Parallel Facet Imaging in Obit

W. D. Cotton, February 4, 2009

***Abstract*—This note explores the concept of parallel imaging in the formation of multiple facets needed for Fly's Eye imaging. This is a "read once, grid many" technique as opposed to the serial "read many, grid once" approach. In a small test case presented in which the data fit entirely in disk cache, the parallel imaging ran up to 50% faster than a serial version; an enhancement significant enough to also make noticeable improvement (30%) in a more realistic test. This difference is believed due to the improved efficiency of threading. For a large test case which did not fit in the disk cache, the serial method using threads running in 8 cores was about a factor of two from being CPU bound. The parallel method is solidly CPU limited and the multiple imaging steps ran in half the time of the serial method.**

***Index Terms*—imaging, interferometry, parallel processing**

I. Introduction

ONE of the common solutions to the "co-planarity" problem in interferometry - that images are flat but the sky is not - is the "Fly's Eye" technique [1] of using multiple facets, each tangent to the celestial sphere, to tile the region of the sky to be imaged. One drawback to this technique is that simple implementations require reading the entire dataset for each facet image and each associated facet dirty beam. In practical cases, a few to several thousand facets may be required.

Current instruments generally produce continuum data sets that are sufficiently small that they easily fit in memory of modern computers. For example, large VLA continuum data sets are seldom larger than a few hundred MByte while the NRAO Obit development machine in Charlottesville, mortibus, contains 8 GByte of memory. The disk caching technique of the Unix operating system will thus keep the entire dataset in memory even if the software is formally using a strip-mining technique (as is the case in Obit and AIPS) and the actual transfer rate from the disk is not an issue.

This will not be the case for instruments currently under development (e.g. EVLA and ALMA) which can produce datasets several orders of magnitude larger. In this regime, the hundreds to thousands of "reads" of the data are unacceptable.

This note explores an idea discussed by Eric Greisen at the Oxford Algorithms meeting in December 2008 which he has implemented in AIPS. This concept is simple. Unless direction dependent calibration is being applied in the imaging, each facet image and dirty beam uses exactly the same data; they are merely rotated to a different tangent point on the sky, gridded and FFTed. Multiple image/beam grids can be accumulated from a single read of the dataset.

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

Manuscript received ; revised

With the multi-threading implemented in Obit ([2], http://www.cv.nrao.edu/∼bcotton/Obit.html), the work of griding can also be spread over multiple threads; parallel, threaded gridding can both dramatically reduce the total I/O and result in increased efficiency of execution.

II. Parallel Imaging

Parallel facet imaging was implemented in Obit in the ObitImageUtil utility package as routine ObitImageUtilMakeImagePar. This routine is passed a UV dataset and an array of images (and possibly dirty beams) to be formed from this data. If multiple images and/or beams are to be formed, this routine creates an ObitUV data object for each image/beam; these all point to the same dataset. In addition, an array of ObitUVGrid objects are created, one for each image or beam. These, among other things, contain the UV grid onto which the data is to be accumulated.

The arrays of ObitUVs and ObitUVGrids are passed to the ObitUVGrid class routine ObitUVGridReadUVPar which implements a parallel read/griding operation to fill the accumulation grids on each of the ObitUVGrid objects

A. Parallel "I/O"

A single read of the data is used to fill multiple buffers. This uses ObitUV class functions ObitUVReadMulti and ObitUVReadMultiSelect for the actual read and to fill the first set of thread buffers. These two routines differ in that the latter allows calibration, editing, and selection.

Subsequent sets of thread buffers are filled using ObitUVReReadMulti and ObitUVReReadMultiSelect which uses data saved from the initial read and possibly applies a facet dependent calibration. Actual parallel I/O is implemented in the ObitIO virtual class as ObitIOReadMulti, ObitIOReadMultiSelect, ObitIOReReadMulti and ObitIOReReadMultiSelect which then call the ObitIO derived class for the specific data type (currently, FITS or AIPS).

B. Parallel Griding

Parallel gridding is done in ObitUVGridReadUVPar which determines how many threads are to be used (a user specifiable parameter) and creates an array of accumulation uv grids, one for each image/beam and thread. Two buffers are used for the parallel I/O. The gridding process first reads a buffer of data and then loops over the images/beams, dividing the work amoung the threads for each as is described in Obit Development Memo no. 1. Griding consists of multiplying the randomly sampled UV visibilities by a continuous anti–aliasing ("griding convolution") function which is then resampled onto the UV accumulation grid. Multiple copies of the

accumulation grid are needed to avoid the dependency between threads when they are gridding points in the same region of the grid. When all of the data have been read and gridded, the individual thread accumulation uv grids are combined into a single grid.

This implementation of griding modifies the contents of the UV data buffer as needed for each facet. This includes rotation of the data to the facet tangent point and replacing the data with (1,0) to form a dirty beam. This technique requires a separate copy for each image/beam. The efficiency of the threading is a function of the amount of data in each buffer; the buffer size read is temporarily modified (in ObitImageUtilMakeImagePar) to ensure at least a minimum for efficient operation. Note: the computational expense of the rotation of the position to the facet center is comparable to the gridding; the computation time for making a beam is roughly half of that for the corresponding image of the same size.

C. FFT, Griding Correction

Once the entire data set is gridded onto all the facet UV grids, each grid is FFTed to the image domain; this uses ObitUVGrid function ObitUVGridFFT2ImPar. In principle, these FFTs could be done in parallel threads. However, the FFTW implementation also allows multiple threading and there appears to be a conflict between the Obit and FFTW threading. Therefore, the FFTs are done serially but using the FFTW threading.

After the Fourier transform, the images need to be corrected for the effects of the griding convolution function. This correction is performed in parallel threads by ObitUVGridFFT2ImPar.

The final step is the normalization of the images and writing to disk. The results of the previous steps have an arbitrary multiplicative factor common to each image/dirty beam pair. This results from the data weighting as well as the details of the griding and FFT. Since the dirty beam is defined to be 1 at the center, the normalization factor is the value of the derived dirty beam central pixel. Both the beam and image are normalized by this factor and the image is written to disk. These final steps are done serially in ObitImageUtilMakeImagePar.

III. Example 1: Small data set

The implementation described above was tested on mortibus, the Obit development computer at NRAO/Charlottesville. This machine has dual, quad core processors so testing could use up to 8 threads. All available VLA data sets easily fit its 8 GByte memory. A data set was selected from the VLA archive of an NRAO VLA Spitzer First Look Survey field. Combined data from two days of observing on a single pointing containing 200 MByte of data was used. The observations were in the VLA "B" configuration at 20 cm wavelength. This data has two "IFs" of 7 channels each and both RR and LL correlations. Faceted images covering a radius of 0.75° were imaged plus strong outliers selected from the NVSS catalog; 116 facets were used in all. Final images were 3600×3600 pixels. Imaging used Obit task Imager.

TABLE I
Example 1 Timing tests

Test	threads	Real sec	CPU sec	Ratio
Serial short	1	907.96	907.76	1.00
Serial short	2	541.15	904.27	1.67
Serial short	4	419.16	927.08	2.22
Serial short	8	283.39	946.89	3.34
Parallel short	1	911.34	911.14	1.00
Parallel short	2	574.93	911.16	1.58
Parallel short	4	315.04	925.41	2.94
Parallel short	8	183.19	909.72	4.97
Serial Full	8	3618.0	19253.8	5.30
Parallel Full	8	2843.4	16023.4	5.64

The first test was to make a set of facets and do a light clean, 100 components, then flatten the restored facets. This operation is dominated by the initial formation of the images and beams and the final set of residual images. Identical tests were performed using the parallel griding and the serial griding, each using 1, 2, 4 and 8 threads. The timing results are given in Table I as tests "Serial short" and "Parallel short". For each test, this table contains the wall clock ("Real") time, the CPU time and the ratio of CPU to wall clock. This last value indicates the efficiency of the threading.

Since the making of large numbers of simultaneous facets is not the dominant part of a practical processing run, a second test involved using the same data sets but instead of a token clean, doing a more substantial processing including, CLEANing to the noise, doing a phase self calibration followed by another CLEAN to the noise. The results of these, more realistic, timing trials are given in Table I as tests "Serial Full" and "Parallel Full". Wall clock results are shown in Figure 1.

The parallel imaging uses multiple simultaneous UV grids, one set for each image and beam. This additional memory usage is insignificant, the task never using more than 1.7% of the available memory.

IV. Example 2: Larger data set

Since the previous example data set fit comfortably in memory, actual I/O wasn't an issue. A second test case is a larger data set, 2.5 Mvis of 1024 channel continuum data generated to test the casa development cluster at NRAO/Socorro. This cluster is composed of 16 dual, quad core nodes. This dataset was ∼40 GByte in "compressed" format and ∼120 GByte with all values as floats. The single polarization (Stokes I) used for imaging comprised ∼30 GByte. This data set is substantially larger that the 8 GByte memory in the individual nodes of the cluster so test the I/O of the system. Several test runs were made with various amounts of data; 1) small enough that all data imaged fit in memory, 2) half the total data and 3) the full data set. Each test run consisted of imaging a 9 arcmin radius field which required 7 facets, CLEANing 1 component, forming the residual images, and finally flattening them to a single 4000×4000 image. All tests used all 8 processors on a

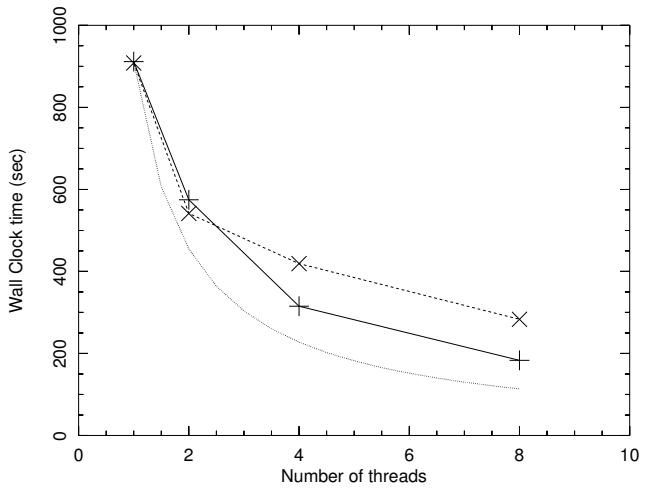

Fig. 1. Wall clock time for test case processing using different numbers of threads. Pluses and the solid line indicate parallel imaging, "X"s and the dashed line indicate serial imaging and the dotted line, the optimum for full utillization of all threads.

TABLE II
Example 2 Timing tests

Test	Total min	Initial Image min	Size	Scaled min
Serial short	24.5	3.4	1.0	3.4
Parallel short	13.7	3.3	1.0	3.3
Serial half	145.6	51.6	7.1	24.1
Parallel half	118.2	25.9	7.1	23.6
Serial full	306.6	108.8	15.0	51.0
Parallel full	249.2	54.7	15.0	49.1

single cluster node of the casa development cluster and were repeated for serial and parallel imaging. The results are given in Table II.

The entries in Table II were for three sizes, 1) "short" = 168 kVis, 2) "half" = 1.2 Mvis, and 3) "full" = 2.5 Mvis. Both the total run wall clock time ("Total") and the wall clock time to grid and FFT the initial set of images (7) and beams (7). The "Size" column gives the ratio of the data volumes to the "short" version. The final column, "Scaled" is the value of the "Initial Image" entry of the short test scaled by the "Size".

V. Discussion

The Example 1 timing test presented above compared two executions dominated by the making of large numbers of facets. The serial and parallel versions have comparable run times for small numbers of threads, but with four or more threads the parallel imaging version was significantly faster; using all 8 available cores, the parallel test was over 50% faster. Much of the speed enhancement from larger numbers of threads came from the more efficient use of threading by the parallel version Even in the second, more realistic test, the parallel facet version was nearly 30% faster when using 8 threads.

For all of the tests in Example 1, the full problem easily fit in the Unix disk cache, meaning I/O speed was not an issue. However, for large data sets, the performance enhancement is expected to be more dramatic. The second example is of such a problem, but the results in Table II show only a factor of two difference in imaging run times for the two techniques for a range of problem sizes. A close examination of the tabulated values shows that the problem is nearly CPU limited even in the serial test, especially in the stage of forming the initial set of images and beams. The run times for the "Initial Image" stage for the parallel imaging show a roughly linear scaling with data volume; that is, there is no significant additional time used over what is needed to do the computation. The amount of I/O for the parallel test is 1/14 of the I/O for the serial test so expected to be CPU limited. The run time for the serial imaging for the two larger test sizes is roughly double the scaled CPU times of the small data test, indicating that the larger tests are moderately I/O bound.

With the hardware and gridding algorithms used, the parallel image formation is likely to remain CPU bound and the serial method only moderately I/O bound. Both the I/O and computation time scale linearly with data volume. Without heavy use of multithreading, both methods are strongly CPU limited.

Acknowledgment

The author would like to thank Eric Greisen for the basic idea explored here and Sanjay Bhatnagar for the large test data set.

References

[1] R. A. Perley, "Imaging with Non-Coplanar Arrays," in *Synthesis Imaging in Radio Astronomy II*, ser. Astronomical Society of the Pacific Conference Series, G. B. Taylor, C. L. Carilli, and R. A. Perley, Eds., vol. 180, 1999, pp. 383–+.

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