Summary of the CASA 6.2 Cube Imaging Parallelization Refactor

For CASA 6.2 (5.8), cube imaging within the synthesis module has been re-wired. The following are details of the development and tests performed by the CASA Imaging Team.

Code Design

Goals :

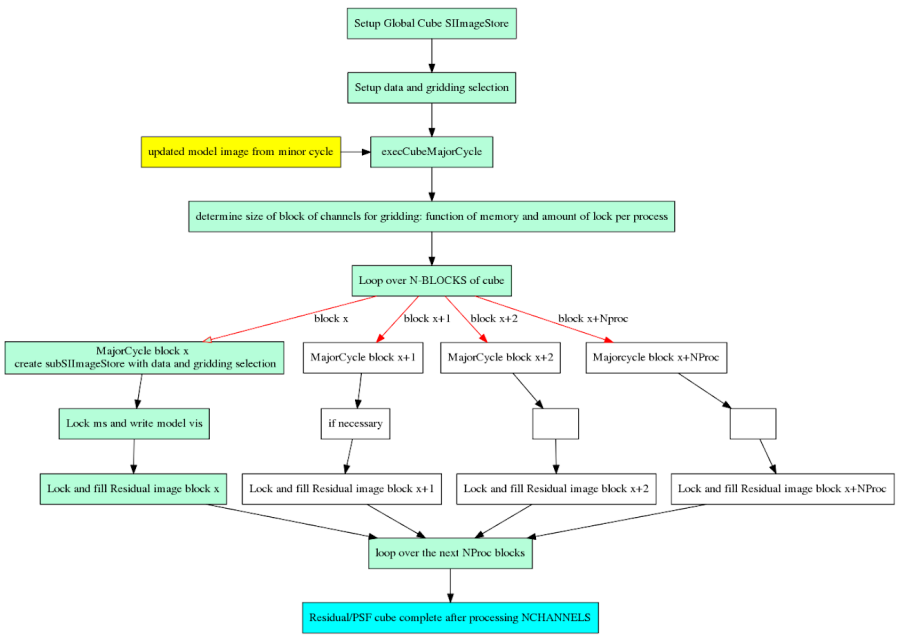
(1) Remove redundant code and make serial runs be the same as parallel with n=1. Ensure same code is used for serial and parallel (to help with maintainence).

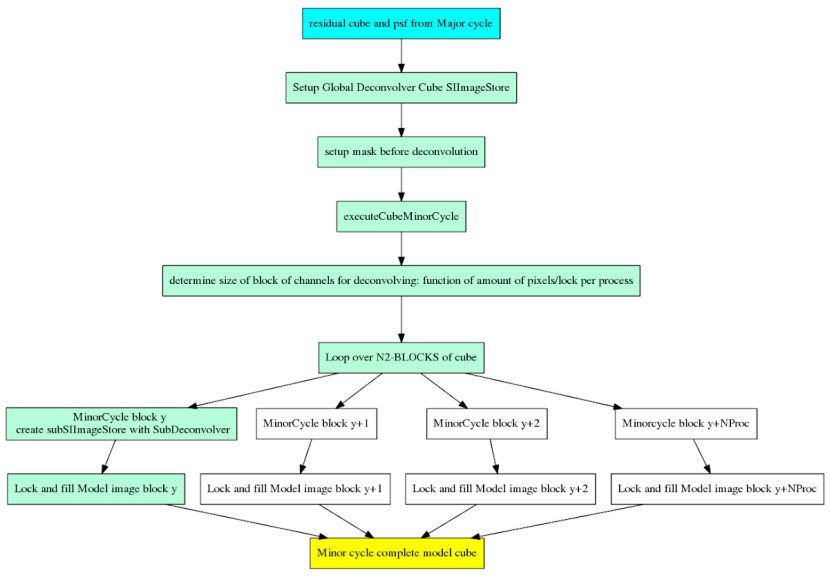
(2) Eliminate refconcat cubes

  (a) Eliminate the need for special refconcat image handling (such as a separate serial restoration step at the end)

  (b) Allow subsequent image-analysis task to operate efficiently

The major and minor cycle modules of PySynthesisImager are initialized and destroyed at all major/minor cycle boundaries. Earlier, gridders and deconvolvers were initialized once and kept alive through the run (holding on to memory) but now, they release resources at each major/minor cycle boundary. This design allows customization (if needed later) of resource allocation for major and minor cycles separately. However, there is a runtime increase associated with restarting/reinitializing modules at every major/minor cycle boundary. Parallelization uses MPI from C++.





Major Cycle

- Initial setup:
    1) Data selection
    2) Image definition
    3) Other choices e.g type of gridder
- Determine how many channels can be gridded (block) due to memory/number of processors available (default use memory of machines or user can set in .casarc)
- Loop over blocks i
- Setup block i
    1) Data selection for block i
    2) Make a reference copies of block i out of original big cube
    3) use same other choices for gridder etc as in initial setup
- Grid block i and make subset residual/psf of block i
    - Lock and write model vis if requested
    - Lock and write subset channels of block i to original big cube psf/residual

Using MPI within C++ : We now can ensure that the blocks in the stages mentioned above are performed in the same fashion. In serial it is the same process that loop over blocks i. In parallel the master sends each processor a block to process and write back results

Minor Cycle

Note: The code that does deconvolution for cube always sent only one plane to deconvolution at a time i.e only one plane worth of memory is kept in memory in serial. For hogbom that means 3 images worth in memory (1 psf + 1 residual + 1 model). For multiscale 3 scales that means a total of equivalently 13 planes worth of memory is used (3 scaled psfs, 3 convolved residual, 1 model image, 6 FFT of scaled psfs) per channel.

Effectively that will remain the case in serial and in parallel the deconvolver will use the amount of memory used in serial multiplied by the number of processes used.

By design we are no longer using independent images per process (i.e virtual image concat to allow for restart with different number of processes). So the model image while deconvolution is proceeding in parallel can be updated by multiple processes. Locks are used to prevent race conditions. Now if there are many small planes in frequency (say 10000) but small in X, Y (say 200x 200) and one has many processes. Each processes wants to lock when after deconvolving the small X,Y image and this can limit speedup in deconvolution as processes will wait for lock more than doing deconvolution. On the cluster at DSOC it was found that go away from this problem if each processor is given at least 8 million pixels to perform deconvolution on before updating the model image (what is labelled “pixels/lock per process” in the above diagram). So to achieve this each process is given a small section of the model image (containing enough planes that has more than 8 million pixels) as temporary copy which they update during the process of deconvolution and lock the model cube and copy back their section only at the end.

A new CubeMinorCycle method in SynthesisDeconvolver

(1) **Iteration Control** : The iteration controller that evaluates cyclethreshold and checks stopping criteria operates on the single large cube, in-between the major cycle and deconvolution steps. The same cycleniter/cyclethreshold information passed into all deconvolvers

(2) **Automasking** : Included within the deconvolvers and therefore it is parallelized as well. (Initial timing tests without automasking parallelization showed a clear need to parallelize this step as well). It required adding the temporary positive-mask image to the list of ImageStore variables so that it got properly saved to disk and re-used upon re-start of the deconvolvers.

Performance Characterization

Runtimes, memory use and numerical consistency were compared between CASA 5.6 release, CASA-6-master and CAS-9386. For timing mpi CASA is used with n=5,9,17. To test the numerical results n=0 (serial CASA), n=1 and n=9 are used. Note that n=1 only works for CASA_9386.

Timing

The following plots show separate timings for major cycles, deconvolution as well as a full run with and without automasking.

The timing tests require that all runs (and CASA versions) execute the same number of iterations and number of major cycles. Therefore, these tests pick a loop gain of 0.0001 and employ cycleniter to ensure a consistent number of iterations on which to do basic timing tests. (Without this, the casa5 and casa6-master parallel runs will choose different niter and cycleniter than serial and 9386 serial/parallel. The plots below are therefore meant to demonstrate basic runtimes and show proof of speedup without unexpected amortization. For real-world numerical runs, please look at the Numerics section below which include both timings as well as threshold-based iteration controls.)

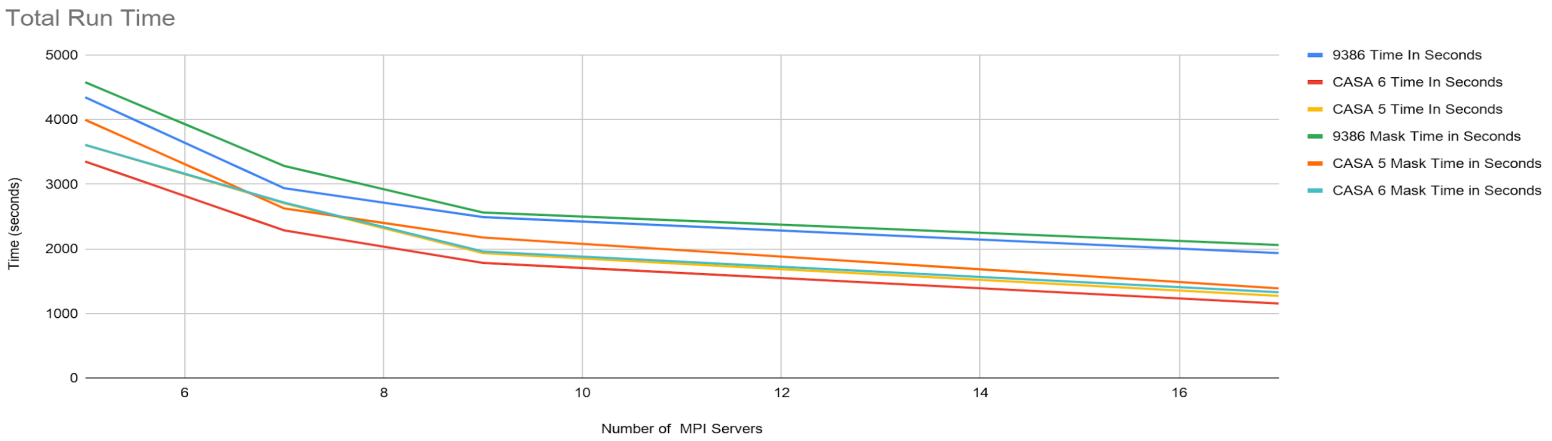
CASA 6 = CASA_6.1.0-62 (https://casa.nrao.edu/download/distro/casa/releaseprep/casa-6.1.0-62.tar.xz)
CASA5 = CASA_5.6.1-8 (https://casa.nrao.edu/download/distro/casa-pipeline/release/el7/casa-pipeline-release-5.6.1-8.el7.tar.gz)
CASA_9386 = The build from the ticket (https://open-bamboo.nrao.edu/browse/CASA-C6DPT10-32/artifact/shared/tarfile/casa-CAS-9386-47.tar.xz)

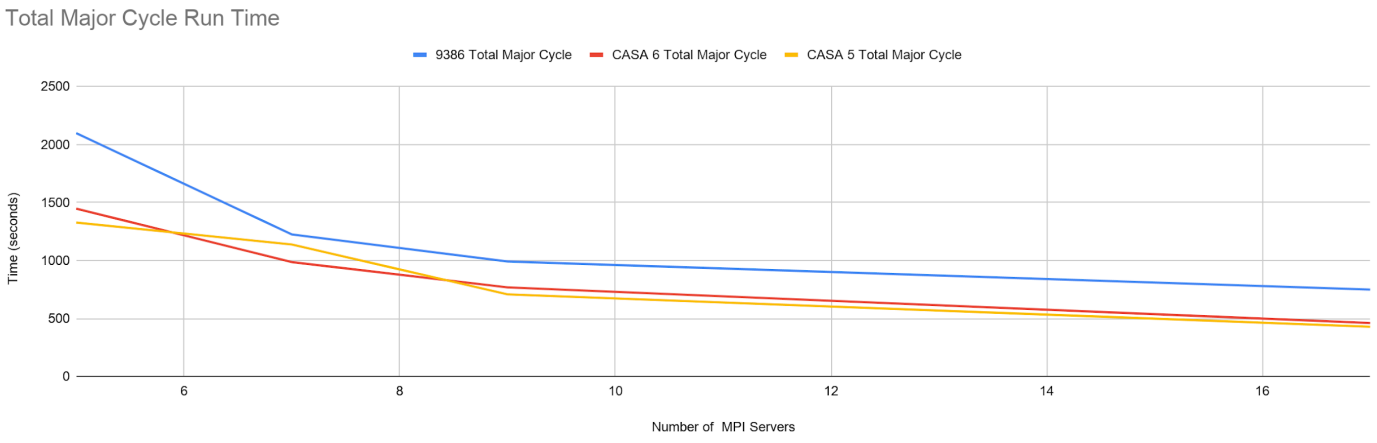
Timing numbers can be found at:
https://docs.google.com/spreadsheets/d/16Ehu5cEFqGISfH5LeNtg81QsYVyGafb3Z77qDJ5cQyk/edit?usp=sharing

Standard Gridder Tclean command:

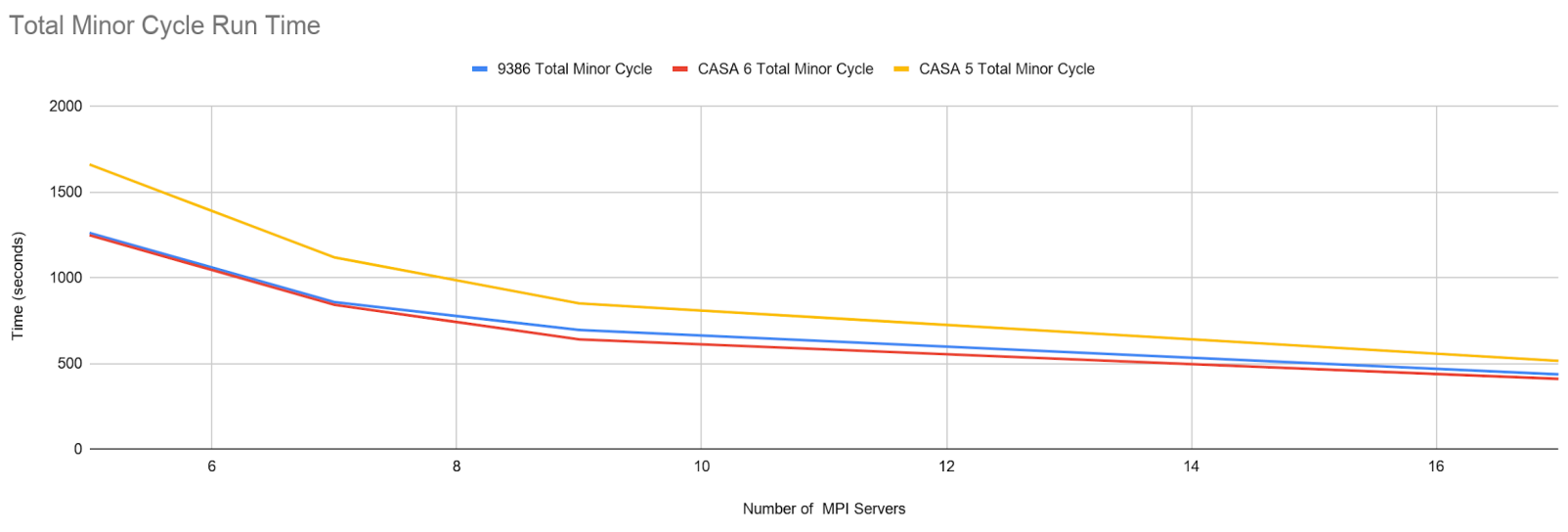
```
vis_name = '/.lustre/cv/users/jsteeb/CASA/CAS_9386/alma_benchmark.ms'

niter = 10514000
cycleniter = 3000
gain = 0.0001
spw = '0'
gridder = 'standard'
perchanweightdensity = True

if casa_version != 'CASA_9386':
    n_chunks = MPI.COMM_WORLD.Get_size() - 1
    niter =     int(10514000/(n_chunks))

tclean(vis=vis_name, spw=spw, imagename=imagename, imsize=[500, 500], cell=['0.03arcsec'], stokes='I', specmode='cube', outframe='LSRK',
perchanweightdensity=perchanweightdensity, gridder=gridder, chanchunks=-1, mosweight=False, usepointing=False, pblimit=0.2, deconvolver='hogbom',
restoration=True, pbcor=True, weighting='briggs', robust=0.5, npixels=0, cycleniter = cycleniter, niter=niter, gain = gain, nsigma=0.0,
usemask='auto-multithresh', sidelobethreshold=2.0, noisethreshold=4.25, lownoisethreshold=1.5, negativethreshold=15.0, minbeamfrac=0.3, growiterations=50,
dogrowprune=True, minpercentchange=1.0, fastnoise=False, restart=True, savemodel='none', calcres=True, calcpsf=True, parallel=True)
```

For CASA5/6 niter must be divided by the number of mpi servers, because the parallelism is implemented such that each mpi server is passed niter. CASA_9386 divides niter by the number of mpi servers.





Our current understanding of the increased major cycle runtime is due to data selection being repeated for every major cycle, instead of just at the beginning of the entire run. It is a fixed increase that is independent of N_Procs.

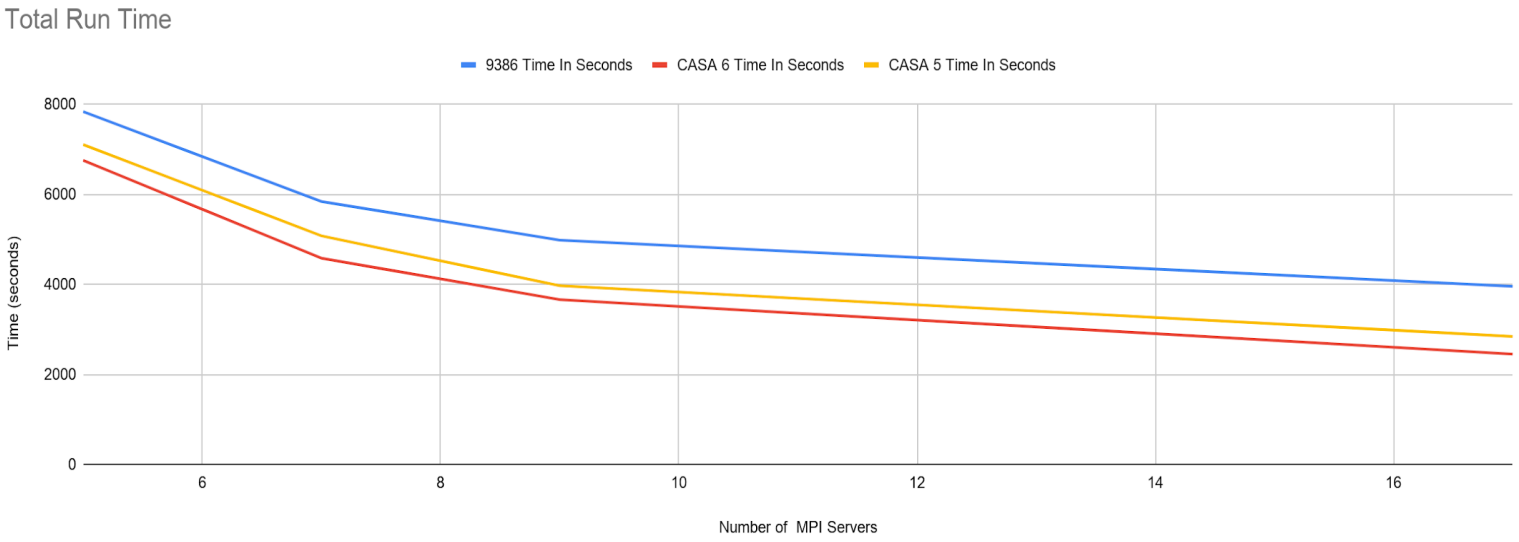


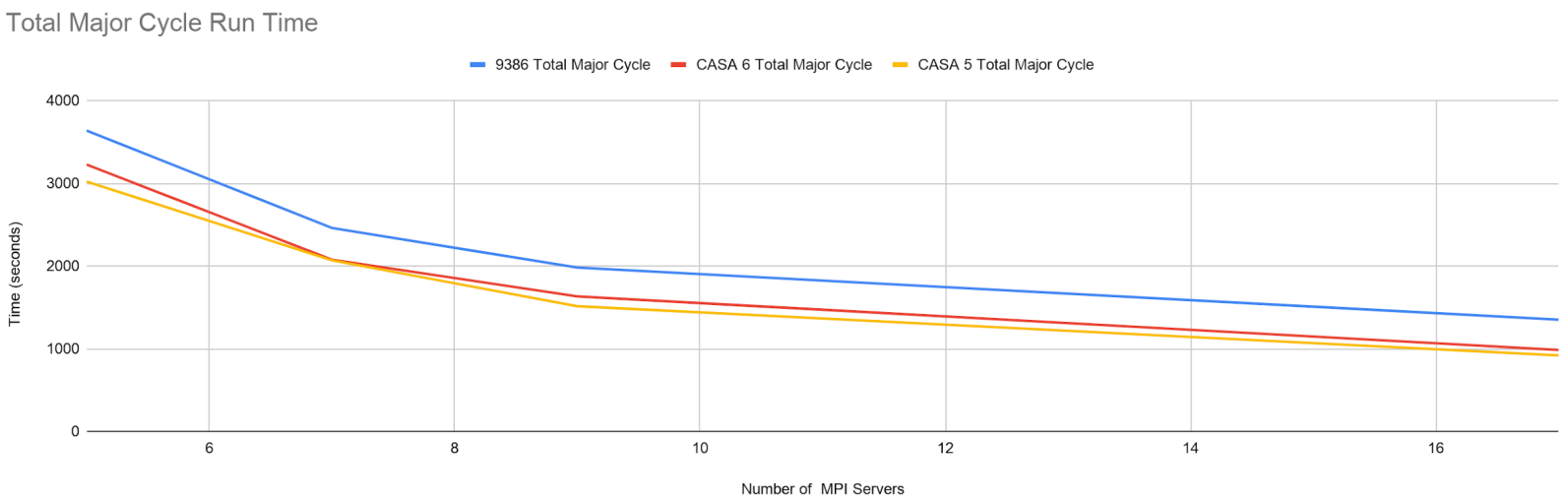
Minor cycle runtime for casa6 appears to be less than for casa5. No idea why.
But, CAS-9386 and CASA6-master are similar for the minor cycle.

Mosaic Gridder Tclean command:

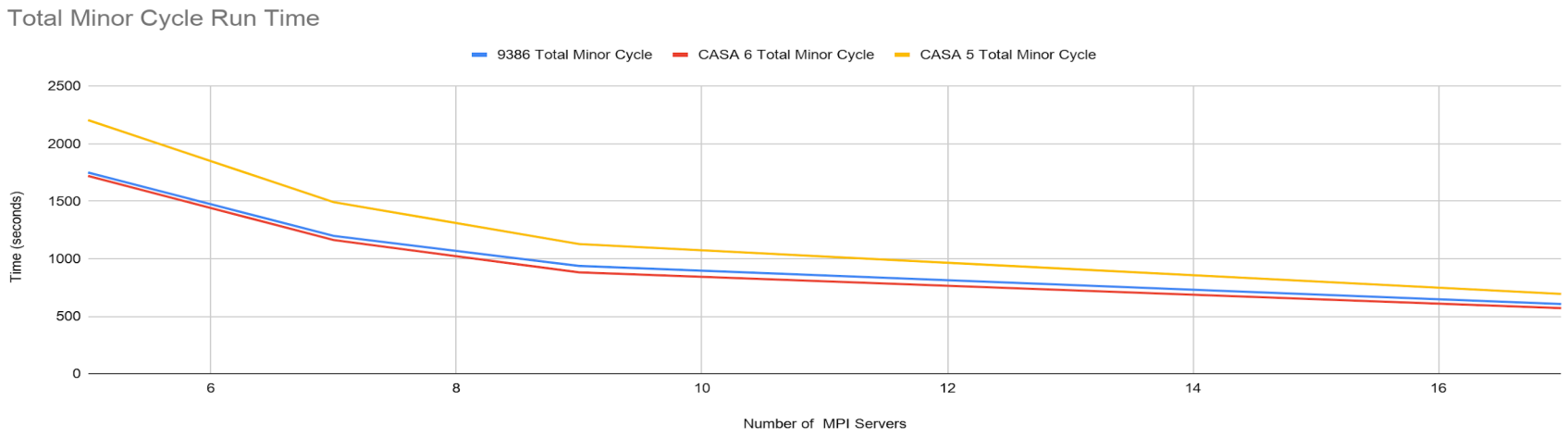
```
vis_name = '/.lustre/cv/users/jsteeb/CASA/data/uid___A002_Xd0adbe_Xa075_target.ms/'

niter = 12000000
cycleniter = 2000
gain = 0.0001
spw = '25'

if casa_version != 'CASA_9386':
    n_chunks = MPI.COMM_WORLD.Get_size() - 1
    niter =     int(niter/(n_chunks))
tclean(vis=vis_name,selectdata=True,datacolumn='corrected',imagename=imagename,imsize=[540, 504],cell=['0.37arcsec'],phasecenter='ICRS 19:37:00.8899
+007.34.09.600',stokes='I',projection='SIN',startmodel='',specmode='cube',gridder='mosaic',deconvolver='hogbom',restoration=True,outlierfile='',weighting='brig
gs',niter=niter,parallel=True,field='B335',spw=spw,timerange='',uvrange='',observation='',intent='OBSERVE_TARGET#ON_SOURCE',reffreq='',width='0.015263
4589239MHz',outframe='LSRK',veltype='radio',restfreq=[],interpolation='linear',perchanweightdensity=False,facets=1,psfphasecenter='',chanchunks=-1,wprojpl
anes=1,vptable='',mosweight=True,aterm=True,psterm=False,wbawp=True,conjbeams=False,cfcache='',usepointing=False,computepastep=360.0,rotatepaste
p=360.0,pointingoffsetsigdev=0.0,pblimit=0.2,normtype='flatnoise',scales=[],nterms=2,smallscalebias=0.0,restoringbeam=[],pbcor=True,robust=1.0,noise='1.0Jy
',gain=gain,cycleniter=cycleniter,threshold=0.0,cyclefactor=0.0,minpsffraction=0.0,maxpsffraction=0.8,interactive=0)
```

For CASA5/6 niter must be divided by the number of mpi servers, because the parallelism is implemented such that each mpi server is passed niter. CASA_9386 divides niter by the number of mpi servers.





Our current understanding of the increased major cycle runtime is due to data selection being repeated for every major cycle, instead of just at the beginning of the entire run. It is a fixed increase that is independent of N_Procs (same as in gridder='standard')



PLWG data set

Large dataset, multiple MS : As more verification tests are run, these additional tests need to be included
- A multiple-MS test (with all parameters set as lists like the pipeline does)
- Some very large cubes, in the regime that previously would require chanchunks > #cores

The following example will test both of those regimes:
2017.1.00661.S, MOUS: uid://A001/X128e/X21a, SB: NGC6334I_a_07_TM2, 2 EBs, 27 pointings
The data and tclean commands from a recent pipeline run for this data set are located at:
/lustre/naasc/sciops/comm/akepley/casa_test/cas9386/test_data_for_devs [1]
For this case it will be particularly important to also assess memory profiling and # of open files, to begin the verification process of whether the refactor is robust in that regime.

The script "2017.1.00661.S_spw25.py" contains 3 sections *iter0(*which calculates psf and residual), *iter1* (starts from *iter0* deconvolves and automasks stopping on threshold) and *restart*(which restart from results of *iter1*)
n0 denotes serial run by casa
n8 denotes parallel run by mpicasa -n 8

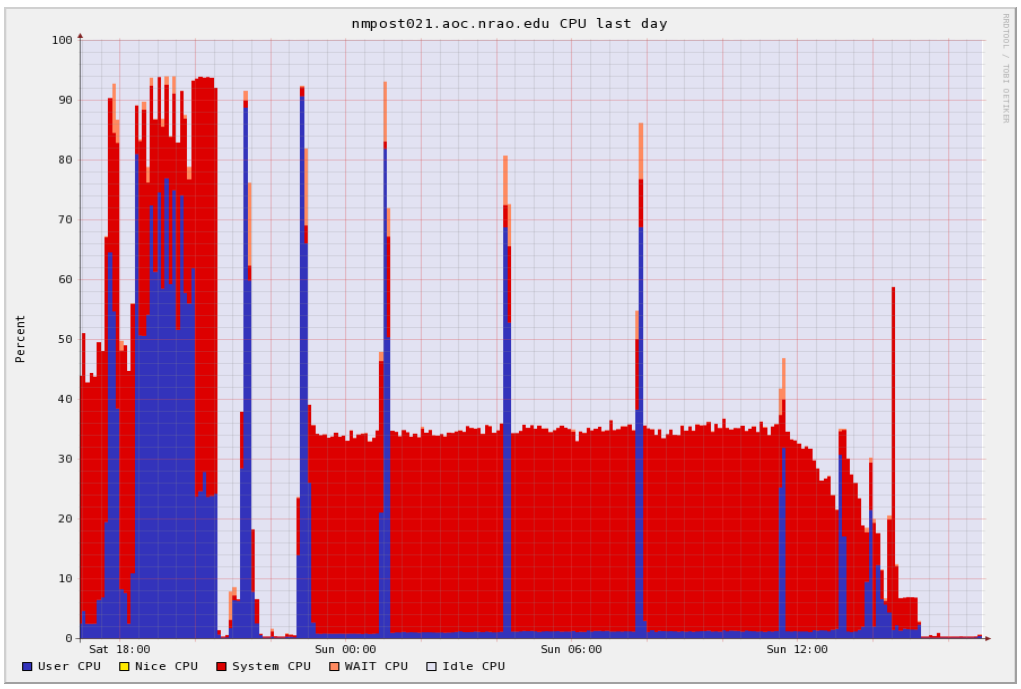
n16 denotes parallel run by mpicasa -n 16

	iter0	iter1	restart
Old code Prerelease v 6.1-59 n0	3700	58300	1400
Old code Prerelease v 6.1-59 n16	1000	56200	2100
Old code Prerelease v 6.1-59 n8			
Cas9386 v 5 n0	3500	16200	2100
Cas9386 v 5 n8	1500	8900	1800
Cas9386 v 5 n16	1200	6500	1800
Cas9386 v6 build49 n0	3500	13200	2200
Cas9386 v6 build49 n8	1700	8400	1900
Cas9386 v6 build49 n16	1300	6100	1800

As can be seen with this dataset and script automasking did not parallelize very well in the old code (grey cells)
In the new code, *after some TempImage fixes in automasking* , there is a sensible speedup.

Note : Despite the slight increase in major cycle runtime due to data selection, there is a net speedup in the overall run in the CAS-9386 code.

Note : Before TempImage memory fixes to automasking, a significant overhead was noticed with a very large number of system reads.
The reason for this can be seen in this plot about CPU usage for the n16 run



All the red is just system request ...here most probably small reads during automasking on lustre TempImages…
We have changed the memory to use by Tempimages in the automasking code :
Iter1 with 16 proc goes down to >6000 and gives some speedup >2 w.r.t serial . where as oin the old code the speedup was ~1.0 (i.e no speedup)

Memory

The number of channels of the image cube to include in each chunk during gridding is automatically decided based on a user-supplied resource constraint in .casarc. This section demonstrates an example without and with this setting, showing that the memory limit is honored. The purpose of this control is to allow multiple jobs to be packed onto individual nodes.

Usage :
Ncores : Mpicasa -n X (this also sets OMP_NUM_THREADS=1)
Memory : .casarc line : system.resources.memory: XXXX (in MB)

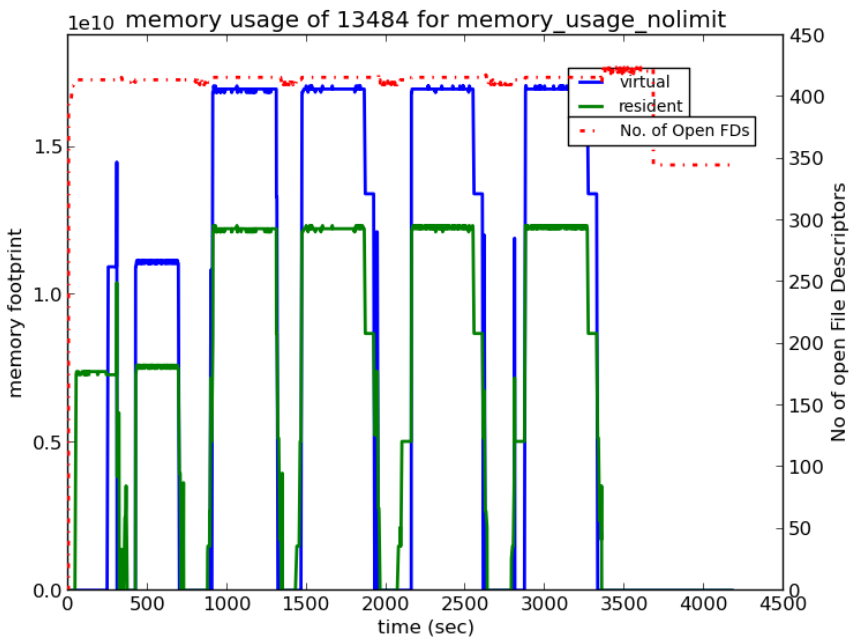
=> It is possible to limit memory to a user-specific value. This memory limit applies to the gridder and uses the chanchunk=-1 calculation from before.

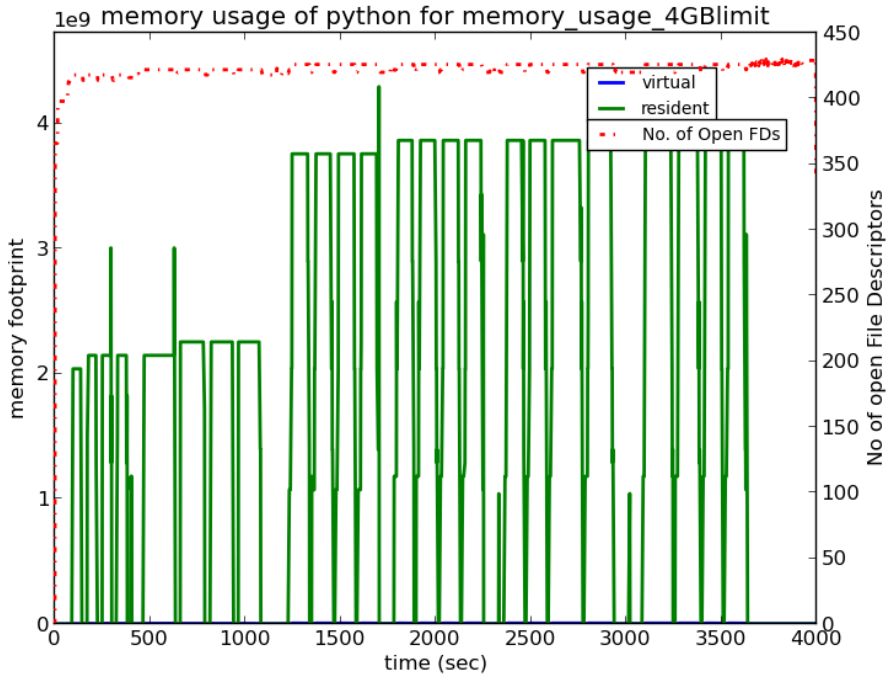
Examples :

Memory Limit Demonstration

Memory limiting test using the .casarc line :: system.resources.memory: 4000

LEFT : No limit : For a cube in serial with 384x384x1918 the maximum memory used is 12.5 GB as can be seen in this plot it had 5 major cycles without the above .casarc line
RIGHT : 4GB limit : When we set the memory line in .casarc we can the maximum memory used is around 4GB...it loops 4 times to do the gridding and degridding in each major cycle.

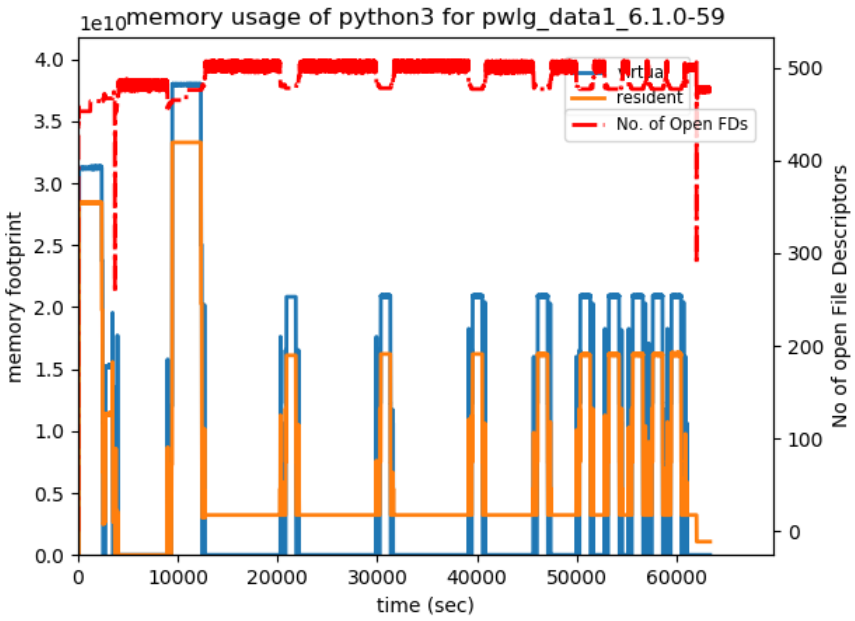


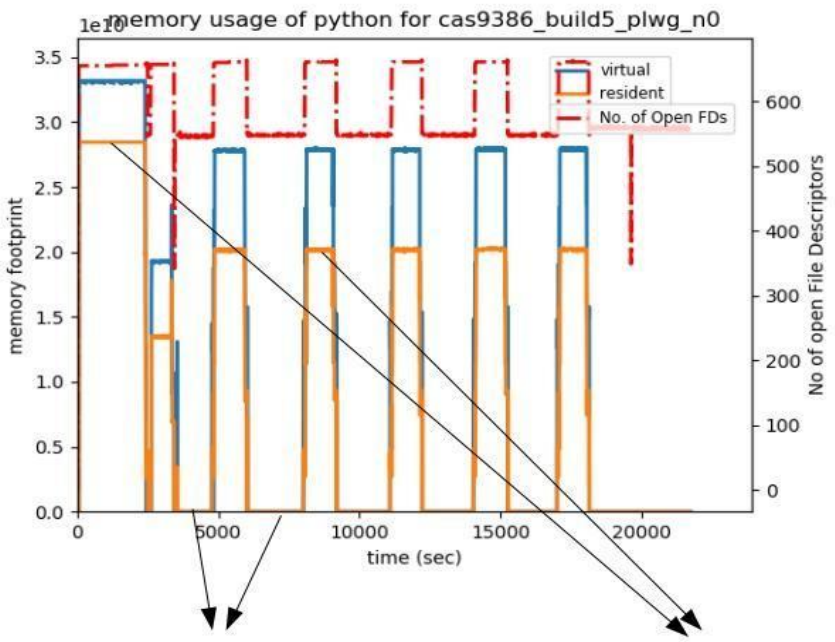


PLWG Data Set : Compare Old/New memory usage

LEFT : Memory profile of serial run for 6.1.0-59 : Maximum memory usage for this data set is ~32 GB
RIGHT : Memory profile for the same script and data for cas9386v6 build 49 : Maximum memory usage here is ~ 28 GB

Note : For both these tests, there was NO memory limit specified in .casarc. The tests were run on a 128GB machine. The purpose was to check that there were no drastic memory use changes between the old code and the CAS-9386 code.





Minorcycle + automask memory usage

Psf and majorcycle
Mem usuage

Numerical Characterization

Goals : Remove two sources of differences

(1) Remove data and weight interpolation/binning errors at chunk/partition boundaries.

(2) Make iteration control view the entire cube for its cyclethreshold and iterdone calculations, instead of each partition getting independent iteration controls. Eliminate the problem of different chunks doing different numbers of iterations and major cycles.
-> parallel cube should follow the same rules as current serial cube.

(3) perchanweightdensity=False psf beam shapes should show no changes at chunk boundaries

Output images were compared for equivalent runs with different builds and nprocesses were compared. E.g. : Show/plot some metric from the output of 'imstat' on the final output image ?

DataSet : The initial numerical tests use the iter1 commands in E2E6.1.00016.S with a subselection of 128 channels.

Info From Amanda : In order to reliably assess the potential effects on ALMA operations, the tclean commands from ALMA pipeline runs should be used. The E2E6.1.00016.S commands for spw=25 and the 2017.1.00717.S commands for spw=33 from a recent pipeline run are located here: /lustre/naasc/sciops/comm/akepley/casa_test/cas9386/test_data_for_devs . Note that the commands give include iter0 (psf and residual creation), iter1 (deconvolution), and the so-called iter2 (set restoring beam). The refactor should make iter2 obsolete since there would be no need to revert to serial mode to find a common beam over the whole cube. In that case you could modify the iter1 command to set restoringbeam='common' and omit the iter2 command.

Commands, CASA builds

Tclean command without mask:

vis_name =’/.lustre/cv/users/jsteeb/CASA/data/uid___A002_Xd0adbe_Xa075_target_spw_25_1900_2027.ms'

tclean(vis=vis_name, field='B335', scan=['6,9,12,15'], intent='OBSERVE_TARGET#ON_SOURCE', datacolumn='data', imagename=imagename, imsize=[540, 504], cell=['0.37arcsec'], phasecenter='ICRS 19:37:00.8899 +007.34.09.600', stokes='I', specmode='cube', outframe='LSRK', perchanweightdensity=**False**, gridder='mosaic', chanchunks=-1, mosweight=**True**, usepointing=**False**, pblimit=0.2, deconvolver='hogbom', restoration=**True**, pbcor=**True**, weighting='briggs', robust=1.0, npixels=0, niter=8000000, threshold='0.0291Jy', nsigma=0.0, interactive=0, restart=**True**, savemodel='none', parallel=parallel)

Tclean command with mask:

tclean(vis=vis_name, field='B335', scan=['6,9,12,15'], intent='OBSERVE_TARGET#ON_SOURCE', datacolumn='data', imagename=imagename, imsize=[540, 504], cell=['0.37arcsec'], phasecenter='ICRS 19:37:00.8899 +007.34.09.600', stokes='I', specmode='cube', outframe='LSRK', perchanweightdensity=**False**, gridder='mosaic', chanchunks=-1, mosweight=**True**, usepointing=**False**, pblimit=0.2, deconvolver='hogbom', restoration=**True**, pbcor=**True**, weighting='briggs', robust=1.0, npixels=0, niter= 8000000, threshold='0.0291Jy', nsigma=0.0, interactive=0, usemask='auto-multithresh', sidelobethreshold=2.0, noisethreshold=4.25, lownoisethreshold=1.5, negativethreshold=15.0, minbeamfrac=0.3, growiterations=50, dogrowprune=**True**, minpercentchange=1.0, fastnoise=**False**, verbose=verbose, restart=**True**, savemodel='none', parallel=parallel)

Column and Row Labels:
n0: CASA without mpi.
n1: CASA with mpi n set to 1. //CASA5 and CASA6 do not work with n=1
n9: CASA with mpi n set to 9 (8 servers).

CASA6 = CASA_6.1.0-62 (https://casa.nrao.edu/download/distro/casa/releaseprep/casa-6.1.0-62.tar.xz)
CASA5 = CASA_5.6.1-8 (https://casa.nrao.edu/download/distro/casa-pipeline/release/el7/casa-pipeline-release-5.6.1-8.el7.tar.gz)
CASA_9386 = The build from the ticket (https://open-bamboo.nrao.edu/browse/CASA-C6DPT10-32/artifact/shared/tarfile/casa-CAS-9386-47.tar.xz)
CASA_9386_5 = A CASA 5 build of CASA_9386 (AOC Lustre /lustre/kgolap/casa-CAS-9386.tgz)

Summary : Current Results on Numerical Consistency between Run Modes

1. For CASA_9386 (the CASA 6 build of the ticket) there is a difference between n0 and nx (x >= 1) run images that is not present for CASA_9386_5 (the CASA 5 build). This is possibly caused by a difference in library selection between the n0 (serial) and parallel runs (nx).
2. When using automasking a difference appears for both CASA_9386 and CASA_9386_5 (not present for n0 and n1 runs). *The cause of this is currently thought to be related to a data-selection (retuning) roundoff difference that occurs only for datasets that have frequency going in descending order. This is unrelated to the cube refactor and will be addressed in a separate ticket (post 6.1)*

Metric 1 : Percentage difference images

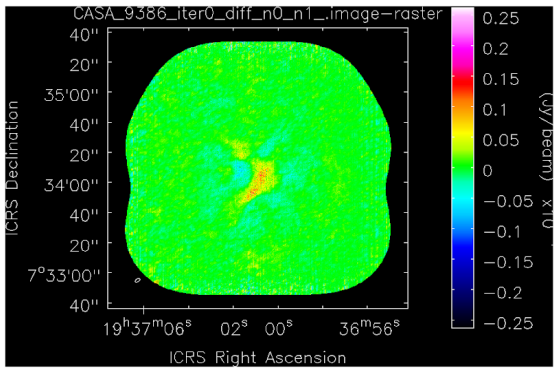
To check the numerical consistency the percentage difference images are created for a single channel from the above the tclean runs.
The immath expression used is expr='100*(IM0-IM1)/MAX(IM0)' . The channels are chosen that have the largest difference.

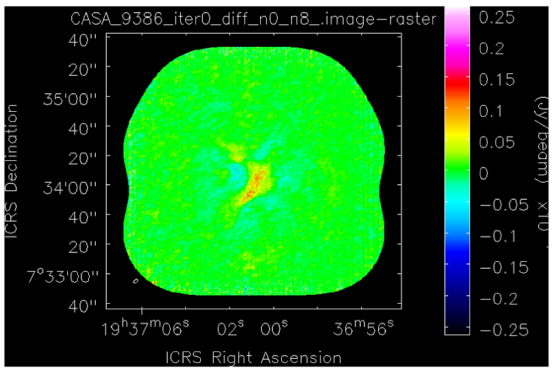
Results for Iter0

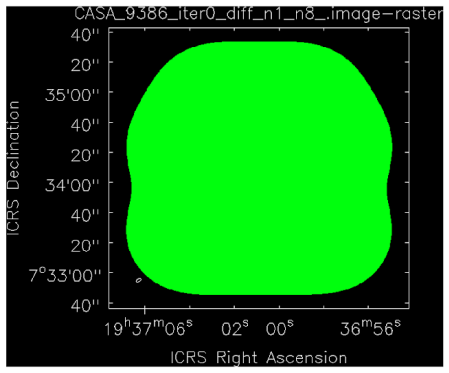
Summary for iter0 (one major cycle) :
=> Based on the CASA_9386_5 results, the application and math code in the major cycle appears perfectly consistent between both kinds of serial runs (n0 and n1) and parallel runs (n9).
=> Only casa6 builds with CASA_9386 show a difference between runs started with ‘casa’ (n0 serial) versus those started with ‘mpicasa’(n1 serial, n9 parallel). This is consistent with the level of difference earlier seen between casa5 and casa6 that might point to library changes.

iter0 runs (used above no mask tclean call with niter=0)

Channel = 122
Plots for CASA_9386: n0 vs n1, n0 vs n9, n1 vs n9 (Jy/beam x 10^-5)



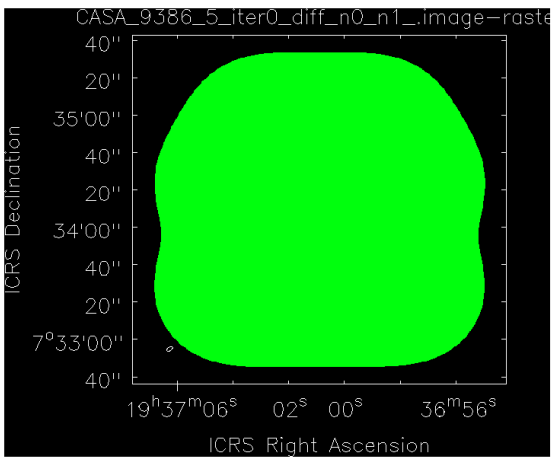


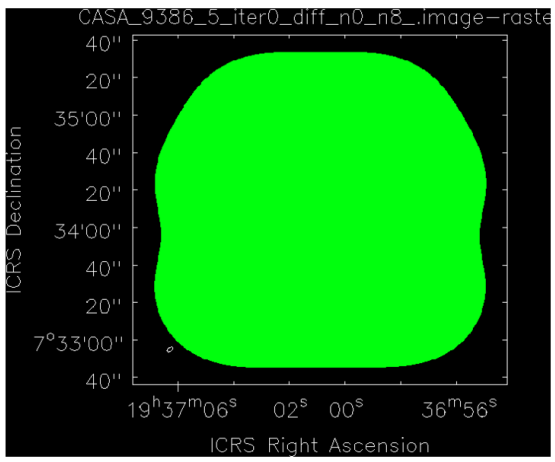


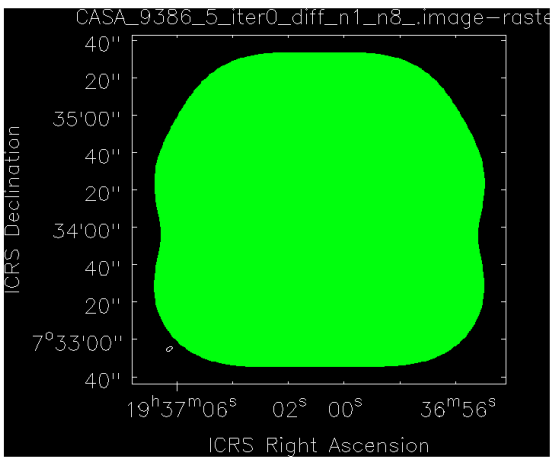
=> With CASA_9386 (casa6 build of 9386), there is a difference (<0.25*10^-5%) between n0 and n>0 (i.e. starting 'casa' versus 'mpicasa') at a similar level as seen with CASA5 and CASA6. All runs with mpicasa (serial n1 and parallel n>1) give identical numbers.

Channel = 122

Plots for CASA_9386_5: n0 vs n1, n0 vs n9, n1 vs n9 **(all 0)**



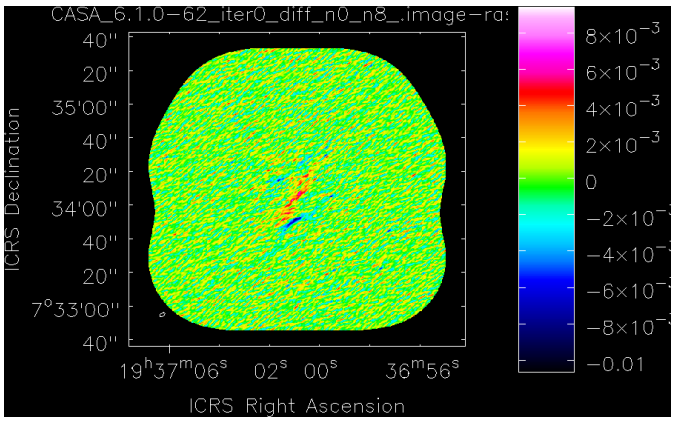




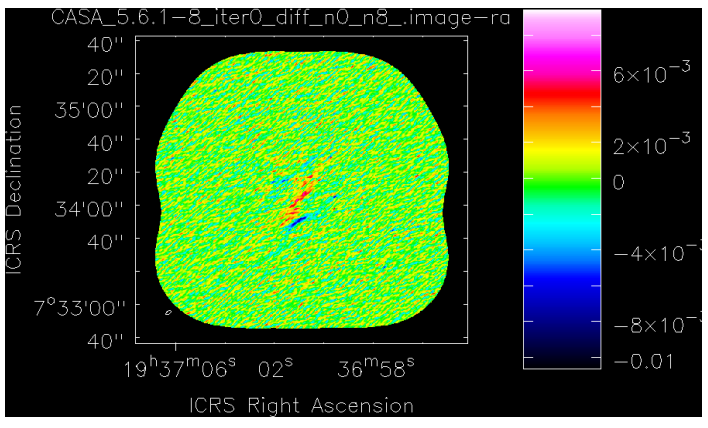
With CASA_9386_5 (casa5 build of 9386) there are no differences between any of the serial vs parallel or casa vs mpicasa runs.

Below are comparisons with the old CASA5 and CASA6 code, where we expect differences because of chunk-boundary interpolation issues and separate iteration control. The purpose of these figures is to show the relative magnitude of serial-vs-parallel numerical differences that are being fixed by CAS-9386.

Plot for CASA6: n0 vs n9



Plot for CASA5: n0 vs n9



Results for Iter 1 (no mask)

Summary for iter1 : No Masking (several major and minor cycles):

=> Based on the CASA_9386_5 results, the application and math code in the major cycle, hogbom minor cycle, and threshold-based iteration control, appears perfectly consistent between both kinds of serial runs (n0 and n1) and parallel runs (n9).

=> Only casa6 builds with CASA_9386 show a difference between runs started with 'casa' (n0 serial) versus those started with 'mpicasa'(n1 serial, n9 parallel). This is consistent with the level of difference earlier seen between casa5 and casa6 that might point to library changes.

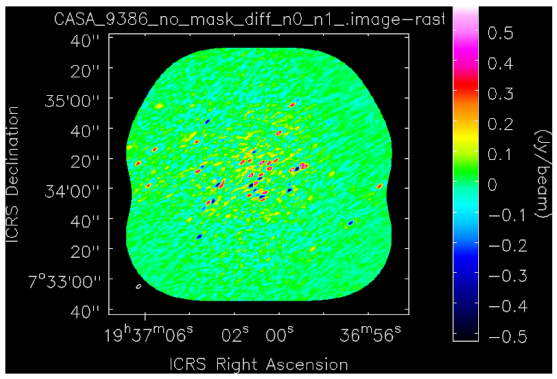
=> The difference after Iter1 is larger than after Iter0. This could be just error propagation, combined with the natural uncertainty associated with using hogbom clean on extended emission (where the choice of 'peak residual location' can be affected by slight (1e-06) differences in pixel values.

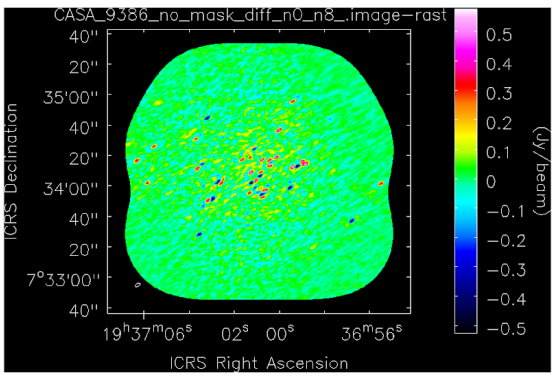
<u>iter1 with no mask</u>

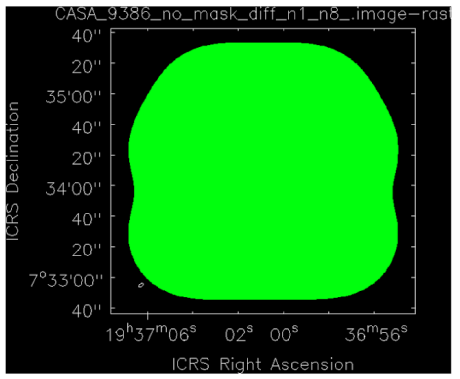
n0 vs n1, n0 vs n8, n1 vs n9

Channel = 112

Plots for CASA_9386: n0 vs n1, n0 vs n9, n1 vs n9



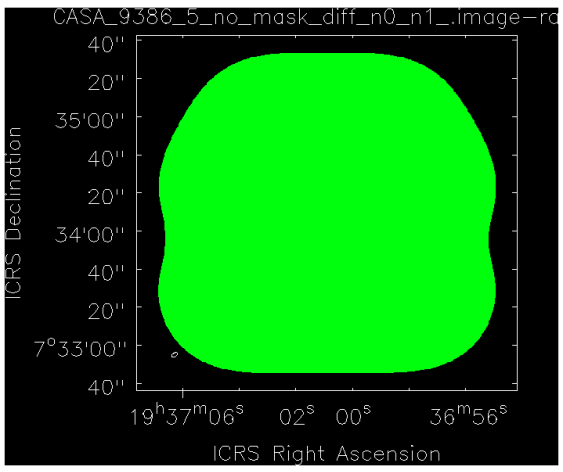


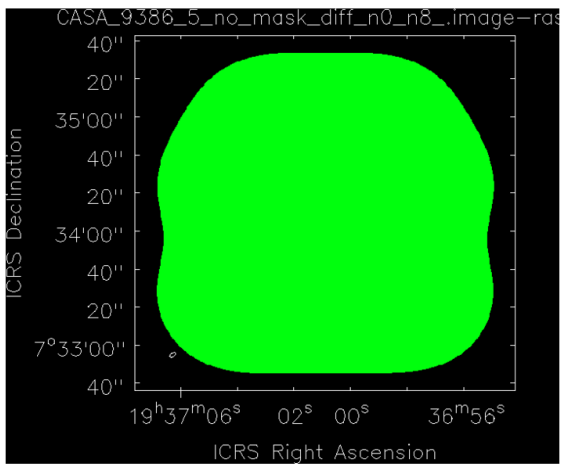


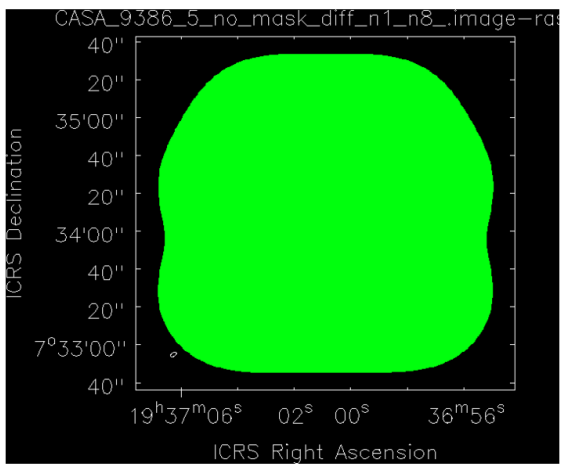
Similar to iter0, for casa6 builds of 9386, there is a difference (< 0.5%) between n0 (casa) and all other runs started with mpicasa (n1 serial, n9 parallel). All runs (serial and parallel) with mpicasa are identical. The difference is because iterations start from different dirty images (from the iter0 runs).

Channel = 112

Plots for CASA_9386_5: n0 vs n1, n0 vs n9, n1 vs n9 **(all 0)**



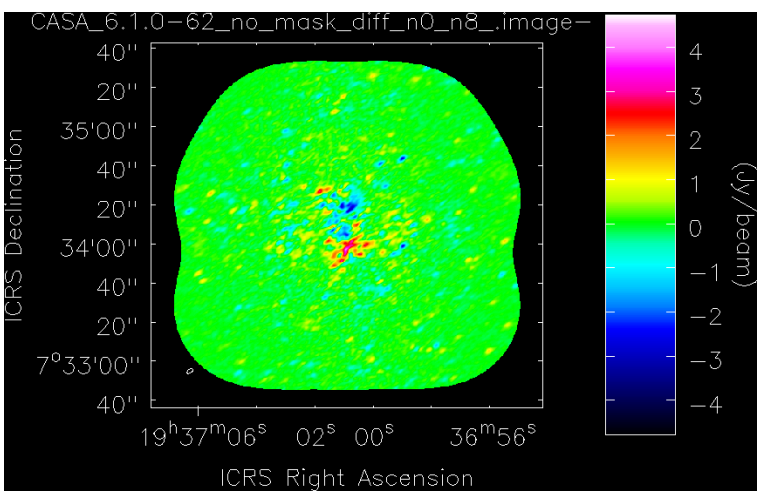




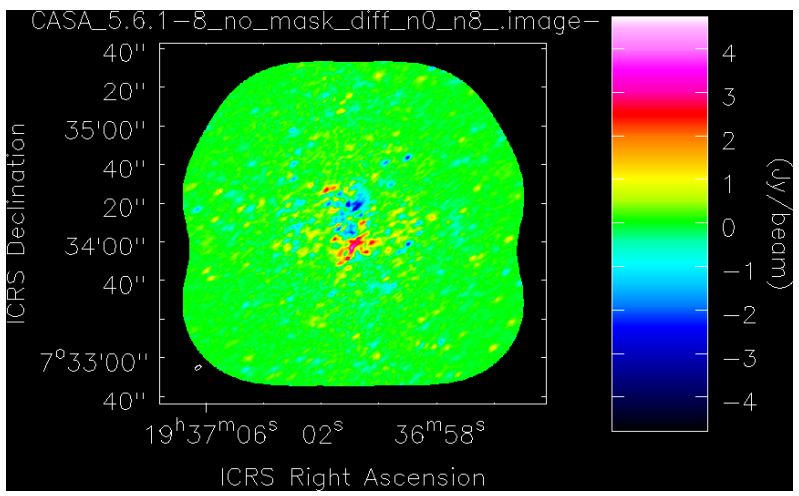
Similar to iter0, casa5 builds of 9386 show identical minorcycle and iteration control behaviour for all runs n0, n1, n9 (and casa vs mpicasa). I.e. iter0 results are identical and so are iter1.

Below are results from old code (CASA5 and CASA6) to illustrate the relative level of serial and parallel differences that the new code in CAS9386 is fixing.

Plot for CASA6 n0 vs n9



Plot for CASA5 n0 vs n9



Results for iter1 with mask

Summary for iter1 : With Automasking (several major and minor cycles):

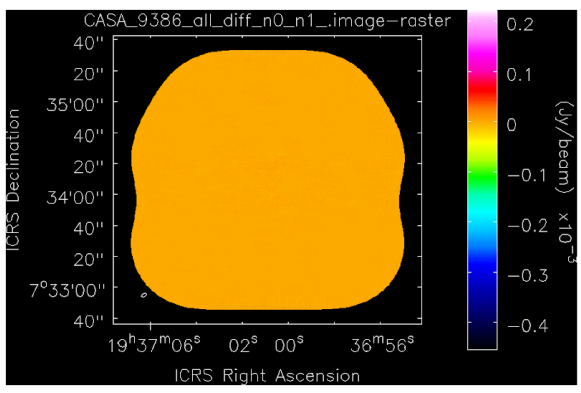
=> Based on results with both casa5 and casa6, there are numerical differences between serial and parallel when automasking is turned on.
=> For both serial runs (casa n0 and mpicasa n1), the behaviour is similar to no-masking, where casa6 shows slight diffs but casa5 gives identical results.

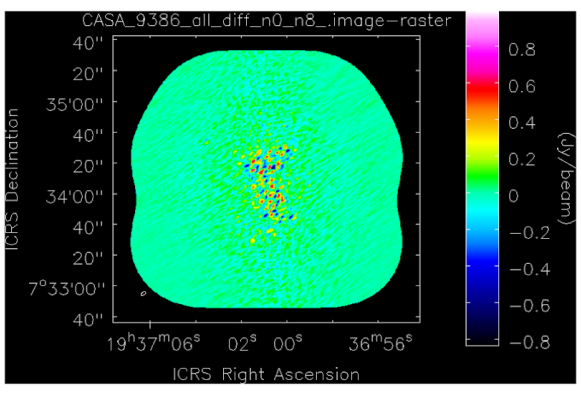
The source of the remaining numerical differences when automasking is used in parallel is currently attributed to what looks like a round-off difference during data selection (more specifically, the re-tuning of selection to optimize data selection runtime per chunk) that occurs only for datasets that have frequency decreasing with increasing channel number. For other datasets (positive freq) this is not seen. Also, if data-selection retuning is turned off, the differences go away (but the runtime increases). This is expected to be a minor difference similar to choosing a slightly different channel width. This issue has likely been present for a long time, and is unrelated to the cube refactor. It will be investigated after the cube refactor work is complete. Based on all our tests so far, this has no effect on scientific validity of the output.

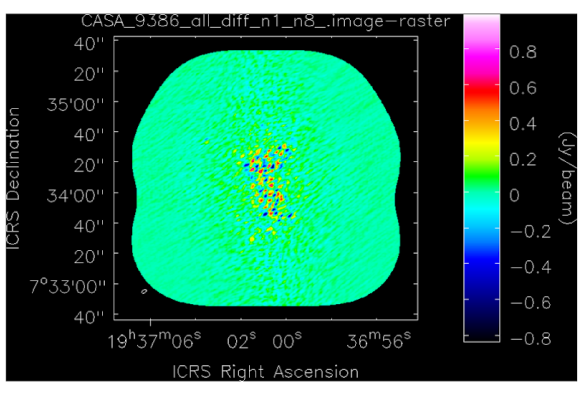
Channel 116 (extended emission)

n0 vs n1, n0 vs n9, n1 vs n9

Channel = 116
Plots for CASA_9386: n0 vs n1, n0 vs n9, n1 vs n9



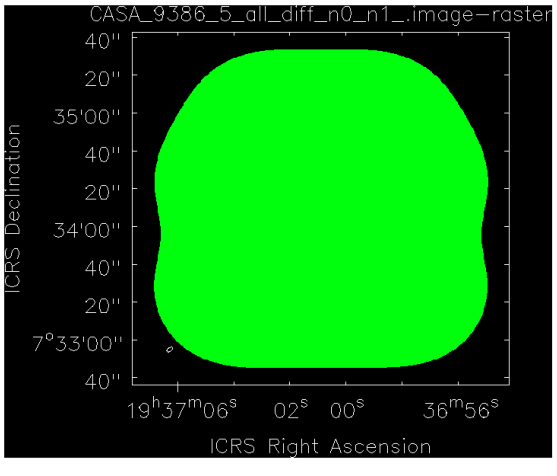


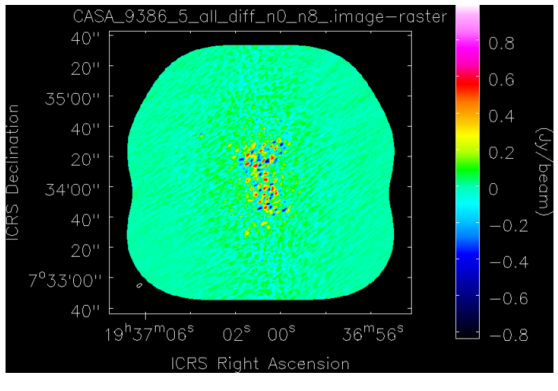


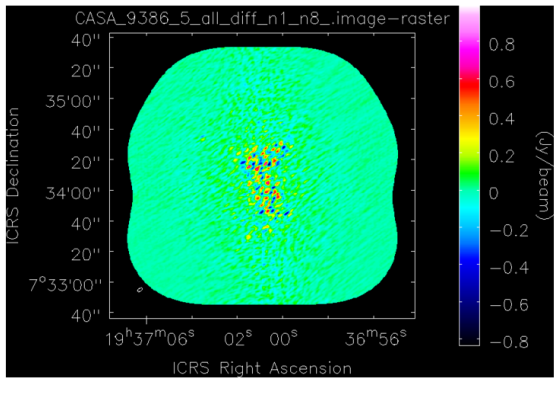
Here, for casa6 build of 9386, n0 and n1 show very small differences (they are non-zero but invisible because of the choice of color scale. They are at the same level as without automasking.

And, there is a much larger difference (<1%) between serial and parallel behaviour when automasking is turned on.

Channel = 116
Plots for CASA_9386_5: n0 vs n1, n0 vs n9, n1 vs n9



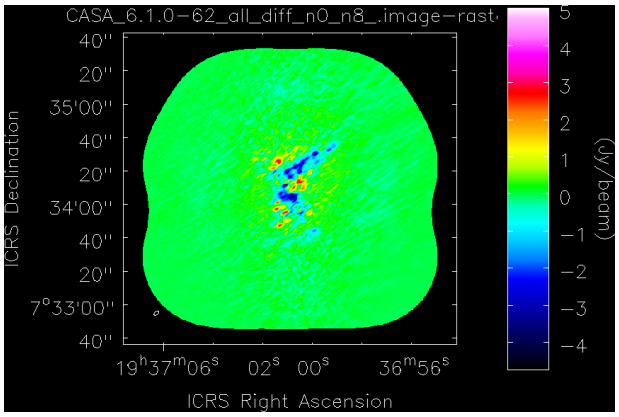




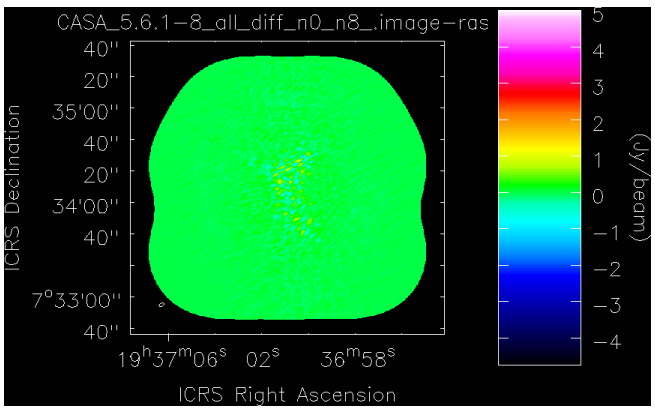
The casa5 build of 9386 shows identical results (zero diff) between n0 and n1 (similar to without masking), but there is a difference between serial (n0,n1) and parallel runs (n9).

Below are for reference, figures for the old code. They illustrate the magnitude of error/differences that are being fixed by the new code in CAS9386.

Plot for CASA6 n0 vs n9



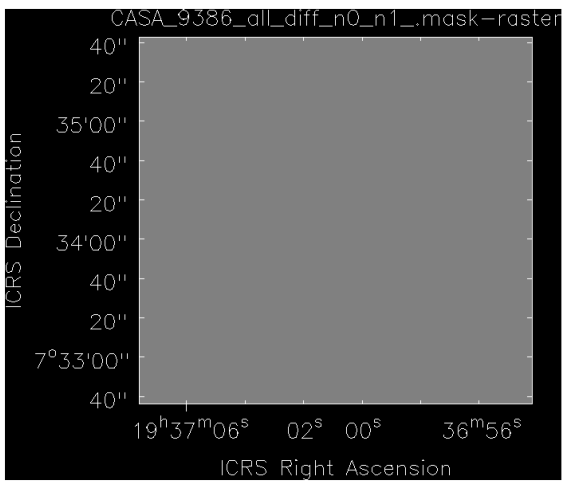
Plot for CASA5 n0 vs n9

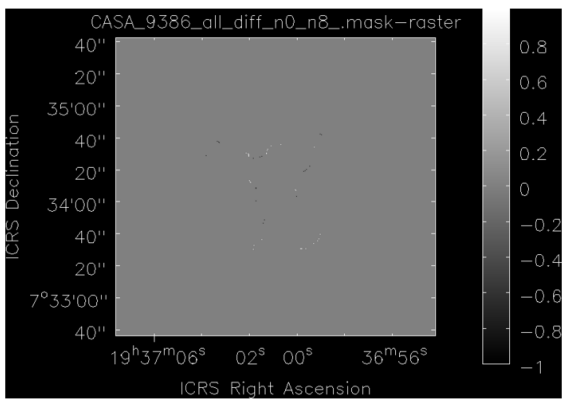


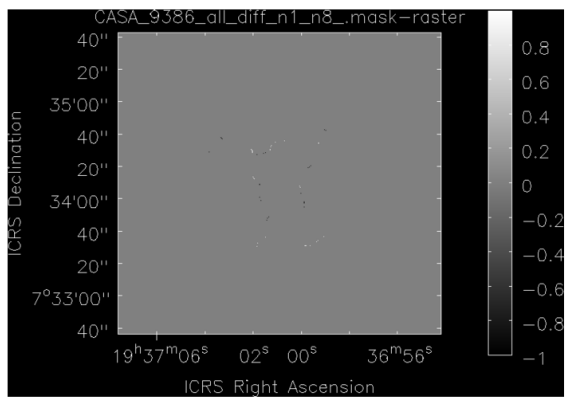
The CASA_9386_5 n0 vs n1 runs have no difference. However, there is still a noticeable difference between the n1 and n9 run.

Mask Diffs :

Below are diffs of the final output MASKS (for the casa6 (or casa5??) runs. Note that all diffs are at the outer boundaries. This is indicative of a slight change in the input residual image, and perhaps not a change in automasking code itself. According to Amanda, in the past, changes in the automasking algorithm/code usually result in systematic boundary changes (growth or shrinkage) and this random positive-negative difference around the edge is more likely due to a difference in the starting residual image. This is consistent with what is noted above w.r.to a slight data-selection difference being the source of numerical difference.



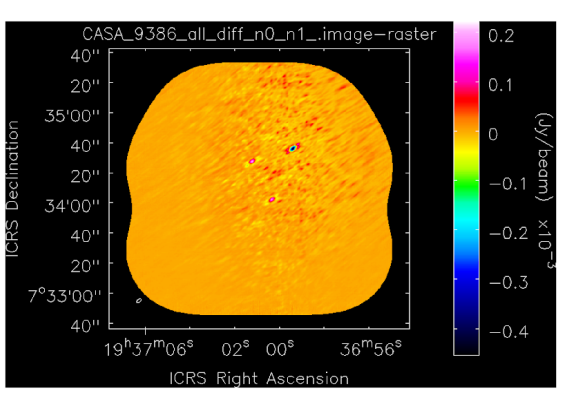


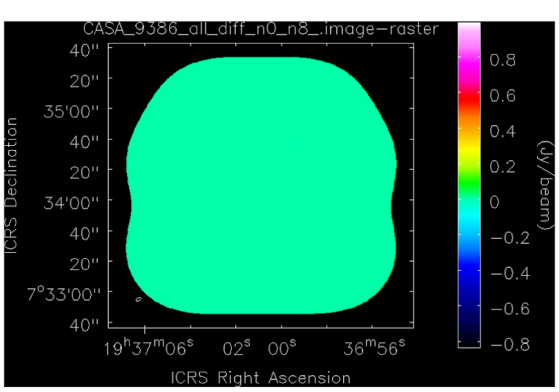


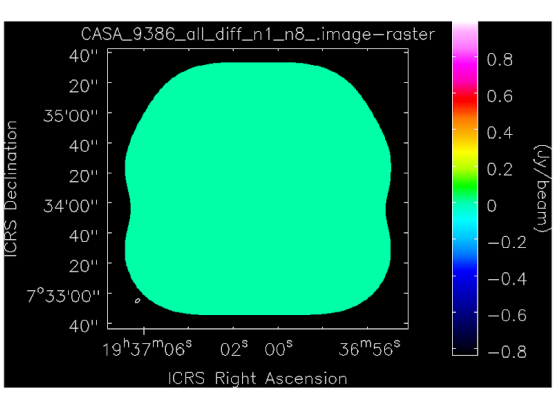
Channel 15 (point sources)

Another channel is included that show a more (visibly) pronounced difference for CASA_9386, but where the automasking difference is actually less :

Channel = 15
Plots for CASA_9386: n0 vs n1, n0 vs n9, n1 vs n9



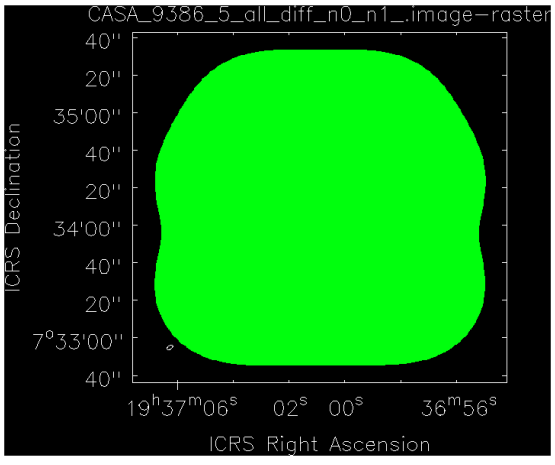


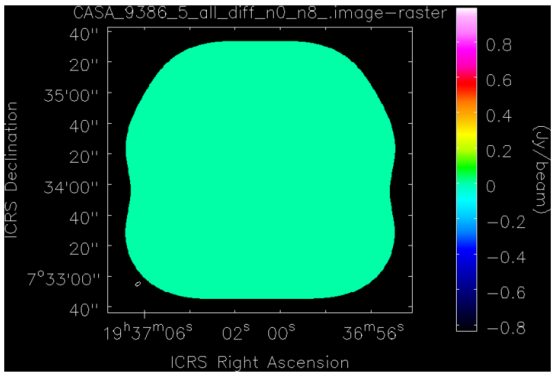


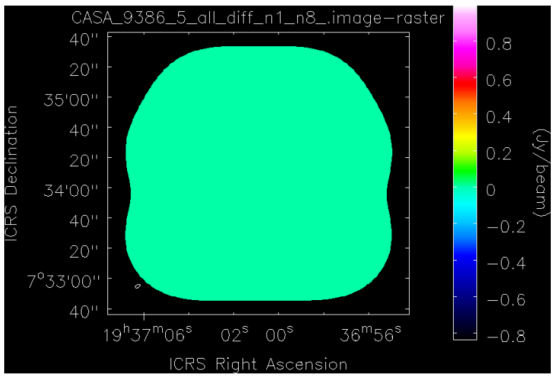
Here, the casa6 difference between casa vs mpicasa is prominent (but please note that it is at <0.4%) and still consistent with the iter0 casa-vs-mpicasa difference.

This (in our understanding) is a point-source dominated channel, where the effect of automasking (and of the uncertainty associated with using hogbom on extended emission) is reduced and therefore the differences between serial and parallel are also reduced.

Channel = 15
Plots for CASA_9386_5: n0 vs n1, n0 vs n9, n1 vs n9







For the casa5 build of 9386, again, no difference between casa vs mpicasa. The serial vs parallel automasking-related differences are similar to casa6.

===> This is additional evidence that in cases where deconvolution uncertainty is itself reduced (i.e. compact emission), and automasking has a negligible effect, the differences are lower.

Metric 2 : Flux Density per channel, Noise per channel, Beam ratio per channel

The following tools/commands were used to construct spectra of flux density, noisel and beam axial ratio. (Scripts from Amanda)

Spectrum plot (flux density and per channel noise):
plotSpectrumFromMask from toddTools.py

Axial Ratio:

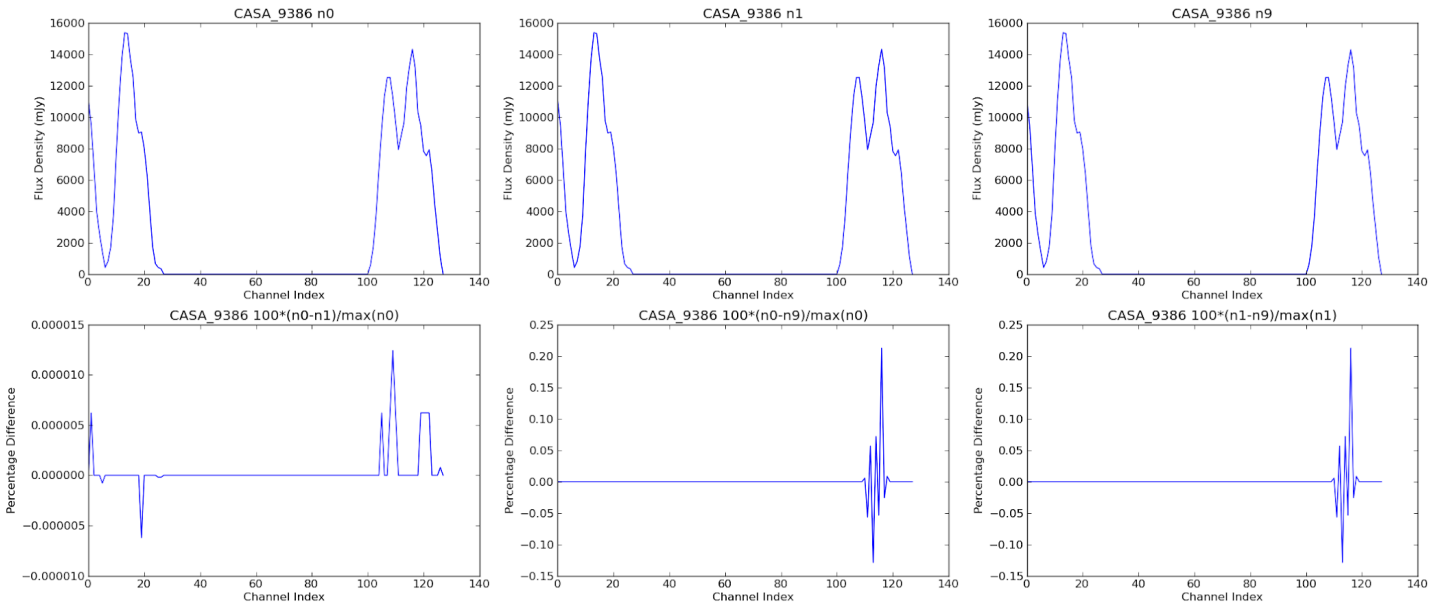
```
for ii in range(0,nchan):
    axmajor = qa.convert(qa.quantity(allbeams['beams']['*'+str(ii)]['*0']['major']), 'arcsec')['value']
    axminor = qa.convert(qa.quantity(allbeams['beams']['*'+str(ii)]['*0']['minor']), 'arcsec')['value']
    areas[ii] = axmajor * axminor
    axratio[ii] = axmajor/axminor
    weight[ii] = numpy.isfinite( axratio[ii] )
```

These metrics were evaluated for all 4 builds being compared.

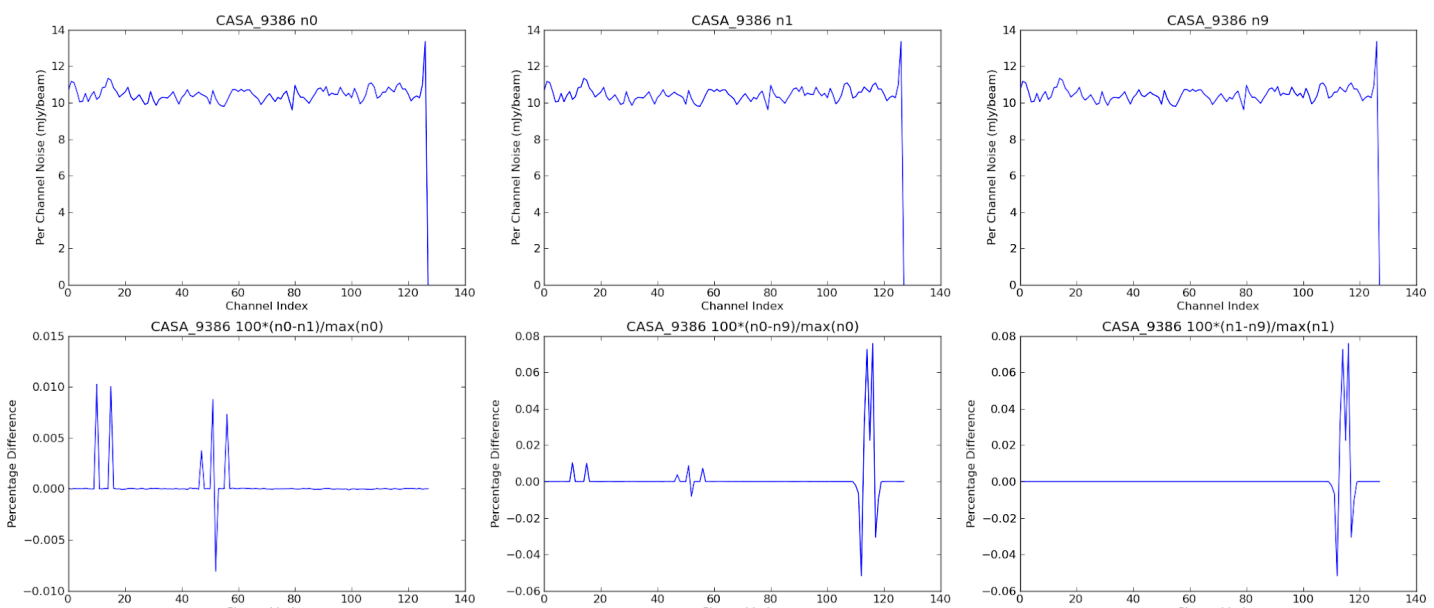
CASA_9386 (casa6 build from 9386)

Differences are a mix of the casa-vs-mpicasa diffs (initial channels) and the serial-parallel automasking diffs (later channels). All less than 0.25%. No diffs in beam ratios.

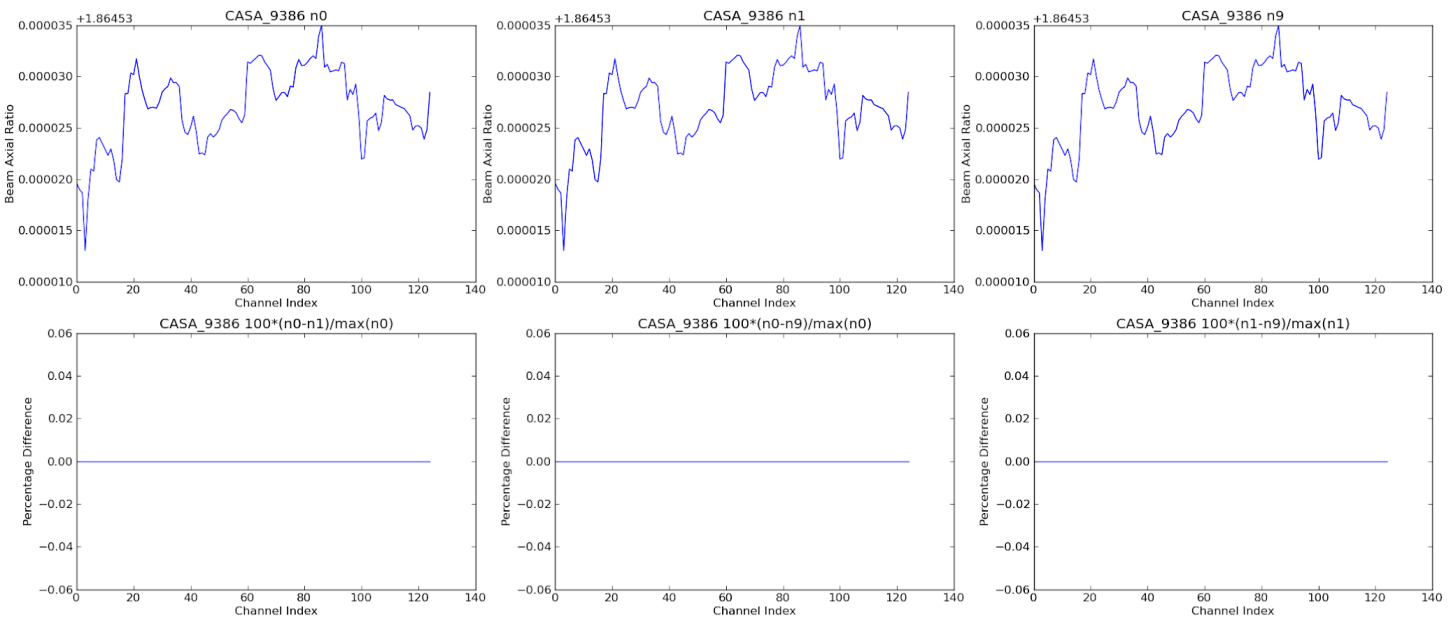
Flux Density plots for CASA_9386: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



Noise per channel plots for CASA_9386: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



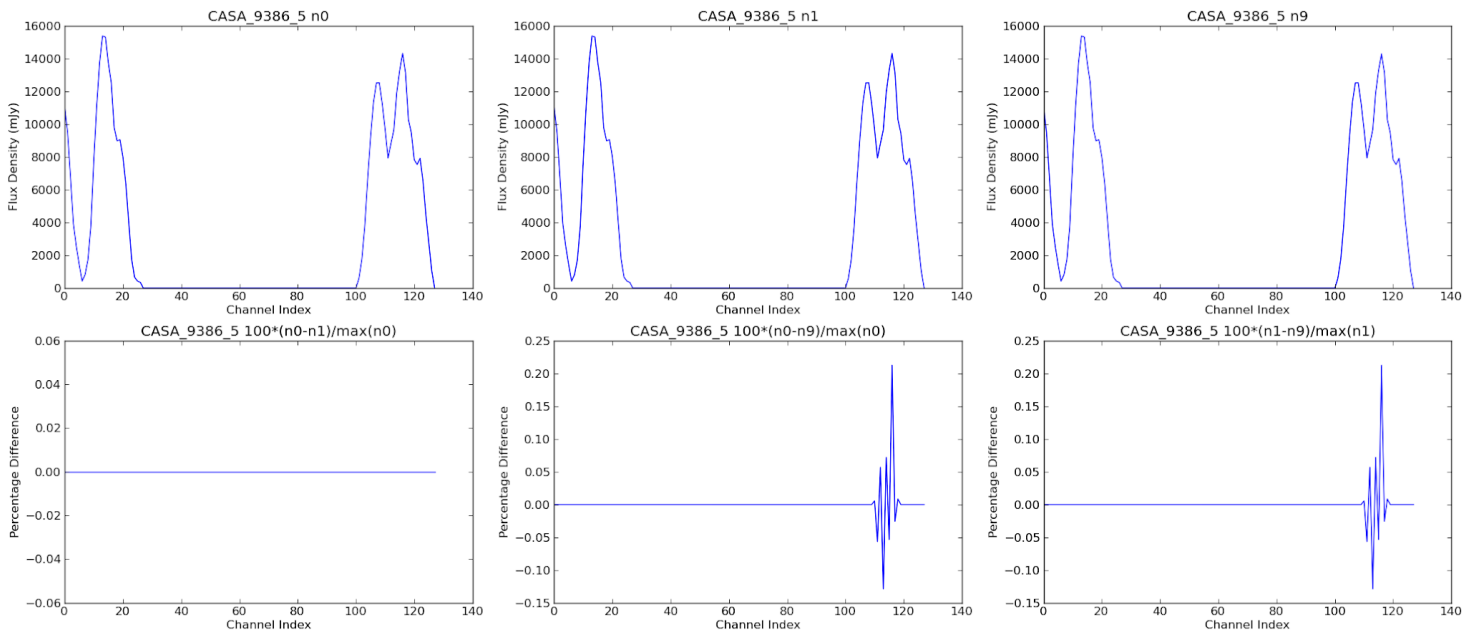
Axial ratio for CASA_9386: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



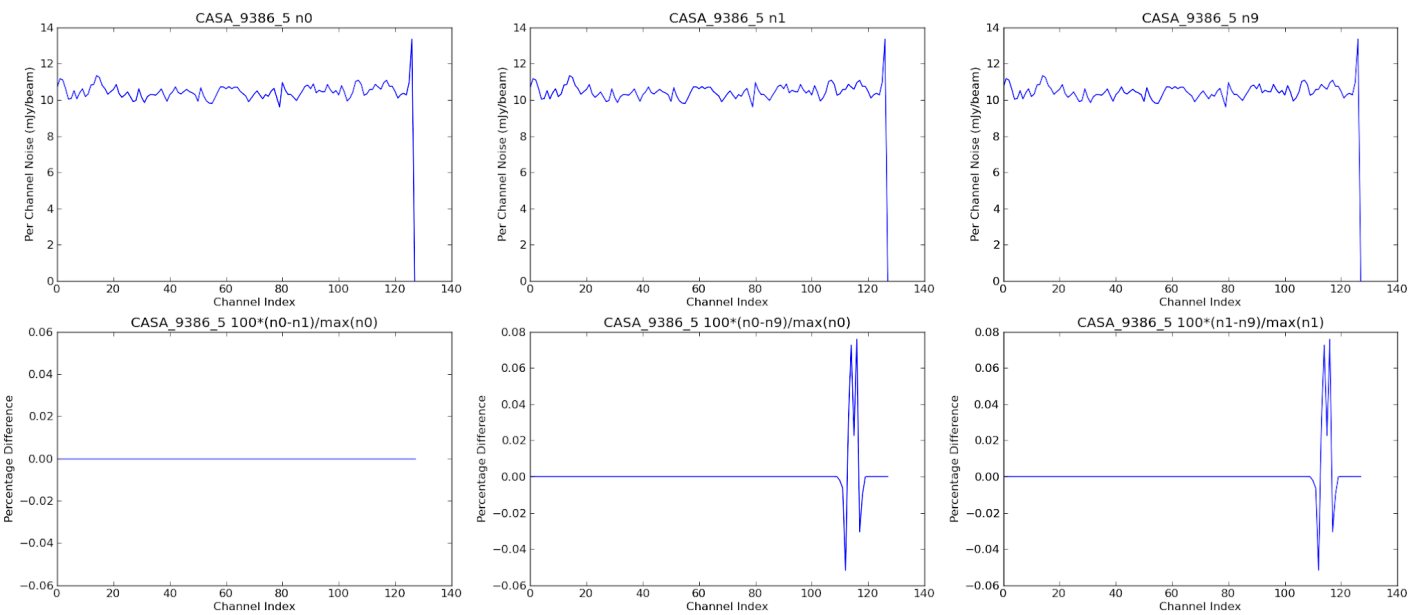
CASA_9386_5 (casa5 build from 9386)

Differences are only from the serial-parallel automasking diffs (later channels). All less than 0.25%. No diffs in beam ratios.

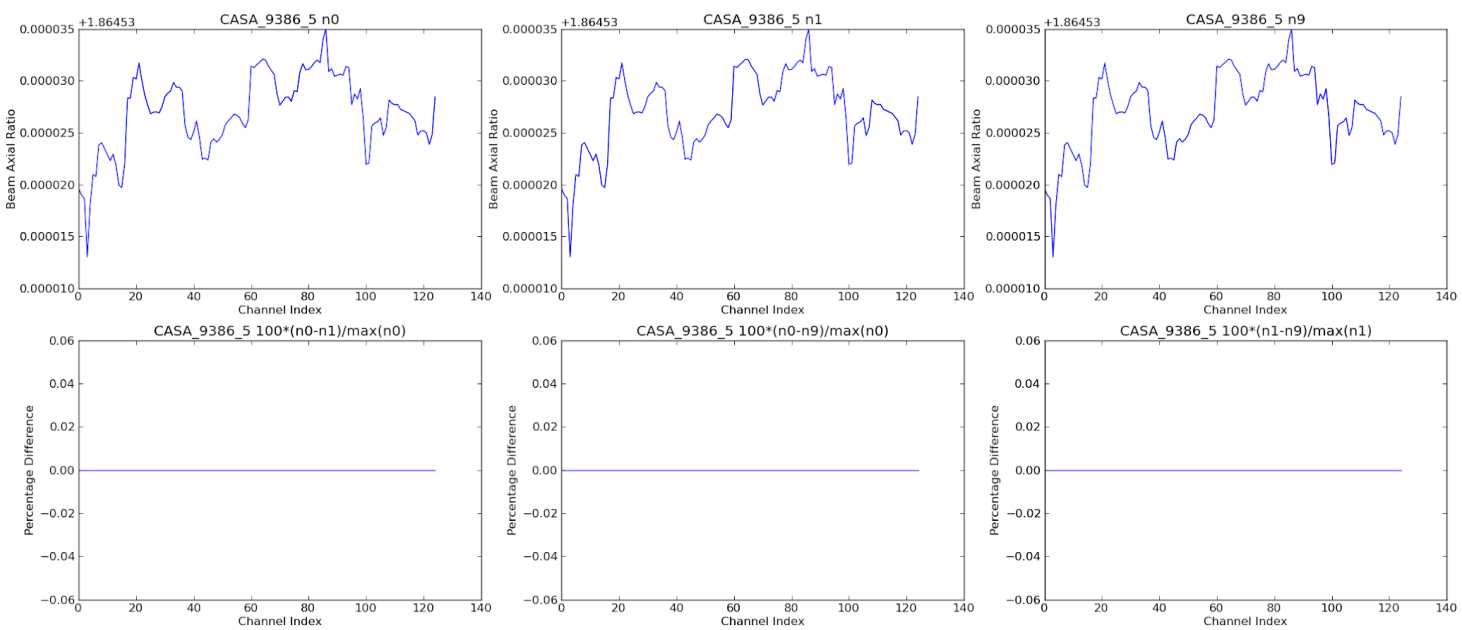
Flux Density plots for CASA_9386_5: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



Noise per channel plots for CASA_9386_5: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



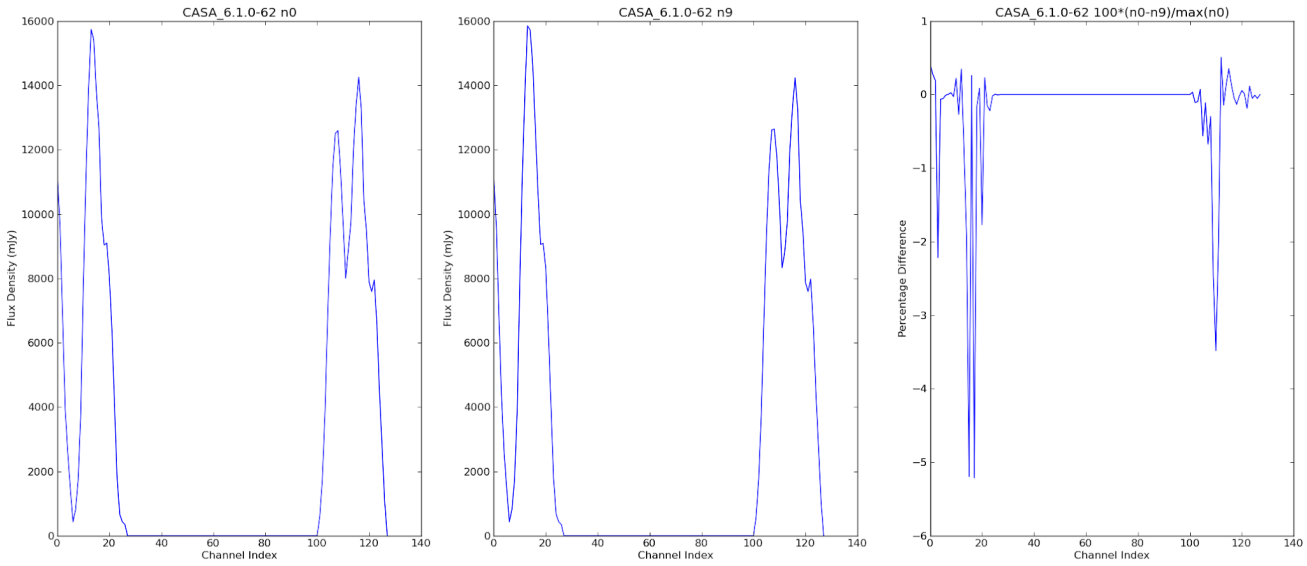
Axial ratio for CASA_9386_5: top row n0, n1, n9; bottom row n0 vs n1, n0 vs n9, n1 vs n9



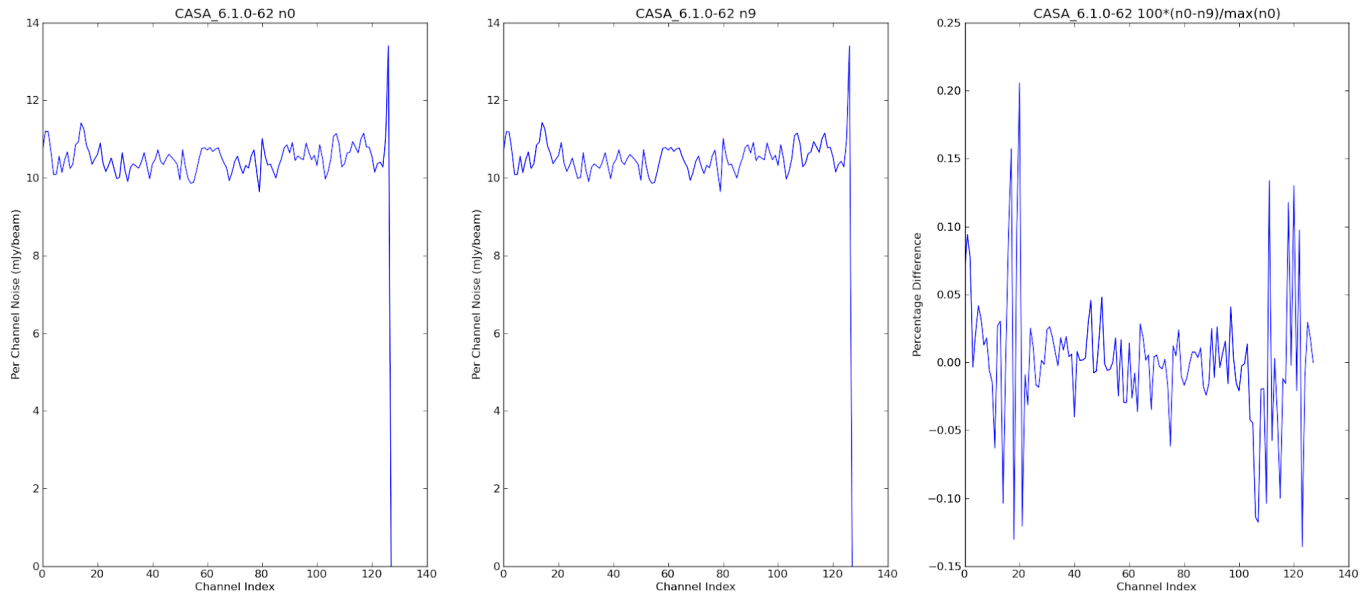
CASA_6.1.0-62 (casa6 master)

Old code : Differences are all in the 1-5% range. There are axial beam ratio differences. This is what ALMA operations has evaluated with the initial CASA6 tests.

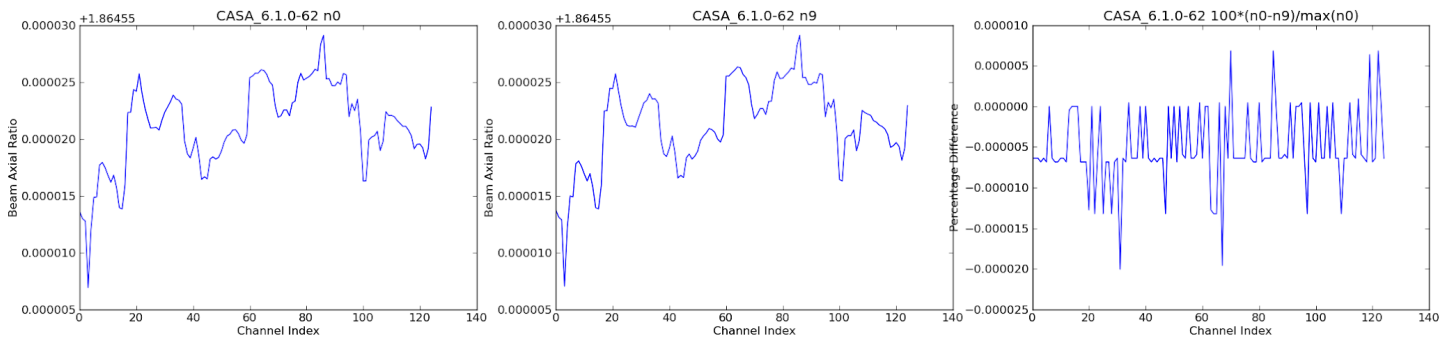
Flux Density plots for CASA_6.1.0-62: n0, n9, n0 vs n9



Noise per channel plots for CASA_6.1.0-62: n0, n9, n0 vs n9



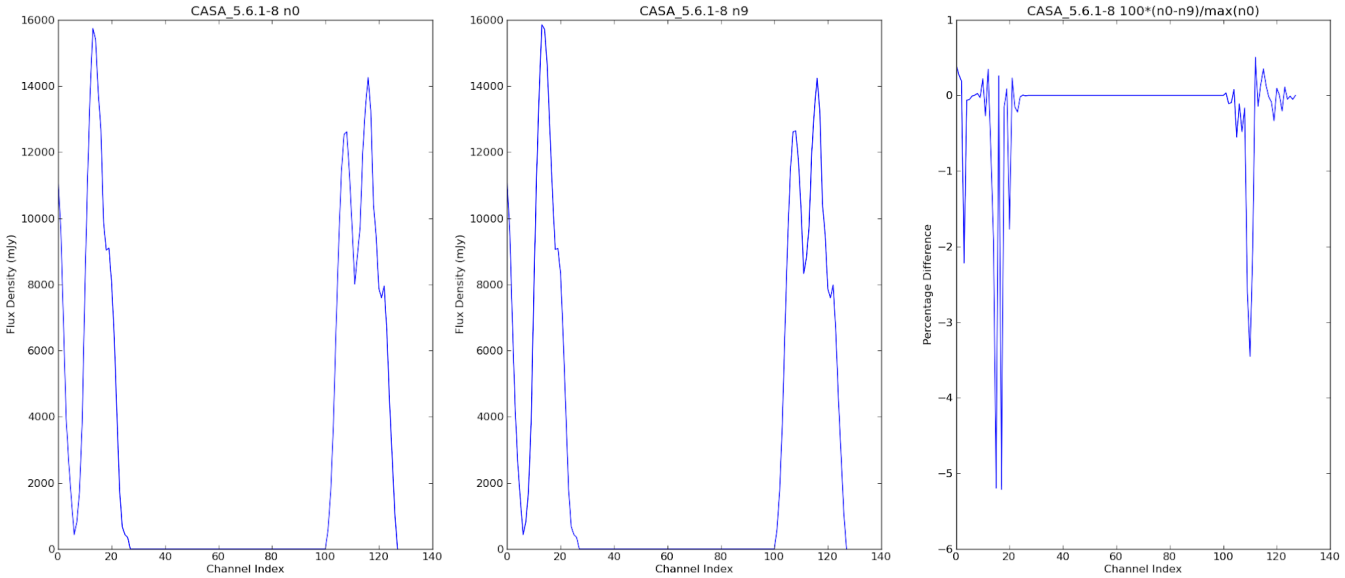
Axial ratio for CASA_6.1.0-62: n0, n9, n0 vs n9

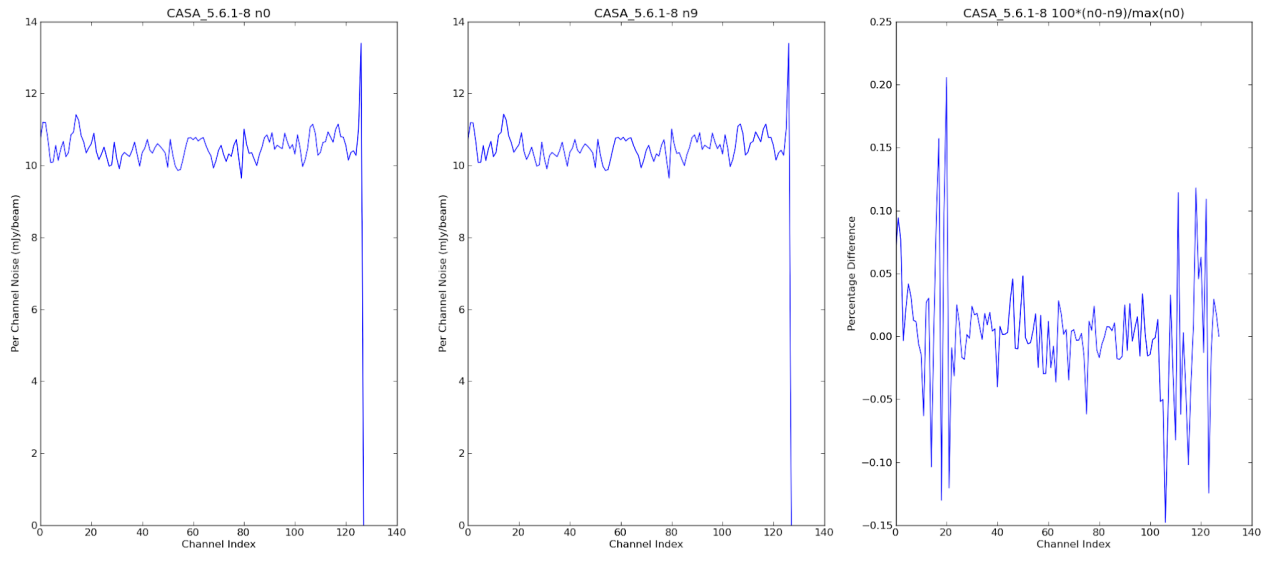


CASA_5.6.1-8 (casa5.6 release)

Old code : Differences are all in the 1-5% range. There are axial beam ratio differences. This is what ALMA operations has been using till date.

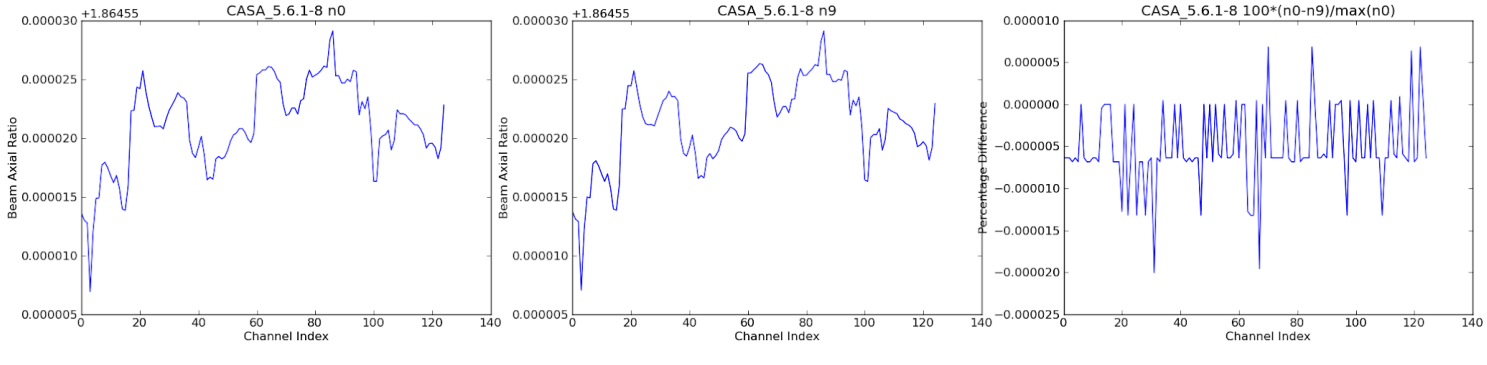
Flux Density plots for CASA_5.6.1-8: n0, n9, n0 vs n9





Noise per channel plots for CASA_5.6.1-8: n0, n9, n0 vs n9

Axial ratio for CASA_5.6.1-8: n0, n9, n0 vs n9



##

A few basic checks with the PLWG dataset (multiple MS, EB)

A few basic checks with this set of datasets. The primary purpose of this test was to ensure operational readiness for the situation of lists of MSs and multiple EBs. The full suite of numerical tests (as in the previous section) were not repeated here.

The comparison between old and new code showed no visible differences, within the spectrum noise level => Scientific Equivalence.

Profile plot at 2 places (on a source and off in a noisy region) with cubes imaged serially with old code 6.1.0-59 and v6 cas9386 build 47. This is to (roughly) assess whether differences reported as percentage differences in the previous section are scientifically relevant or not.