Testing Obit with a 100 GByte simulated dataset

W. D. Cotton, May 21, 2009

***Abstract*—When the EVLA correlator becomes operational it will be capable of generating datasets orders of magnitude larger than the VLA system was capable. This means that both development and operational software will need to handle far larger datasets than they have in the the past. In addition, far greater performance will be required to process this data in a timely fashion. This memo describes the generation of a 100 GByte simulated EVLA dataset and its use to test Obit pipeline software. Processing of this data volume is shown to be tractable with a single state of the art workstation. The use of multiple cores (8) made a dramatic improvement in the processing speed obtained; a factor or 5 or more in the imaging tests performed.**

***Index Terms*—interferometry, performance**

I. Introduction

THE era of data sets with volumes of 100s to thousands of GByte has nearly arrived and data analysis software needs to be prepared to handle this data in an efficient fashion. Until actual datasets become available testing must be done using simulated data. The following describes use of Obit([1], http://www.cv.nrao.edu/∼bcotton/Obit.html) to simulate realistic data with a size of 100 GByte. This data is then passed through a suite of editing, calibration and imaging processes similar to what would be done to a typical continuum dataset.

II. Simulated Data

The simulated data was generated using the VLA "B" configuration and consists of 19 ten minute scans over 9.5 hours of a "target" source at 60° declination. These were interspersed with one minute "phase calibration" scans and a single "Flux density/bandpass" calibration scan of 10 min. in length. Data samples were 2 second integrations and contained 1024 spectral channels divided among 32 "IFs" and spanning from 1.4 to 1.9 GHz. This generated a total of 2323269 visibility records of which 2003508 were on the "Target". In full precision (3 floats per visibility) this is 106.3 GByte of data and in "compressed" (2 shorts per visibility) 35.5 GByte of data.

The models used for the calibrator sources were 1 and 10 Jy point sources at the origin for the phase and amplitude/bandpass calibrators respectively. The sky model used for the target field was the CLEAN model derived from a moderately deep VLA survey (22 μJy/beam RMS) pointing using a similar setup. This sky model consists of 116 facets. Therefore, the sky model was completely realistic in terms of the distributions of positions, flux densities, sizes and shapes.

Gaussian noise of 0.5 Jy was added to each visibility measurement (to give roughly 10 μJy/beam RMS in the image); no primary beam gain or spectral index corrections were applied. Simulated data were written into compressed AIPS format and index (NX) and initial calibration (CL) tables were generated.

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

Manuscript received ; revised

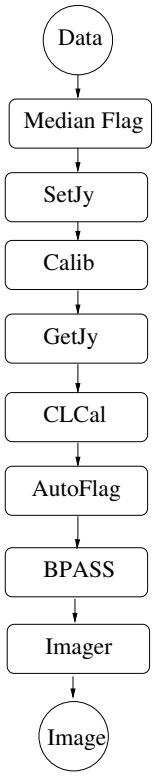

Fig. 1. Steps included in the processing pipeline.

III. Test Processing

Testing was performed on the Obit Development machine, mortibus, in Charlottesville. This machine has dual quad core Xenon processors for a total of 8 cores, a clock speed of 3 GHz, 8 GByte memory and a fast disk RAID system. Mortibus uses the Dell SAS/SATA RAID 5, PERC 6/i Integrated controller (made by LSI) with the Seagate 146GB15K RPM SAS 3Gbps 2.5-in HotPlug Hard Drive.

The test processing consisted of a pipeline containing a sequence of calibration, editing and imaging steps which might be typical for a continuum observation. This pipeline is described schematically in Figure 1 and listed in the appendix. Each of these steps is described in the following:

1) **Median Flag**
   Obit task MednFlag. This step looks for strong interference or data drop outs on calibrators by comparing each data sample with a running median amplitude. Data are flagged by entries in a flagging (AIPS FG) table.

2) **SetJy**
   Obit task SetJy. This sets the flux densities of the amplitude calibrator(s) in the AIPS SU table for use in subsequent calibration steps.
3) **Calib**
   Obit task Calib. This step determines the calibration gains for the amplitude and phase calibrators using either a point model or a CLEAN sky model. Solutions are written into a AIPS SN table.
4) **GetJy**
   Obit task GetJy. This task uses the gain solutions to derive the flux density of the phase calibration source from the amplitude calibration source. The phase calibrator gains in the AIPS SN table and the flux densities in the AIPS SU are modified.
5) **CLCal**
   Obit task CLCal. This step takes the calibration gains from the AIPS SN table and applied them to the initial calibration (AIPS CL) table and writes a new table which is used for subsequent processing.
6) **AutoFlag**
   Obit task AutoFlag. This editing process helps remove RFI by flagging any data with excessive values of Stokes I or V as well as doing a time domain search for periods of excessive fluctuations. Data are flagged by entries in a flagging (AIPS FG) table.
7) **BPASS**
   AIPS task BPASS. This task applies the prior calibration and determines the channel specific gain corrections. The version used in this test (DEC06) failed on all solutions and was not actually used.
8) **Imager**
   Obit task Imager. This step takes the data, applies the calibration and editing tables and then images, deconvolves and optionally does phase and/or amplitude and phase self–calibration on each of the target sources. This produces final self–calibrated, deconvolved, flattened images. A blowup of a region of the image is shown in Figure 2.

IV. VLACal.py

A "pipelined" processing script can be written using the calibration/editing/imaging routines in the Obit/python module VLACal.py. The functions are described below and in more detail in the appendix.

- **VLAAutoFlag**
  Does Automated flagging.
- **VLABPCal**
  Bandbass calibration
- **VLACal**
  Basic Amplitude and phase cal.
- **VLAClearCal**
  Clear previous calibration
- **VLAMedianFlag**
  Data flagging based on deviations from a running median.
- **VLASetImager**
  Setup to run Imager to image/deconvolve/selfcalibrate.
- **VLASplit**
  Write calibrated data

V. Timings

A. Multi-threading

Several of the more CPU intensive operations, such as generating the simulated data, the median window editing and the imaging/deconvolution/self–calibration have been written to allow using multiple cores if available. These were run allowing the maximum of 8 available on mortibus. This results in a very substantial processing speedup.

B. Timing Results

The timing results for the various stages are given in Table I. For operations for which multi-threading was enabled the CPU time as well as the ratio of CPU to real time are also given. Eight cores were available so the maximum possible ratio is 8.

VI. Discussion

It is clear from Table I that imaging and deconvolution are by far the most expensive operations. These tests did not include self calibration but this is expected to increase the processing time by a factor of several. Each cycle of self calibration includes a calculation of the model visibility, a calculation of the self–cal gains, and then reimaging/deconvolution. The cost of this will be dominated by the imaging/deconvolution. The additional time for each loop of self calibration is therefore, very roughly, the time of the initial imaging/deconvolution.

The original single threaded version of MednFlag took 1071 minutes and was clearly too slow to be useful for this size dataset. Therefore, the critical routine was recast to run in parallel threads and reduced the run time to a merely painful level.

Several tests were performed using the imaging task Imager. The first was a simple image with no CLEANing; the imaging parameters resulted in 144 facets needed to cover the field of view and anticipated outliers. The final image was 3599x3599 pixels. The second test was a moderate depth unconstrained CLEAN which used 40 major cycles and 1883 CLEAN components before reaching the minimum flux density.

The most dramatic result from Table I is the dramatic impact of using multiple cores. The various tests runs of Imager had CPU/real time ratios in excess of 5; meaning single core operations would have taken at least a factor of 5 longer. The simple image task would have taken at least 44 hours with a single core rather than the 7 1/4 hours with 8.

In conclusion, it appears tractable to process EVLA datasets of order 100 GByte on a state of the art work station on which a substantial number of cores are available.

Appendix

Following are the documentation for the VLACal python module and the pipeline processing script.

References

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

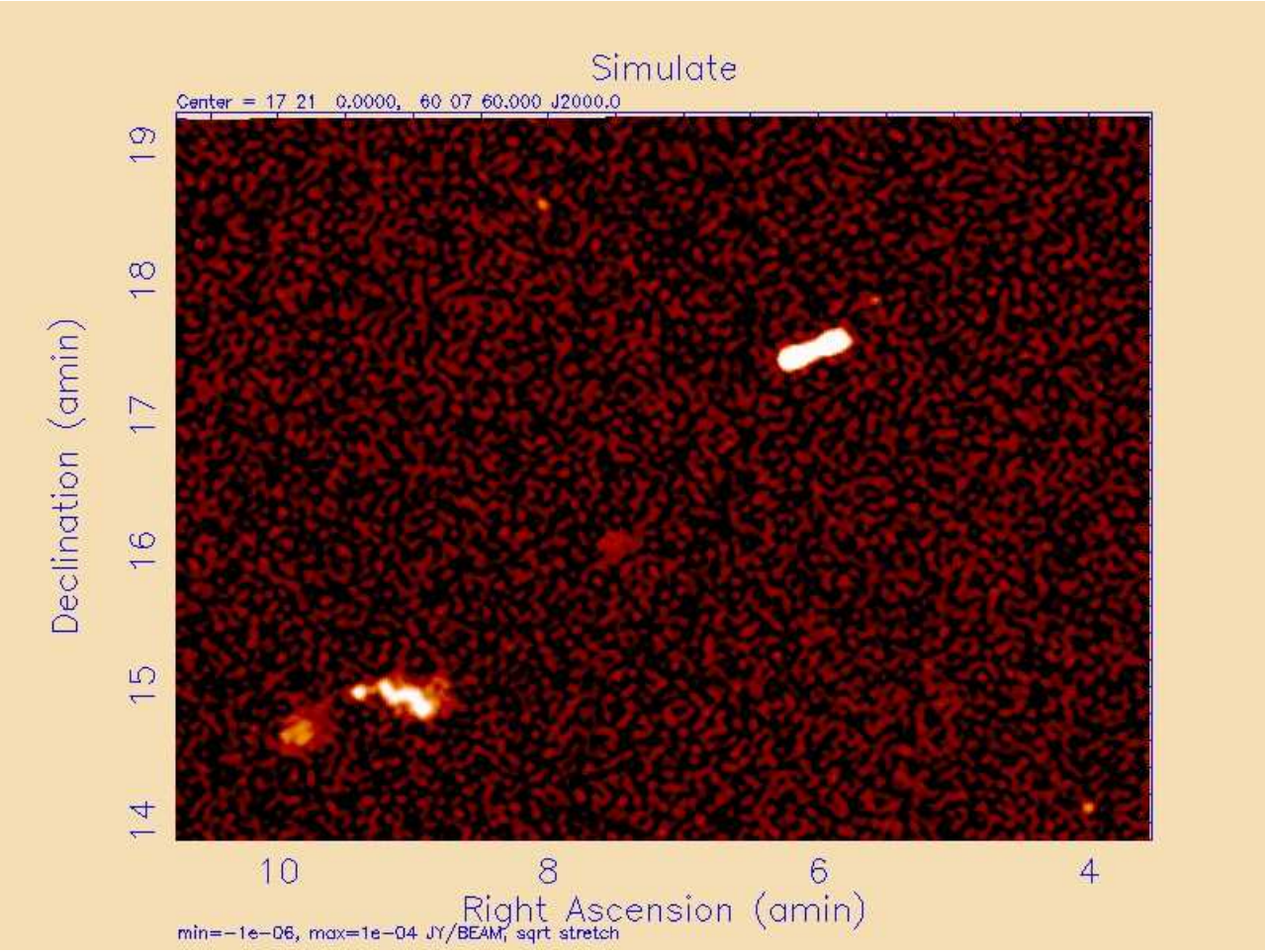

Fig. 2. Region of simulated image. Stretch is the square root of the pixel values intended to show the weaker features.

TABLE I
TIMINGS OF PROCESSING STEPS

Process	Real Time min.	CPU time min.	CPU/Real	Comments
Simulate	574	4017	7.0	Full script run
Median Flag	265	831	3.1	8 threads
SetJy	0.02			Average
Calib	1.3			Amplitude calibrator
Calib	4.6			Phase calibrator
GetJy	0.02			
CLCal	0.15			
AutoFlag	92.0			
BPASS	1.7	1.1	0.68	
Imager	436	2650	6.1	Image only
Imager	1066	6067	5.7	Image + CLEAN

```
The on-line help for VLACal is given in the following.
FUNCTIONS
    VLAAutoFlag(uv, target, err, doCalib=0, gainUse=0, doBand=0,
    BPVer=0, flagVer=-1, flagTab=1, VClip=[0.0, 0.0],
    IClip=[0.0, 0.0], RMSClip=[0.0, 0.0], RMSAvg=0.0, maxBad=0.25,
    timeAvg=1.0, doFD=False, FDmaxAmp=0.0, FDmaxV=0.0, FDwidMW=5,
    FDmaxRMS=[0.0, 0.0], FDmaxRes=6.0, FDmaxResBL=6.0,
    FDbaseSel=[0, 0, 0, 0])

        Does Automated flagging

        Flag data based on any of a number of criteria
        See documentation for task AutoFlag for details
        uv         = UV data object to flag
        target     = Target source name or list of names, blank = all
        err        = Obit error/message stack
        doCalib    = Apply calibration table
        gainUse    = CL/SN table to apply
        doBand     = If >0.5 apply bandpass cal.
        BPVer      = Bandpass table version
        flagVer    = Input Flagging table version
        flagTab    = Output Flagging table version
        VClip      = If > 0.0 VPol (clipping level, fract. amp)
        IClip      = If > 0.0 IPol (clipping level, fract. amp)
        RMSClip    = Abs and fractional clip levels for
                     Time domain RMS filtering
        RMSAvg     = Max RMS/Avg for time domain RMS filtering
        maxBad     = Maximum fraction of baselines for
                     correlator or antenna to be
                     flagged before all are flagged
        timeAvg    = Flagging interval (min)
        doFD       = do frequency domain editing?
        FDmaxAmp   = Maximum average amplitude
        FDmaxV     = Maximum average VPol amp
        FDwidMW    = Width of the median window
        FDmaxRMS   = Channel RMS limits
        FDmaxRes   = Max. residual flux in sigma
        FDmaxResBL = Max. baseline residual
        FDbaseSel  =  Channels for baseline fit (start, end, increment,IF)

   VLABPCal(uv, BPCal, err, newBPVer=1, doCalib=2, gainUse=0,
        doBand=0, BPVer=0, flagVer=-1, solInt=0.0, refAnt=0,
        ampScalar=False, specIndex=0.0, noScrat=[])

        Bandbass calibration

        Do bandbass calibration, write BP table
        uv       = UV data object to calibrate
        BPCal    = Bandpass calibrator, name or list of names
        err      = Obit error/message stack
        newBPVer = output BP table
        doCalib  = Apply calibration table, positive=>calibrate
        gainUse  = CL/SN table to apply
        doBand   = If >0.5 apply previous bandpass cal.
        BPVer    = previous Bandpass table (BP) version
        flagVer  = Input Flagging table version
        solInt   = solution interval (min), 0=> scan average
```

```
    refAnt   = Reference antenna
    ampScalar= If true, scalar average data in calibration
    specIndex= spectral index of calibrator (steep=-0.70)
    noScrat  = list of disks to avoid for scratch files

VLACal(uv, target, ACal, err, PCal=None, FQid=1, calFlux=None,
    calModel=None, calDisk=0, solnVer=1, solInt=10.0, nThreads=1,
    refAnt=0, ampScalar=False, noScrat=[])

    Basic Amplitude and phase cal for VLA data

    Amplitude calibration can be based either on a point flux
    density or a calibrator model.
    If neither calFlux nor calModel is given, an attempt is made
    to use the setjy.OPType="CALC" option.
    uv       = UV data object to calibrate
    target   = Target source name or list of names
    ACal     = Amp calibrator
    err      = Obit error/message stack
    PCal     = if given, the phase calibrator name
    FQid     = Frequency Id to process
    calFlux  = ACal point flux density if given
    calModel = Amp. calibration model FITS file
               Has priority over calFlux
    calDisk  = FITS disk for calModel
    solnVer  = output SN table version
    solInt   = solution interval (min)
    nThreads = Number of threads to use
    refAnt   = Reference antenna
    ampScalar= If true, scalar average data in calibration
    noScrat  = list of disks to avoid for scratch files

VLAClearCal(uv, err, doGain=True, doBP=False, doFlag=False)
    Clear previous calibration

    Delete all SN tables, all CL but CL 1
    uv       = UV data object to clear
    err      = Obit error/message stack
    doGain   = If True, delete SN and CL tables
    doBP     = If True, delete BP tables
    doFlag   = If True, delete FG tables

VLAMedianFlag(uv, target, err, flagTab=1, flagSig=10.0, alpha=0.5,
    timeWind=2.0, doCalib=0, gainUse=0, doBand=0, BPVer=0,
    flagVer=-1, nThreads=1, noScrat=[])

    Does Median window flagging

    Flag data based on deviations from a running median
    See documentation for task MednFlag for details
    uv       = UV data object to flag
    target   = Target source name or list of names, blank = all
    err      = Obit error/message stack
    flagTab  = Output Flagging table version
    flagSig  = Flagging level (sigma)
    alpha    = Smoothing parameter
    timeWind = Averaging window (min)
```

```
        doCalib  = Apply calibration table
        gainUse  = CL/SN table to apply
        doBand   = If >0.5 apply bandpass cal.
        BPVer    = Bandpass table version
        flagVer  = Input Flagging table version
        nThreads = Number of threads to use
        noScrat  = list of disks to avoid for scratch files

    VLASetImager(uv, target, outIclass='', nThreads=1, noScrat=[])
        Setup to run Imager

        return Imager task interface object
        uv       = UV data object to image
        target   = Target source name or list of names
        outIclass= output class
        FQid     = Frequency Id to process
        nThreads = Number of threads to use
        noScrat  = list of disks to avoid for scratch files

    VLASplit(uv, target, err, FQid=1, outClass='      ')
        Write calibrated data

        uv       = UV data object to clear
        target   = Target source name source name or list of names
        err      = Obit error/message stack
        FQid     = Frequency Id to process
```

The following python script was used for processing the simulated data.

```
# Process 100 GByte Obit test
# Data in user 105 on Mortibus
import OErr, OSystem, UV, AIPS, FITS

user = 105   # AIPS user number
############################## Initialize OBIT ###################################
# On Mortibus define disk areas
adirs = [ \
    "/export/data_1/MORTIBUS_1",      \
    "/export/data_1/MORTIBUS_2",      \
    "/export/data_1/MORTIBUS_3",      \
    "/export/data_1/MORTIBUS_4",      \
    "/export/data_2/obit/MORTIBUS_5", \
    "/export/data_2/obit/MORTIBUS_6", \
    "/export/data_2/obit/MORTIBUS_7", \
    "/export/data_2/obit/MORTIBUS_8"  \
    ]
fdirs = ["/export/data_1/FITS","/export/data_2/FITS"]

# Init Obit
err  = OErr.OErr()
ObitSys=OSystem.OSystem ("Process100G", 1, user, len(adirs), adirs, \
                         len(fdirs), fdirs, True, False, err)
OErr.printErrMsg(err, "Error with Obit startup")
```

```
# Setup AIPS, FITS
AIPS.AIPS.userno = user
disk = 0
for ad in adirs:
    disk += 1
    AIPS.AIPS.disks.append(AIPS.AIPSDisk(None, disk, ad))
disk = 0
for fd in fdirs:
    disk += 1
    FITS.FITS.disks.append(FITS.FITSDisk(None, disk, fd))

############################## Set Processing parameters ##########################
from VLACal import *

# Set processing parameters

# Data
disk        = 8           # Input data AIPS disk number
seq         = 1           # Input data AIPS sequence number
inName      = "Simulated2"# Input data AIPS name
inClass     = "100G"      # Input data AIPS class

# Specify calibrators/targets
PCal        = "Cal"       # Phase calibrator
ACal        = "Amp"       # Amplitude calibrator
BPCal       = "Amp"       # Bandpass calibrator
specIndex   = 0.0         # Spectral index of BPCal
targets1    = ["Target"]  # targets
refAnt      = 14          # Reference antenna
calFlux     = 10.0        # Amplitude calibrator flux density

# Processing
nThreads    = 8           # Number of threads to use when possible
noScrat     = [1,2,3,4]   # Disks to avoid for scratch

# Editing parameters
timeWind    = 2.0         # Window width for median window editing
flagVer     = 1           # Prior flagging table to apply
IClip       = [1.0,0.1]   # If > 0.0 IPol (clipping level, fract. amp)
VClip       = [0.1,0.05]  # If > 0.0 VPol (clipping level, fract. amp)
RMSAvg      = 20.0        # Max RMS/Avg for time domain RMS filtering
timeAvg     = 2.0         # Time domain flagging  interval (min)

# Imaging
FOV         = 0.75        # Field of view radius in deg
xCells      = 1.5         # Cell spacing in RA (asec)
yCells      = 1.5         # Cell spacing in Dec (asec)
UVRange     = [0.0,45.0]  # UV range (klambda)
Robust      = 0.0         # Imaging weight
OutlierDist = 1.2         # Distance (deg) to which  outlying fields
OutlierFlux = 0.001       # Minimum outlier apparent flux density
logFile     = "Imager.log"# Imager logging file, blank = none

# CLEAN
Niter       = 15000       # Maximum number of CLEAN components
Gain        = 0.10        # CLEAN loop gain
minFlux     = 50.0e-6     # Minimum CLEAN flux density
Beam        = [5.0,5.0,0.]# CLEAN restoring beam size (asec,asec,deg)
autoWindow  = True        # Use auto window
```

```
 Self calibration
maxPSCLoop  = 0           # No phase self cal
maxASCLoop  = 0           # No amp+phase self cal

################################## Process ####################################
# Define data
uv1 = UV.newPAUV("raw", inName, inClass, disk, seq, True, err)

# Delete any prior calibration
VLAClearCal(uv1, err, doGain=True, doFlag=True, doBP=True)
OErr.printErrMsg(err, "Error resetting calibration")

# Median window editing
VLAMedianFlag (uv1, "    ", err, timeWind=timWind, flagVer=flagVer, noScrat=noScrat)

# Amp & phase Calibrate
VLACal (uv1, targets1, ACal, err, PCal=PCal, \
        calFlux=calFlux, nThreads=nThreads, \
        refAnt=refAnt, noScrat=noScrat)
OErr.printErrMsg(err, " Error in calibration")

# More editing
VLAAutoFlag (uv1, targets1, err, RMSAvg=RMSAvg, flagVer=flagVer, \
             IClip=IClip, VClip=VClip, timeAvg=timeAvg)

# Bandpass calibration
VLABPCal(uv1, BPCal, err, refAnt=refAnt, specIndex=specIndex, noScrat=noScrat)

# Image
img = VLASetImager(uv1, targets1, outIclass="I100G",
    nThreads=nThreads, noScrat=noScrat)
img.FOV         = FOV          # Field of view radius in deg
img.xCells      = xCells       # Cell spacing in RA
img.yCells      = yCells       # Cell spacing in Dec
img.UVRange     = UVRange      # UV range
img.OutlierDist = OutlierDist  # Distance (deg) to which  outlying fields
img.OutlierFlux = OutlierFlux  # Minimum outlier apparent flux density
img.Robust      = Robust       # Imaging weight
img.Niter       = Niter        # Maximum number of CLEAN components
img.Gain        = Gain         # CLEAN loop gain
img.minFlux     = minFlux      # Minimum CLEAN flux density
img.Beam        = Beam         # CLEAN restoring beam size
img.autoWindow  = autoWindow   # Use auto window
img.maxPSCLoop  = maxPSCLoop   # Phase  self cal
img.maxASCLoop  = maxASCLoop   # Amp+phase self cal
img.dispURL     = "None"       # No display
img.logFile     = logFile      # Imaging log file
img.g                          # Image/deconvolve
```