7	20.82	0.380	1.006	0.229
8	50.30	0.919	1.001	0.228
9	54.73	1.0	1.0	0.227

References

[1] W. D. Cotton, “Obit: A Development Environment for Astronomical Algorithms,” *PASP*, vol. 120, pp. 439–448, 2008.

[2] ——, “High dynamic range wideband imaging.” *Obit Development Memo*, no. 19, 2008. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/MFImage.pdf

[3] ——, “Wideband phase and delay calibration.” *Obit Development Memo*, no. 20, 2008. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/SelfCalWB.pdf

[4] ——, “Polarization Calibration and Imaging:Hercules A EVLA Data,” *Obit Development Memo*, no. 34, pp. 1–11, 2013. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/HercAPCal.pdf

[5] W. D. Cotton and R. Perley, “EVLA Off-axis Beam and Instrumental Polarization,” *Obit Development Memo*, no. 17, pp. 1–22, 2010. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/EVLABeam.pdf

[6] R. A. Perley and B. J. Butler, “Integrated Polarization Properties of 3C48, 3C138, 3C147, and 3C286,” *ApJS*, vol. 206, p. 16, Jun. 2013.

[7] W. D. Cotton, “EVLA Continuum Scripts: Outline of Data Reduction and Heuristics.” *Obit Development Memo*, no. 29, pp. 1–29, 2016. [Online]. Available: ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/EVLAObitScripts.pdf

APPENDIX

Imaging Stream Driver

The following has the script giving the details of a given processing stream.

```
# OTF data imaging from a list
# Loop over list, first imaging to get rough total flux density
# Then clipping
# Full Stokes I,(Q,U) imaging
# Save /quantize images
# Cleanup

from EVLACal import EVLAGetSumCC
import UV, Image
# Parameters
Aname  = 'EVLAOTFB'  # AIPS name of input uv data
Aclass = 'UVAvS'     # AIPS class of input uv data
idisk = 2            # AIPS disk of input uv data
disk = 1             # AIPS disk temporary files
seq = 1              # AIPS sequence of input uv data
noScrat = [1,3,4,5,6,7,8,9]  # AIPS disks to avoid for scratch
nThreads = 4         # Max number of threads
quant = 0.0002       # Quantization level
Stream = 1           # Stream number
StreamID = 'Stream'+str(Stream)
faillog  = StreamID+'Fail.log'  # Failure log
outDir = './Results/'           # directory for images
outDirt = '/mnt/ramdisk/RAM'+str(Stream)+'/'  # working directory
doStokes = True     # Image Q, U?

#example points = ['0037470+413416', '0038192+413416', '0042202+413416',]
execfile("pointS"+strStream+".py")
execfile("pass2s1.py")   # Pointing list
execfile("SISetup.py")   # Setup script
execfile("SILoop.py")    # Looping script
```

APPENDIX

Imaging setup

The following has the imaging setup script.

```
import OSystem, Image
uv = UV.newPAUV('Data',Aname, Aclass, disk, seq, True,err)

# First MFImage to measure flux density
mft = ObitTask("MFImage"); setname(uv,mft)
mft.doPS  = True   # use PS table to tell what has already been done
mft.doCalib = 2; mft.doBand = 1; mft.flagVer = 2
mft.xCells = 0.6; mft.yCells = mft.xCells; mft.doFit=False
mft.outDisk = disk; mft.out2Disk = disk; mft.outSeq = 1; mft.out2Seq = 1
mft.outClass = 'ITmp'; mft.out2Class = 'Temp'; mft.Stokes = 'F'; mft.maxFBW = -0.05
mft.FOV = 0.05; mft.minFlux = 0.25; mft.Niter = 50; mft.noScrat = noScrat
mft.OutlierDist = 0.35; mft.OutlierFlux = 0.1; mft.PBCor = False; mft.dispURL = 'None'
mft.targBeam = [3.0,3.0,0.0]; mft.nThreads = nThreads; mft.prtLv=1
if doStokes:
    mft.doPol = True; mft.PDVer = 1;
else:
    mft.doPol = False;

# Second MFImage to do real imaging
mf = ObitTask("MFImage"); setname(uv,mf)
mf.inClass = 'Temp'; mf.inDisk = disk; mf.inSeq = mft.out2Seq
mf.xCells = 0.6; mf.yCells = mf.xCells; mf.doFit=False
mf.doCalib = 0; mf.doBand = 0; mf.flagVer = 1
mf.outDisk = disk; mf.out2Disk = disk; mf.outSeq = 2; mf.out2Seq = 2
mf.outClass = 'I';   mf.out2Class = 'Final'; mf.maxFBW = -0.05
mf.FOV = 0.20; mf.minFlux = 0.0005; mf.Niter = 200
mf.OutlierDist = 0.35; mf.OutlierFlux = 0.005; mf.PBCor = False; mf.dispURL = 'None'
mf.targBeam = [3.0,3.0,0.0]; mf.Beam = [3.0,3.0,0.0]; mf.nThreads = nThreads; mf.prtLv=1
mf.refAnt = 16; mf.minSNR = 3.5; mf.avgPol = True; mf.noScrat = noScrat
mf.maxPSCLoop=1; mf.minFluxPSC=0.025; mf.solPInt=0.5; mf.solPType='L1'; mf.solPMode='P'
mf.maxASCLoop=1; mf.minFluxASC=0.10 ; mf.solAInt=0.5; mf.solAType='L1'; mf.solAMode='A&P'
if doStokes:
    mf.Stokes = 'FQU';
    # Less aggressive CLEANing
    NiterQU = mf.Niter/3; minFluxQU = mf.minFlux*2
    addParam(mf,"NiterQU", paramVal=NiterQU, shortHelp="Niter for Q,U,V", \
                 longHelp="  NiterQU.....Niter for Q,U,V\n")
    addParam(mf,"minFluxQU", paramVal=minFluxQU, \
                 shortHelp="minFlux for Q,U,V", \
                 longHelp="  minFluxQU...minFlux for Q,U,V\n")
else:
    mf.Stokes = 'F';

# AutoFlag
af = ObitTask("AutoFlag"); af.DataType='AIPS'; af.inClass='Temp';
af.inSeq=1; af.inDisk=disk
af.flagTab = 1; af.nThreads = nThreads; af.noScrat = noScrat;

# OTF flag
otf = ObitTask("OTFFlag"); otf.DataType='AIPS'; otf.inClass='Temp';
otf.inSeq=1; otf.inDisk=disk otf.flagTab = 1; otf.flagVer = 1;
otf.nThreads = nThreads;
```

Imaging Setup continued

```
# Get rid of wild points, reset max,min
mangle =  ObitTask("BASMangle"); mangle.DataType = ’AIPS’;
mangle.inClass = ’I’; mangle.inSeq = mf.outSeq; mangle.inDisk = disk
mangle.nSigma = 20; mangle.nThreads = nThreads
if doStokes:
    mangle.nStokes = 3
else:
    mangle.nStokes = 1

# Threading
OSystem.PAllowThreads(nThreads)

import OData, Table
def isDone(uv, point, err):
    """
    Check a PS Table to see of a pointing has been done

    Returns True if found and STATUS is "Done"
    uv    = OData with PS table
    point = name of pointing
    err   = Obit Error/message stack
    """
    #----------------------------------------------------------------
    ver = uv.GetHighVer("AIPS PS")
    if ver<1:
        return False
    pstab=uv.NewTable(Table.READONLY,"AIPS PS", 1, err)
    nrow = pstab.Desc.Dict[’nrow’]
    pstab.Open(Table.READONLY, err)
    pt = point.strip()
    done = False
    for irow in range(1,nrow+1):
        psrow = pstab.ReadRow(irow,err)
        if psrow[’FIELD NAME’][0].strip()==pt:
            done = psrow[’STATUS’][0].strip()==’Done’
            break

    pstab.Close(err)
    return done

# end isDone
```

APPENDIX

Imaging loop

The following has the imaging looping script.

```
# Loop over pointings
import FArray, OErr
OErr.PInit(err, taskLog =StreamID+’.log’)
OErr.PLog(err, OErr.Info, " Start processing "+StreamID);
import FArray
points.sort()  # Do in RA order
for p in points:
    # Already done:
    if isDone(uv, p, err):
        print p," Already done"
        continue
    try:
        name12 = p[0:12]; name8 = p[0:8];
        mft.Sources[0] = p; mft.g
        # do anything?
        if mft.nImaged<=0:
            print p," Already done"
            continue

        # Get clean flux for clip level
        xt = Image.newPAImage(’temp’, name12, mft.outClass, \
              mft.outDisk, mft.outSeq, True, err)
        maxVal = EVLAGetSumCC(xt, err)
        clip = max(0.5, 3*maxVal)
        print "Clip level", clip, "maxVal",maxVal
        zap(xt); del xt

        # clip
        af.inName = name12
        af.IClip = [clip,0.05];
        if doStokes:
            af.XClip = [clip,0.05];
        af.g

        # OTF flag
        otf.inName = name12; otf.g

        # Full Image
        mf.outName = p[8:12]; mf.out2Name = p[8:12]
        mf.inName = name12; mf.Sources[0] = p; mf.g

        # Get rid of wild points
        mangle.inName = name12;
        mangle.g
```

Imaging Loop continued

```
        # save images, quantize at quant
        xt = Image.newPAImage('tempI', name12, "I"+mf.outClass[1:], \
             mf.outDisk, mf.outSeq, True, err)
        xf = imtab(xt, outDir+p+"_I.fits", 0, err, quant=quant); del xf
        if doStokes:
            xtq = Image.newPAImage('tempQ', name12,"Q"+mf.outClass[1:], \
                mf.outDisk, mf.outSeq, True, err)
            xf = imtab(xtq, outDir+p+"_Q.fits", 0, err, quant=quant); del xf
            xtu = Image.newPAImage('tempU', name12,"U"+mf.outClass[1:], \
                mf.outDisk, mf.outSeq, True, err)
            xf = imtab(xtu, outDir+p+"_U.fits", 0, err, quant=quant); del xf
        # GZip with nThreads threads to ramdisk
        f = "pigz -4 -p "+str(nThreads)+" "+outDirt+p+"_*.fits"
        print f
        os.system(f)
        f = "mv "+outDirt+p+"_*.fits.gz "+outDir  # move to hard drive
        os.system(f)

        # cleanup
        uvt = UV.newPAUV('temp', name12, mft.out2Class, mft.out2Disk,\
            mft.out2Seq, True, err)
        zap(uvt); del uvt
        uvt = UV.newPAUV('temp', name12, mf.out2Class, mf.out2Disk, \
            mf.out2Seq, True, err)
        zap(uvt); del uvt
        xt = Image.newPAImage('tempI', name12, "I"+mf.outClass[1:],mf.outDisk, \
            mf.outSeq, True, err)
        zap(xt); del xt
        if doStokes:
            xt = Image.newPAImage('tempQ', name12,"Q"+mf.outClass[1:], mf.outDisk, \
                mf.outSeq, True, err)
            zap(xt); del xt
            xt = Image.newPAImage('tempU', name12,"U"+mf.outClass[1:], mf.outDisk, \
                mf.outSeq, True, err)
            zap(xt); del xt
    except:
        print "Processing failed for",p
        if (faillog):
            f=open(faillog,'a')
            line = "Processing failed for "+p+"\n"
            f.writelines(line)
            line = "    exception "+str(e)+"\n"
            f.writelines(line)
            f.close()
            AIPSDir.PAllDest (disk,err,Atype="  ",Aname=name12,Aclass='      ',Aseq=0)
# end loop
```

APPENDIX

Image Mosaic formation

The following scripts generate mosaics combining pointing images in a list (pointings) overlapping a list of positions (targets). Also needed are a template image (tmpl.fits) with the same geometry and other information as a pointing image and an effective beam image (see appendix “Create Effective Primary Beam”). Input and output images are FITS and AIPS used as internal accumulators.

```
# Script to make mosaic of EVLA test OTF data
# Images made with MFImage, make mosaics on each pointing adding all overlapping

import os
# List of pointings
execfile("pointAllC.py")
target   = ’list’     # Point to mosaic
IStream  = 1          # Processing stream
execfile(’Targ’+str(IStream)+’.py’)  # Stokes and mosaic names/positions

##############################################################################
strId    = ’S’+str(IStream)+stktrans   # Stream Id, unique, <= 5 char
ch0      = 0         # restart first at channel ch0
minWt    = 0.35      # Minimum weight
doGPU    = False     # Use GPU?
minCh    = 1         # Minimum channel number
maxCh    = 18        # Maximum channel number

datadisk  = 0   # disk for data (FITS)
fitsdir   = "../Results/" # FITS data directory
accumdisk = 1+IStream     # RAM disk for accumulators
accumseq  = 10   # seq for accumulators
isCont    = True # Continuum?
f = ’/mnt/ramdisk/OTFBeamQuant.fits’     # Effective primary beam
BeamImage = Image.newPFImage("Beam", f, 0, True, err)

nThreads = 4   # How many threads
size    = 5500   # size of master
cells   = 0.75   # cellsize
print "Target=",targets[0][0],"restart=",restart,"ch0=",ch0,"minWt=",minWt
execfile("mosaicBasic.py")
```

mosaicBasic.py

```
# Basic mosaicing functions
import Image, ImageDesc, MosaicUtil, OSystem, OErr
from OTObit import imlod
err = OErr.OErr()

OSystem.PAllowThreads(nThreads)
def separ(p,pos):
    """ get crude separation between p and pos

    EVLA OTF-like pointing names
    p   pointing name as hhmmsss+ddmmss
    pos mosaic position as (RA, Dec) in deg
    """
    rh   = int(p[0:2]); rm = int(p[2:4]); rs = 0.1*int(p[4:7]);
    sign = p[7:8]
    dd   = int(p[8:10]); dm = int(p[10:12]); ds = float(p[12:])
    ra  = (rh + rm/60. + rs/3600.)*15.0
    dec = dd + dm/60. + ds/3600.
    if sign=='-':
        dec = -dec
    delta = ((pos[0]-ra)**2 + (pos[1]-dec)**2)**0.5
    return delta
# end separ

def StitchF (nameList, stktrans, SumWtImage, SumWt2, BeamImage, err, restart=0,):
    """ Mosaic looping over planes of a list of cubes

        nameList  = list of source names
        stktrans  = Stokes, e.g. "ICube"
        SumWtImage= Image*wt accumulator
        SumWt2    =  Wt**2 accumulator
        OTFRA     = Array of RA offsets in deg not corrected for Declination
        OTFDec    = Array of Declinations offsets in deg, same size as OTFRA
        err       = Obit err stack
        restart   = restart channel no. 0-rel
    """
    # How many channels? */
    nchan = SumWtImage.Desc.Dict["inaxes"][SumWtImage.Desc.Dict["jlocf"]]
    # Crude position setup
    #   Mosaic center
    mosPos   = SumWtImage.Desc.Dict['crval'][0:2]
    # How close is close enough for proper test?
    mosDelta = abs(SumWtImage.Desc.Dict['cdelt'][0])*SumWtImage.Desc.Dict['inaxes'][0]
    # Loop over images
    i = -1;
    acc = None; acct = None
    for name in nameList:
        acc = None; acct = None
        i += 1
        # Quick test
        if separ(name,mosPos)>mosDelta:
            continue
        f = fitsdir+name+'_'+stktrans+".fits"
        print f,separ(name,mosPos)
        if not os.path.exists(f):
            # try gzipped
            f = fitsdir+name+'_'+stktrans+".fits.gz"
            if not os.path.exists(f):
                continue
```

mosaicBasic.py continued

```
        acct  = Image.newPFImage("accum", f, 0, True, err)
        if ImageDesc.POverlap(acct.Desc, SumWtImage.Desc, err):
            try:
                # Get overlap
                (blc,trc) = MosaicUtil.PGetOverlap(acct, SumWtImage, err)
                #blc=[1,1,1]; trc = acc.Desc.Dict['inaxes']
                if (trc[0]>blc[0]) and ((trc[1]>blc[1])):
                    blc[2] = minCh
                    trc[2] = maxCh  # Limit channels
                    # Too big for GPU?
                    if (trc[0]-blc[0])*(trc[1]-blc[1])>(6000*6000):
                        useGPU = False
                    else:
                        useGPU = doGPU
                    # unzip to ssd/ramdisk (disk 1)
                    acc = imlod(f,0,name[0:12],stktrans+strId,1,1,err)
                    # Use first plane to normalize RMS factors
                    Image.PGetPlane (acc, None, [1,1,1,1,1], err)
                    OErr.printErrMsg(err, "Error reading image for "+Image.PGetName(acc))
                    factor = 1.0;
                    RMS = acc.FArray.RMS;    # combined plane RMS
                    maxRMS = 10*RMS          # maximum acceptable plane RMS
                    print "Accumulate",name,"factor",factor, "maxRMS",maxRMS
                    print "overlap", blc, trc
                    #print "inaxes",acc.Desc.Dict["inaxes"]
                    MosaicUtil.PWeightImage(acc, factor, SumWtImage, SumWt2, err,
                                            iblc=blc, itrc=trc, restart=restart, planeWt=True,
                                            hwidth=2, doGPU=useGPU, inWtImage=BeamImage,
                                            maxRMS=maxRMS)
            except Exception,exception:
                OErr.printErr(err)
                print exception
                print "Failed"
                err.Clear()
            if acc:
                acc.Zap(err); del acc; acc = None
            if acct:
                del acct; acct = None
        else:
            print "Not in mosaic:",name
            if acct:
                del acct; acct = None
# end StitchF
```

mosaicBasic.py continued

```
# Loop over targets
for target in targets:
    # Make accumulation images from gcenter, cells, size x size
    if not restart:
        # make copy of template file 11 x 11 cells
        f = fitsdir+"tmpl.fits"
        if not os.path.exists(f):
            continue
        targ  = Image.newPFImage("target", f, 0, True, err)
        d    = targ.Desc.Dict
        si = ObitTask("SubImage")
        setname(targ,si)
        # select central 11 x 11 cells
        si.BLC = [int(d['crpix'][0]-5), int(d['crpix'][1]-5), 1]
        si.TRC = [int(d['crpix'][0]+5), int(d['crpix'][1]+5),  d['inaxes'][2]]
        si.TRC[2] = min(si.TRC[2],maxCh)  # Limit channels
        si.BLC[2] = max(si.BLC[2],minCh)
        si.outDisk = accumdisk; si.outSeq=accumseq; si.outDType="AIPS"
        si.outName =  target[0]; si.outClass = stktrans+"Tp";
        si.g
        f    = target[0]
        tmpl = Image.newPAImage("tmpl", f, stktrans+"Tp", accumdisk, accumseq, True, err)

        d    = tmpl.Desc.Dict
        d["cdelt"][0]=-cells/3600.;   d["cdelt"][1]=cells/3600.      # Cellsize
        d["crval"][0] = target[1][0]; d["crval"][1] = target[1][1];  # Center
        tmpl.Desc.Dict=d;tmpl.UpdateDesc(err)
        f    = target[0]
        SumWtImage = Image.newPAImage("WtI", f, stktrans+"WI", accumdisk, accumseq, False, err
        SumWt2     = Image.newPAImage("Wt2", f, stktrans+"W2", accumdisk, accumseq, False, err
        print "Create accumulators"
        MosaicUtil.PMakeMaster(tmpl, [size,size], SumWtImage, SumWt2, err)
        # Put reference pixel at center
        d = SumWtImage.Desc.Dict
        d['crpix'][0] = d['inaxes'][0]/2; d['crpix'][1] = d['inaxes'][1]/2;
        SumWtImage.Desc.Dict = d; SumWtImage.UpdateDesc(err)
        d = SumWt2.Desc.Dict
        d['crpix'][0] = d['inaxes'][0]/2; d['crpix'][1] = d['inaxes'][1]/2;
        SumWt2.Desc.Dict = d; SumWt2.UpdateDesc(err)
    else: # restarting
        f    = target
        SumWtImage = Image.newPAImage("WtI", f, stktrans+"WI", accumdisk, accumseq, True, err)
        SumWt2     = Image.newPAImage("Wt2", f, stktrans+"W2", accumdisk, accumseq, True, err)
        restart = False;ch0 = 0;  # Not again
    # end startup

    # Do it
    OErr.PLog(err,OErr.Info, "Start interpolation"); OErr.printErr(err)
    StitchF (pointings, stktrans, SumWtImage, SumWt2, BeamImage, err, restart=ch0)
    OErr.PLog(err,OErr.Info, "Finish interpolation"); OErr.printErr(err)
```

mosaicBasic.py continued

```
    # Normalize to FITS
    print "Normalize",target[0]
    f = fitsdir+target[0]+stktrans+'_Mosaic.fits'
    mosaic = Image.newPFImage("Mosaic", f, 0, False, err)
    SumWtImage.Clone(mosaic, err)
    MosaicUtil.PNormalizeImage(SumWtImage, SumWt2, mosaic, err, minWt=minWt)

     # If continuum (and from MFImage) calculate spectral index plane
    if isCont:
        import ImageMF
        xmf=ImageMF.newPFImageMF('MF', f, 0, True, err)
        ImageMF.PFitSpec(xmf,err)
        del xmf

    zap(SumWtImage); zap(SumWt2); zap(tmpl)
    del tmpl, SumWtImage, SumWt2, mosaic
# end loop
```

Appendix

Create Effective Primary Beam

The following script will create an effective primary beam including OTF effects from a template image tmpl.fits which should have the same geometry as the pointing images. This uses a Jinc approximation for the antenna beam shape.

```
import Image, ImageUtil, OErr
err = OErr.OErr()
inImage = Image.newPFImage('im', "tmpl.fits",0, True, err)
doPlane =[1,1,1,1,1]
Image.PGetPlane (inImage, None, doPlane, err)
OErr.printErrMsg(err, "Error reading image for "+Image.PGetName(inImage))
#WtImage = Image.Image("WeightImage")
#Image.PCloneMem(inImage, WtImage, err)
WtImage = Image.newPFImage("Beam","OTFBeam.fits",0,False,err)
inImage.Clone(WtImage,err)
OErr.printErrMsg(err, "Error reading image for "+Image.PGetName(inImage))
minGain = 0.02
d = 0.0124/2
OTFRA = [-3*d,-2*d,-d,0.0,d,2*d,3*d]; OTFDec = [0.,0.,0.,0.,0.,0.,0.]
for iplane in range(0,18):
    pln =[iplane+1,1,1,1,1]
    ImageUtil.POTFBeam (inImage, WtImage, OTFRA, OTFDec, err, minGain=minGain, outPlane=pln)
```