EVLA Memo #132
Report on the findings of the CASA Terabyte Initiative: Single-node tests

S. Bhatnagar
NRAO, Socorro

May 18, 2009

Abstract

This note reports on the findings of the "Terabyte-Initiative" of the CASA Project. The presumed goal was to do "standard" processing of a data set of typical size expected from EVLA and ALMA. The data used for tests was in the Measurement Set (MS) format containing 86 GB worth of simulated visibilities (2 hours synthesis with 1024 channels, 4 polarizations, 2 fields and integration time of 1 second). The expected average data volume from EVLA and ALMA in a 10hr observation in the next 1–2 years is similar (the data volume for ALMA would be slightly larger than EVLA).

The total time taken for "standard" processing on a single CPU for 1K $\times$ 1K $\times$ 1K spectral line imaging (including flagging and calibration) for this data set was $\sim$ 30 times longer than the total observing time. Spectral line imaging dominates the total run time by a large factor. Furthermore, the single-node run-time depends strongly on the size (and possibly the shape) of the image cube (e.g. 1K $\times$ 1K continuum imaging took $\sim$ 10hr while 1K $\times$ 1K $\times$ 1K spectral line imaging took $\sim$ 64hr). Part of this strong scaling however might be related to the access pattern of the image cube in the RAM and the data from the disk. With the existing code, the estimated total run time split between I/O and computing was in the ratio 40% : 60%. However this ratio can vary by a lot, depending on the code optimization and the hardware used.

Tests and work described here is appropriate for pipe-line processing. Manual processing and/or data visualization could easily require more I/O (more reads of the data) and consequently increase the time required for processing by an amount which is not easy to determine (human free-will gets involved!).

1 Data Simulation

A data set was simulated for two fields with dwell times of 2^m and 30^m respectively at 4.8GHz, 4 polarizations EVLA B-array configuration with 1024 frequency channel across

800MHz bandwidth, spread across 32 spectral windows at an integration time of 1 second. In the Measurement Set format, each row holds the data for all polarizations and all frequency channels. The total number of such rows (N_{rows}) and total number of visibility samples N_{samp} is given by

$$\begin{aligned} N_{rows} &= N_{spw} \times N_{fields} \times \frac{N_{ant}(N_{ant}-1)}{2} \times \frac{T}{\Delta T} && (1) \\ N_{samp} &= N_{ch} \times N_{pol} \times N_{rows} && (2) \end{aligned}$$

where N_{pol} is the number of polarization products (4), N_{ch} is the total number of frequency channels (1024), N_{ant} is the number of antennas in the array (27), N_{spw} is the number of spectral windows (32), N_{fields} is the number of fields observed (2), ΔT is the integration time (1^s) and T is the total length of the simulated observations (2^h) A list of the parameters used for simulation was given in Table 1. With these parameters, the total number of rows in the MS were 86261760. The total size of the simulated visibility samples plus

Table 1: Parameters used for data simulation

λ	6cm	the wavelength of observation
N_{ant}	27	Number of antennas
B_{max}	12.5 Km	Max. baseline length
N_{ch}	1024	Number of channels
N_{spw}	32	No. of Spectral Windows
N_{pol}	4	Number of polarizations
T	2hr	Total length of observations
ΔT	1s	Integration time
N_{Fields}	2	No. of fields observed

associated weights was 86 GB. This is roughly the I/O load for computing residuals in any iterative algorithm (solvers for image or the calibration terms in the measurement equation). The total disk space required for processing (due to scratch data) was approximately 266 GB[1]. The channel width in the simulation was set to minimize bandwidth smearing and the integration time was chosen to generate the data volume we think is appropriate from the telescopes in the near future. In reality however, the typical integration time will be longer (e.g. typical integration time with VLA is 10 second) but the number of channels required to minimize bandwidth smearing will be about the number of channels used in this simulation. These parameters however can vary a lot depending on the scientific goals (e.g. the number of channels can be an order of magnitude larger, etc.)

[1]Work in CASA is currently in progress to eliminate the scratch columns where not necessary. However note that some scratch data on the disk is required during processing in all packages, including CASA.

2 Data volume

The bulk of the data volume comes from the storage of complex visibilities, weights and (u, v, w) co-ordinates. This can be estimated by the following equation:

$$\begin{aligned} Data\,Size &= N_{samp}\left[2 \times N_{col} \times SoF + SoWt/N_{pol}\right] + \\ &\quad N_{rows} \times 4 \times SoF \end{aligned} \quad (3)$$

where N_{col} is the number of data columns, SoF is the size of a floating point number in bytes and $SoWt$ is the number of bytes per weight storage. The first term in Eq. 3 represents complex visibility vector plus spectral weights storage. The second term represents storage for time stamp and (u, v, w) co-ordinates. For $N_{col} = 3$, $SoF = 4$ and $SoWt = 4$, the size of the database is $\sim 250\ GB$ for the storage of weights and complex visibilities and $\sim 0.3\ GB$ for the storage of the (u, v, w) and time-stamp The storage for a single data column is $\sim 86\ GB$. Clearly, any optimization of data storage must focus on the first term in Eq. 3.

The data used here was appropriate for EVLA. The expected average and peak data rates for ALMA are quoted to be 6.4 MB/s and 64 MB/s respectively. The average and peak data volume for a typical 12hr observation with ALMA would 256 GB and 2 TB respectively. The minimum number of frequency channels from the ALMA correlator would be 256.

3 Data Processing

In the absence of an agreed upon definition of "standard" data processing, with the goal of identifying the most time consuming steps in "standard" data processing, the simulated data set was passed through what we think are simplest of the "standard" data processing steps required. In reality, the data processing step would be similar, but with parameter settings that would require more computing and disk I/O. Hence, while the steps enumerated below are close to steps one would take for a real data reduction process, the actual run-time will vary with the requirements of the experiments (high dynamic range vs. low dynamic range, complexity of the field, etc.).

Following are the data processing steps that were used:

1. While a data from the real telescopes (EVLA and ALMA) will need to go through the process of translating the archive database to the Measurement Set (MS) format required by CASA, this step was not required here since the simulations were also done using the CASA software which produced the data on the disk in the MS format. However since the existing plan for EVLA processing would require exporting the MS format to the UVFITS format, the data set was exported to the UVFITS format using the `exportuvfits` task in `casapy`.

   In an one-time operation, the MS was distributed across the cluster by splitting it across the frequency axis. While not done for these tests, this operation can be done in the background (i.e. loading the data for the next job, while processing the current job on the cluster).

2. Flagging: Data flagging can vary from simple `quacking` (flagging few initial and final integrations of each scan) to more elaborate RFI flagging routine. While the I/O and computing requirements for `quacking` are small, for RFI flagging both these can be large. For `quacking`, only the `TIME` and `ROW_FLAG` columns need to be read. RFI flagging on the other hand may need to read the entire visibility data set and might need to read it multiple times. Hence the actual run time for flagging can vary significantly.

3. Calibration: For Stokes-I imaging, the two required calibration steps involve solving for the G-Jones and B-Jones terms (the multiplicative antenna based and frequency dependent complex gains). Solver for B-Jones essentially applies the G-Jones solver for each channel (with on-the-fly correction for G-Jones and averaging in time if required). Note that both G- and B-terms are solved for separately for each parallel hand polarization product (RR and LL polarizations). Hence for both these calibration steps, the software *needs* to read the entire data set. Note also that both these steps are *essentially* independent of whether the final requirement is spectral line or continuum imaging[2].

4. Imaging: Imaging can be of two basic types: (1) continuum imaging where the data from all channels are averaged on a single plane, and (2) spectral line imaging where a separate image is made for each frequency channel. Additionally, imaging can also include full polarization imaging requiring all Stokes image planes to be made. The image size therefore can be $N_x \times N_y \times N_{pol} \times N_{ch}$ where N_{ch} is 1 for continuum imaging and N_{pol} can be 1,2, or 4 (for Stokes-I, Stokes-IV and full stokes imaging).

   The image size depends on the maximum baseline length and the field of view of interest. It is expected that for EVLA full-beam sensitivity limited imaging, the field of view for deconvolution needs to include at least the first sidelobe of the antenna power pattern. The number of pixels on a side required for Nyquist sampled imaging with antennas of diameter D and a maximum baseline length of B_{max} is given by

   $$N_x = 3 \times \eta \times N_{lobes} \times \frac{B_{max}}{D} \qquad (4)$$

   where $N_{lobes} = 1$ for the imaging only the main lobe and 2 for imaging up to the first sidelobe of the antenna power pattern. $\eta > 1$, is required to minimize aliasing effects and is often set to a value of 2.

[2]The current style of calibration involves three steps: (1) Solve of G-Jones, (2) apply G-Jones and solve for B-Jones, and (3) use the solved G- and B-Jones to write out corrected visibilities. The corrected visibilities are then used for imaging. While this nicely decouples the various steps enumerated above decouples the problem into separate solvers and calibration, it results into two extra reads of all the visibilities.

4 Wall-clock Run Time

This section reports the measured run-time (wall-clock) for the various steps enumerated above. These tests were done on a machine with 2 Quad-core Intel Xeon E5420 processors running at 2.5 GHz, each with a cache size of 6144 KB (L2 cache of 1.2MB). The code is however not multi-threaded. Consequently only a single core was used in all these tests. The CPU is rated[3] at 40.0 GFLOPS (peak). The total RAM on the machine was 8 GB and available disk space was 934 GB. A summary of the measured wall-clock run

Table 2: Summary of the wall-clock run time for "standard" steps in offline data processing.

Operation	Run time
Flagging:`quack` only	23^m
Flagging: clipping	66^m
G-Jones solver	1^h54^m
B-Jones solver	1^h53^m
Data calibration	2^h23^m
Imaging	64^h
Export UVFITS	1^h30^m
Total	$\sim 74^h$

time for the various steps of processing are listed in Table 2 above. Some details of the the various data processing steps follow.

1. Flagging: The `flagdata` task in `casapy` was used for this step. To make an estimate of realistic run-times for flagging, two kind of flagging was attempt: (1) running the `flagdata` task in the `quack` mode, and (2) flag the data with amplitude outside a user defined range. Note that both these kind of flagging is typically required for most observations (spectral line or continuum). Note also that the I/O requirements for these are very different and nearly span the range of I/O that the *any* flagging algorithm would require.

   The total run-time for `quack` for this data set was 23^m.
   The run-time for flagging data with amplitude outside a range was 66^m
   (the run-time for this operation is independent of the user defined clipping range).

2. Calibration: This step involved solving for first the G- and the B-Jones terms of the measurement equation. G-Jones was solved on time scale of 2^m while the time constant for B-Jones was 20^m. Both solutions were done for the target source which contributes 95% of the data volume. Hence, this calibration run-time corresponds to the "selfcal" run-time in practice. The run-time for primary calibration (i.e., B- and G-Jones solutions for the calibrater field) is typically much smaller (few percent of the total run time).

[3]http://www.intel.com/support/processors/xeon/sb/CS-020863.htm

The wall-clock run time for G-solver was: 1^h54^m.
The wall-clock run time for B-solver was: 1^h53^m.

Calibrating the data for G- and B-terms only (direction independent gains) involves applying the complex transpose of the G- and B-Mueller matrices to the observed data. The computing required for this step is small and the total time for this step is also dominated by the one read and one write of the entire database. The run-time for this step was 2^h23^m.

3. Imaging: For continuum Stokes-I imaging, an image size of $1K \times 1K \times 1K$ was used to include the first side lobe of the antenna primary beam. Natural weighting was used for imaging weights. For imaging, the total run-time is dominated by the major-cycle in the iterative deconvolution iterations. The number of major cycles in-turn is dependent on the kind of algorithm used for deconvolution and can significantly vary from one algorithm to another. The total run time for the imaging is therefore expected to be proportional to the number of major cycles.

   The major-cycle involves a full prediction of the model visibilities followed by the computation of the residual image. In the implementation in CASA, these operations together require a full access of the database. The run time for a major cycle is therefore dominated by the computing load for doing gridding operation plus the time required for full read of the visibility database.

   For a 1000 component deconvolution with a loop gain of 0.1, the imaging required 5 major cycles. The total run-time for the Stokes-I continuum imaging test was 64^h. Note that a large number of temporary files were generated on the disk during imaging indicating that there was significant swapping of the image cube from RAM to the disk. This is partly because of the fact that it is required to hold equivalent of $4-5$ image cubes[4] in the RAM during imaging and the underlying libraries used in CASA were not fully 64-bit compliant and partly due to the access patterns for the image cube in the RAM and of the data from and to the disk. As a test of this hypothesis, a $1K\times\ 1K$ continuum imaging was done, which finished in $\sim$ 10hr.

4. UVFITS Export: The total wall-clock run time for exporting a single data column of the MS using the `exportuvfits` task of CASA was about 1^h30^m.

4.1 Spectral line imaging vs. Continuum imaging

Spectral line imaging requires imaging each frequency channel in the database separately followed by deconvolution of the images corresponding to channels which have significant signal in them. Continuum imaging on the other hand averages the visibility data during gridding and image deconvolution is done on the resulting image. The only significant difference therefore between continuum and spectral line imaging is in the minor cycle (approximate removal of the PSF from the Dirty Image). The major cycle both cases

[4]Some of these are fundamentally required and some of the memory overhead is a trade-off to minimize disk access for reading and writing the data per imaging major cycle.

goes through gridding and prediction of multi-channel database. Note that even for continuum imaging, a multi-channel database is required. Computationally as well as from the data I/O load point of view, both type of imaging experiments are very similar.

Since the dominant run-time load is due to the major cycles, one would expect that the run time for both kind of imaging experiments would be similar. However in practice, the run-times are obviously vastly different. This is due to the vastly different run-time memory requirements. As mentioned before, imaging requires equivalent of 4–5 full sized images in the memory during iterative deconvolution[5]. While 8GB of RAM or more is more than sufficient for typical continuum imaging, spectral line imaging with, say 1024 channels (image cube of size 1K $\times$ 1K $\times$ 1K), these buffers do not fit in the RAM resulting in excessive swapping and consequent increase in effective I/O load (these buffers are accessed in tight loops for gridding/de-gridding with a quasi-random access pattern). However even modest sized clusters with reasonable RAM per node is sufficient to hold these buffers distributed across the RAM in the cluster. The run-time speed up for distributed spectral line imaging is therefore partly due to distributed computing and I/O and partly due to the elimination of the swapping of these buffers from the RAM.

For imaging of large field-of-view (e.g. at L-band with the EVLA or mosaicking with ALMA) the memory requirements are high. It is possible to reduce this requirement by use of smaller facets to cover sources of interest (particularly strong isolated sources outside the main-beam). However it is unclear if this will help in a distributed computing environment. More research in this area is required to estimate if there will be computing benefits, keeping in mind that more advanced algorithms for imaging (and probably for calibration) will be required in practice.

5 Discussion

Compressing the visibility data by a factor α_c would reduce the I/O by a similar factor but will increase the computing by a small factor which will depend on the compression scheme. Assuming the computing required to un-compress the data in the memory to be negligible, time taken for a single access of the entire visibility at the rate of $R_{I/O}$ bytes per second is:

$$T_{I/O} = \frac{Data\ Size}{\alpha_c\, R_{I/O}} \tag{5}$$

Compute time varies for different operations and for different algorithms for imaging itself. Since the most expensive operation is imaging, which in turn is dominated by the computing required for the gridding step, we can estimate the compute-time for a single gridding operation as:

$$T_{comp} = \frac{N_{op}\, S^2}{R_{comp}} \left[\frac{N_{polused}\, N_{samp}\, N_{IP}}{N_{pol}} \right] \tag{6}$$

[5]This requirement is even higher for more advanced algorithms which will be used in practice. The analysis here uses the simplest algorithms for clarity and clear identification of the issues.

where N_{IP} is the number of image polarization planes, S is the convolution function support size and R_{comp} is the effective number of FLOPS that the code can extract from the CPU. $N_{op} = 14$ is the number of floating point operations required for gridding per pixel of the convolution function. $N_{polused}$ is the number of visibility polarization products used to compute N_{IP} image planes. The minimum value of $N_{polused}$ is 2 when $N_{IP} = 1$ *or* 2 (for Stokes-I only and Stokes-IV imaging) and the maximum is 4 when $N_{IP} = 3$ *or* 4 (for Stokes-IQU or full polarization imaging). All else remaining same, since the compute-only time depends on the number of visibility points and I/O time depends on the data size, data compression by a factor of 2 will reduce the run time significantly only if the total run time is I/O dominated.

The total time taken for the "standard" processing of the data volume for the equivalent of a *typical* 10^h observation is $\sim 74^h$. This is ~ 7 times slower than quasi real-time-processing (defined as the ratio of processing time to observation time be ~ 1.0). The average data volume in 12hrs from ALMA is expected to be 256 GB. The data processing steps for reducing ALMA data would be similar to those used in this test. The total run time of average ALMA data could therefore be ~ 20 times longer. Recall that the data volume for realistic observations is expected to be larger for EVLA and the peak data volume in 12hrs for ALMA would be about 2 TB.

The term in square brackets in Eq. 6 is the total number of visibility samples used in a single gridding operation. Using the spread-sheet associated with ALMA DRSP[6], the average number of visibilities for ALMA is in the range $10 - 100$ Giga samples. Estimated computing time required for single gridding operation is in the range $1.3 - 13$ hrs. Taking the total length of observation to be 69 hrs, and assuming 10 full access of the data required for full scientific data reduction, post processing time for ALMA using a single CPU could also be $\sim 10\times$ the total observing time. Note that this figure could easily vary by a factor of $2 - 5$ due to larger number of full data accesses required in real life as well due to larger than average number of data samples.

It should however be noted that the projected run-time for EVLA and ALMA discussed above assumes that the current single-node run-time is the best we can achieve. At this time it is not clear if the code is fully optimized (e.g. for optimal use of available RAM, sizes of the various buffers optimized for the available RAM, tiling pattern used for storing the Measurement Set on the disk, optimal use of multiple cores per CPU, etc.). Since a significant improvement in the single-node run-time will impact the design parameters for post-processing hardware, optimization of the existing software for improving single-node performance is worth investigating. However, given the data volumes, I/O and computing requirements of the various algorithms and the fact that in practice only a fraction of the CPU peak FLOP rating can be harvested, it is clear that some parallel processing will also be required. Improvements in single-node run-time can reduce the scale but not eliminate the need of parallel processing. Efforts to improve the single-node run-time should therefore be done keeping in mind that processing should scale well in a parallel processing environment.

[6]ALMA Memo #501; http://almasw.hq.eso.org/almasw/pub/SSR/DataRateCalculationResults/AllDRSP.xls

6 Conclusion

The simulated data set was for single pointing EVLA observation. The parameters used for the simulation were such that the data size is comparable to the typical data size expect in the next few years and is also large enough to expose parts of data processing steps that consume the largest amount of time but not so big that it becomes difficult to handle with the available disk space and RAM.

The data was processed using the CASA software running on a machine with four 64-bit CPUs running at 2.5GHz with 8 GB of total available RAM and 934GB of disk space. A single CPU is rated at 40.0 GFLOPS. Our code harvested 20% of the rated GFLOPS. Simple experiments with OpenMP using 2 threads harvested $\sim$ 40% of the single CPU power - the amount computing power harvested per CPU was still $\sim$ 20%. However multi-threaded code also imposes significantly increase in the run-time memory requirements (requires a complex image buffer per thread).

The computer operating system schedules independent processes on separated CPUs/cores on a multi-CPU multi-core machines. Since analysis of the most time consuming operations show that an independent in-memory buffer for accumulation of intermediate results might be required per thread, using OpenMP for lower level threading of the inner loops might be no better than running independent OS level processes and combining the results from these processes at a higher level. The latter is easy to test (at the scripting layer) without significant changes in the lower level code.

The data processing steps were chosen to be the typical steps one would take for the *simplest* data reduction. It must be emphasized that such a test is appropriate for pipeline processing - i.e. no manual flagging or visualization of the visibility data set was attempted. In that sense, the CASA software for visibility data visualization has not been exercised here for handling large data volumes. The data size used for this test was also on the lower side of the expected data rates from EVLA or ALMA. Therefore, the run times given here for various steps should be treated as the lower limits.

The computing needs can vary significantly depending on the imaging dynamic range requirements and the complexity of field emission. In the tests done here, corrections for none of the effects that will be required for noise limited full-beam imaging with the EVLA were used. The I/O requirement to solve for *direction independent* gains is less than that for image deconvolution involving several major-cycles. The I/O requirement for some of the calibration algorithms that involve solving for *direction-dependent* gains is similar to image deconvolution. The computing requirements for the algorithms that can correct for these effects is higher than for the “standard” processing reported here. The FLOPS per I/O ratio for such algorithms is therefore potentially higher. With easy availability of multi-core CPUs, this is a favorable direction of evolution of post-processing algorithms. Efficient implementation utilizing multiple cores per CPU has the potential of mitigating otherwise significant increase in total run-time.

Finally, while the tests were done for a single pointing observation, some useful conclusions can be drawn for mosaic imaging as well. For direction independent calibration, the compute and I/O loads for calibrating mosaic data set are similar to that for single pointing observations (the entire data has be read and written to the disk at least once). The dominant time consumer for mosaic imaging is also the major-cycle. Since the major cycle is really independent of the imaging mode (spectral line or continuum imaging of single pointing or mosaic observations), the run time for imaging a mosaic data set will also be limited by I/O load in the same manner as for the single pointing case.

6.1 High Performance Computing

For High Performance Computing (HPC) needs, assuming "normal" local disks at each node, it appears that the largest reduction in the total run-time will happen by parallelizing the I/O. Assuming that the hardware we will use for HPC is a cluster, the implication is that we need a cluster with distributed disk storage across the compute nodes. Each compute node will be used for calibrating and imaging a subset of the database which will be distributed across the cluster. This will parallelize the I/O. Since in this model, the I/O load is reduced by doing essentially distributed I/O, it also means that we do need to parallelize the computing as well. This is a reasonable way forward since, as mentioned above, more realistic data processing will have higher computing needs. Also once the I/O is parallelized, beyond a certain number of nodes in the cluster, the time taken by each node to do the work can be limited by computing. Once that limit is hit, use of multiple CPUs/cores by either running multiple processes or multi-threading the inner loops might be required (the former is more recommended as a the first step).

6.2 Recommendations for the cluster parameters

With the initial ideas about parallelization for the embarrassingly parallel problems at hand, we think a cluster with local disks of capacity $\sim$ 300GB, at least 2 CPUs per node (to allow experimentation with technologies like OpenMP) with 4K or more of L2 cache per node will be appropriate. At the time or writing this memo, our own analysis as well as discussions with other groups suggests that use of multiple CPUs/cores does in fact requires a significant increase in the RAM requirements. We therefore recommend at least 16 GB of RAM per node or more. Since there are significant run-time advantages in keeping the data and the buffers required during processing in the RAM, in case the amount of RAM per node is limited by cost considerations, we further recommend that, number of CPUs per node and/or cores per CPU be traded for the size of the RAM per node. The kind of interconnect we might want depends strongly on the algorithms we will use (strong dependence on the amount of cross talk between the nodes). Our tests with a 16-nodes cluster so far suggests that a 10 Gbit/s interconnect with NFS to access the local disks of the nodes might be sufficient.

The bandwidth between the CPU/L2 cache and RAM will however dominate the compute-only time (or equivalently, the effective number of FLOPS we can harvest). This, I believe, will also become an issue even if we were to use OpenMP locally at the nodes. A technology survey in this area will be most useful - e.g. nodes with CPU and RAM on a fast bus (Hyper Transport) will probably be very useful.