CASA Imager Parallelization: Measurement and Analysis of Runtime Performance for Continuum Imaging

S. Bhatnagar
and
The CASA HPC Team

March 13, 2015

Abstract

This memo describes the measurement and analysis of the runtime performance of parallelized Imager for the two continuum imaging uses cases:

1. Multi-term, Multi-Scale continuum imaging using Wide-band (WB) A-Projection
2. Multi-term, Multi-Scale continuum imaging using W-Projection

The A-Projection test was done using CASA's MPI Framework (Gonzalez, 2014) to run multiple instances of imager. The W-Projection gridder is multi-threaded using OpenMP. Tests with W-Projection involved using all available CPU cores by launching multiple processes using CASA's MPI Framework (same as for A-Projection tests) as well via multiple OpenMP threads from a single process per node.

The data used for both these tests was a wide-band data from the EVLA from a 106-pointing mosaic observation. This data was stored on the Luster file system as a partitioned Measurement Set (MS) – a.k.a. MMS – with 106 sub-MSes. The target deconvolver for these imaging tests was the multi-term deconvolver with 2 Taylor terms.

1 Introduction

At a block-level, in an end-to-end processing of the data from modern radio interferometric telescopes like the EVLA and ALMA, the imaging step, and in some cases the deconvolution step dominates the runtime[1]. The runtime cost of the imaging step comes from two dominant sources: (a) cost of data i/o, and (b) cost of re-sampling the data onto and from a regular grid (the "gridding" and "de-gridding" steps).

For the purpose of mitigating these computational and data i/o bottlenecks, continuum imaging can be compactly described as accumulation of all the data along the time, frequency and polarization axies. Ignoring various normalizations, imaging can be described by the following simple linear equation:

$$I_{Cont} = \mathbf{F} \sum_{\nu,t,Pol}^{N_{vis}} \sum_{i,j=0}^{N_{sup}} \mathbf{G}(i,j;\nu,t,pol)\,\mathbf{V}(\nu,t,pol) \tag{1}$$

where $\mathbf{V}$ represents the calibrated data[2] with N_{vis} samples, $\mathbf{F}$ is the Fourier transform operator, $\mathbf{G}$ is the gridding/de-gridding operator and N_{sup} is the support size of $\mathbf{G}$. In general, $\mathbf{G}$ is complex valued and varies with time, frequency, baseline length and polarization and absorbs additional terms (like the Mosaic term). For the purpose of understanding the runtime costs, two conclusions are obvious from Eq. 1:

1. Data i/o costs are determined by requiring to read the entire data (plus $\sim$ 10% of auxiliary data, leading to total of about 30 bytes per data sample)

[1]Ignoring inefficient implementations of other processing steps in the software or unreasonable use of software tools (or, in some cases, both!).

[2]Stored in the `CORRECTED_DATA` column of the MS or MMS in CASA.

2. Computational costs are determined by N_{vis} (due to $\sum_{v,t,pol}$) and the support size of $\mathbf{G}$ (due to $\sum_{i,j}$).

That the imaging equation is also linear suggests that is an Embarrassingly Parallel problem. In theory, the entire runtime cost (computational plus the data i/o cost) can be reduced linearly by partitioning the dataset, making partial images using smaller pieces of the data in parallel and making the final image by summing these partial images (Bhatnagar, 2009; Bhatnagar et al., 2009). Mathematically, Eq. 1 can therefore be written in its "parallel" form as:

$$\begin{aligned}
I_{cont}^{Node_0} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \\
I_{cont}^{Node_1} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \\
&\vdots \\
&\vdots \\
I_{cont}^{N_{nodes}} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \qquad (2)
\end{aligned}$$

Once the partial images (I_{cont}^{i}) are made, the final image is made as:

$$I_{cont} = \sum_{i} I_{cont}^{i} \qquad (3)$$

The total runtime cost (cost of data i/o *plus* computations per node) reduces by N_{nodes}. As long as the computation-to-i/o ratio (computational density) remains "high", this approach should lead to runtime scaling as $1/N_{nodes}$.

The computing cost is more dominated by the second summation ($\sum_{ij}$) in Eq. 1. In the software, this summation is a tight loop that uses in-memory buffers. Parallelization of this may also be possible, independent of the coarser grain Embarrassingly Parallel problem. Assuming that the second summation can also be parallelized using N_{th} simultaneous computations, Eq. 2 can be written as:

$$\begin{aligned}
I_{cont}^{Node_0} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup/N_{th}}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \\
I_{cont}^{Node_1} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup/N_{Th}}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \\
&\vdots \\
&\vdots \\
I_{cont}^{N_{nodes}} &= \mathbf{F} \sum_{v,t,Pol}^{N_{vis}/N_{nodes}} \sum_{i,j=0}^{N_{sup/N_{Th}}} \mathbf{G}(i,j;v,t,pol)\,\mathbf{V}(v,t,pol) \qquad (4)
\end{aligned}$$

The N_{nodes} and N_{th} then become the two crucial tunable parameters that determine the efficiency of simultaneous parallelizing data i/o and computing bottlenecks. In principle then, it is possible to find a value for these parameters that deliver an optimal solution with good parallelization efficiency and good utilization of all available resources on the computer (parallel i/o, CPUs, cores and memory). In Sections 2 and 3, we discuss the implementation of this two-level parallelism in the code. Section 4 has the results from current tests.

In practice this two-level parallelism poses two interesting[3] problems:

1. How to solve the thread-clashing in multi-threaded implementation of $\sum_{ij}$: This is discussed in Section 3 below.
2. How to determine the balance between N_{nodes} and N_{Th}: Their values will depend on the computing resources and details of the problem being solved. This is not discussed here and will be subject of a future memo.

[3]Interesting for me anyway.

Note that this two-level parallelism may be necessary for some class of problems and will be an enabler technology for those problems, even if at low computing efficiency.

2 Parallelization: Using MPI

The parallelization of the first summation in Eq. 1 in CASA is done using the MPI technology. Block diagram of the basic computing unit (the gridder) is shown in Fig. 1. Parallel imager launches multiple copies of the CASA imager using the MPI4CASA framework (Gonzalez, 2014) as shown in Fig. 2. Each process, running the CASA imager, is supplied a fraction of the entire database, via physically partitioning the data in smaller parts and/or via the CASA data selection mechanism. The available computing power is utilized by running independent imagers, one per CPU core per node of the compute cluster.

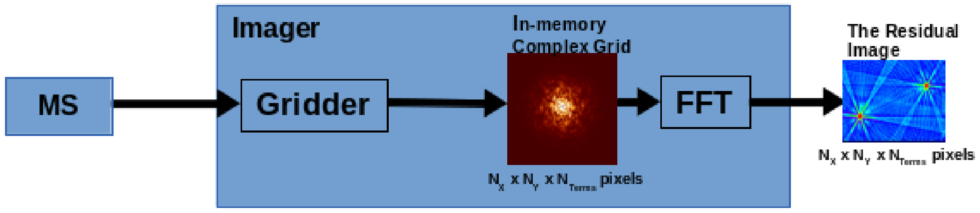
MS
Imager
Gridder
In-memory Complex Grid
N_x x N_y x N_{Terms} pixels
FFT
The Residual Image
N_x x N_y x N_{Terms} pixels

Figure 1: Block level diagram for the Gridder/De-gridder.

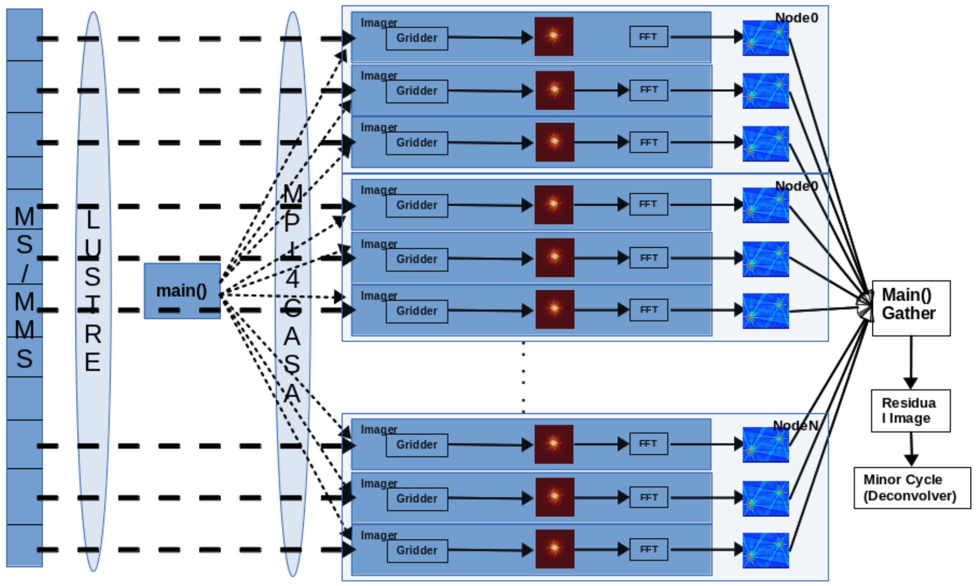

Figure 2: Block level diagram for the parallelized Imager architecture. The fine-dashed lines represent MPI-messages to setup the various instances of imagers. The long-dashed lines represent direct access to the database (MS/MMS on Luster).

As can be seen from Fig. 2, each MPI process holds full-sized complex grids, which are used to accumulate the visibilities (or read gridded visibilities from for de-gridding) by each process in parallel. Each process computes a partial grid using a small fraction of the entire data. Each partial grid is Fourier transformed to compute the partial image. These partial images from each process are then added together in a gather operation to compute the final image corresponding to the full dataset. Since each process is using only a fraction (1/No. of processes) of the full dataset, speedup is expected, both due to parallel i/o as well as lesser computations per node. Since the computing at each process is completely independent, for compute-limited imaging, near linear speed-up is expected.

2.1 Overheads

Following are the overheads with this method of parallelization:

1. **Imager setup:** Imager requires a number of parameters to be set before it can begin to process the input data. These parameters are sent as strings from the single main-process via MPI framework. When the number of processes is large, the overhead of setting up the imagers can become comparable to the runtime of each process.

2. **Memory footprint:** Each process holds the full-size complex grid(s). The memory footprint at each node is therefore a multiple of the number of CASA imager processes running on the node.

3. **Gather operation:** The gather operation requires accumulating the partial images from each imager process to compute the final image. This is currently a serial operation and for large number of processes may become a significant overhead.

3 Parallelization: Using OpenMP

Purely MPI based parallelization, as shown in Fig. 2, which has the advantage of running independent processes, leads to overheads in terms of software complexity, number of MPI processes required, number of sub-images made which directly impacts the gather operation, but most importantly, the memory footprint. For reasons beyond the scope of this document, the gridder holds equivalent of $4 \times N_{terms}$ floating-point grids of the size of the final image in memory during gridding. Using MPI to launch an imager process per CPU core, the total memory footprint particularly for large images becomes $4 \times N_{terms} \times N_{cores}$, which is large and even prohibitive. The

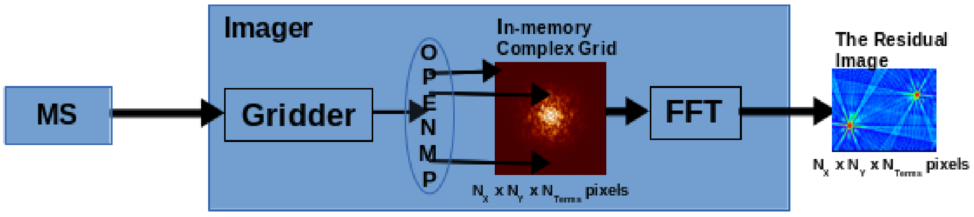
Imager
MS
Gridder
OPENMP
In-memory Complex Grid
$N_x \times N_y \times N_{Terms}$ pixels
FFT
The Residual Image
$N_x \times N_y \times N_{Terms}$ pixels

Figure 3: Block level diagram for the mult-threaded Gridder.

mitigation of these overheads is to develop a multi-threaded gridder as shown in Fig. 3. This will essentially parallelize the second summation in Eq. 1 ($\sum_{i,j}^{N_{sup}}$). A single copy of the gridder is required per node, and the gridder launches as many threads as there are CPU cores. This can fully utilize the available computing power with a memory footprint of a single gridder and does not scale with the number of threads. A single copy of the grids is held in the memory and shared by all threads.

Naive implementation however runs into the problem of thread contention since only one thread can write to the single shared grid. An OpenMP based gridding algorithm however exists (Golap, 2010) and implemented in CASA for WProjection. With this gridder, the parallelized imager (shown in Fig. 4) becomes simpler and needs one copy of the grids in the memory and produces one sub-image per node reducing the number of images to be gathered in the gather-step.

3.1 Overheads

Following are the possible overheads with this method of parallelization:

1. **I/O efficiency:** Since a single processes reads a larger fraction of the data to be parallely processed with multiple threads, there may be a i/o performance penalty. However it may be possible to mitigate this (if present) by overlapping i/o and compute.

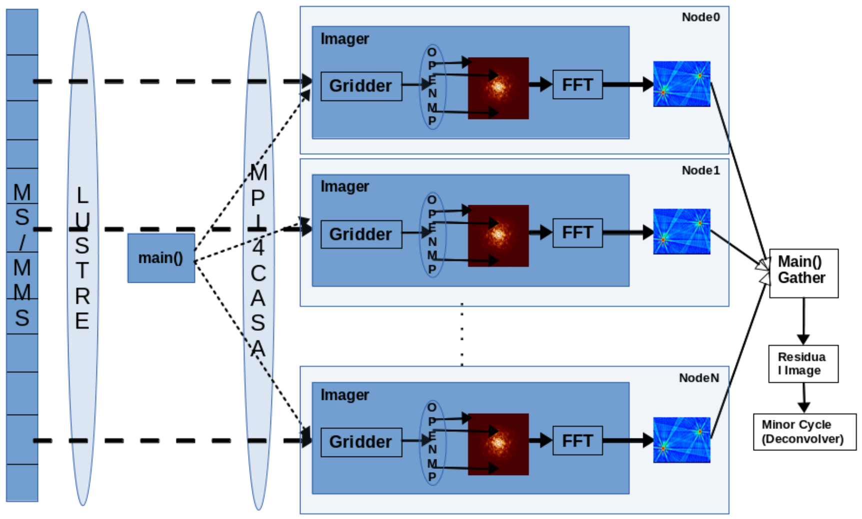

Figure 4: Block level diagram of parallelized Imager architecture with mult-threaded Gridder. The fine-dashed lines represent MPI-messages to setup the various instances of imagers. The long-dashed lines represent direct access to the database (MS/MMS on Luster).

2. **Computing overheads:** The multi-threaded algorithm needs to do more bookkeeping and moderately higher computing than standard gridder.

3. **Cache hits/misses?** It is not clear if by running multiple threads the computing performance of each thread will be affected due to pressure on L1/L2/Lx-caches in the CPU hardware. Someone more familiar with the CPU architecture and performance analysis tools should look at this more carefully.

4 Results

4.1 The Setup

The general computing hardware used for the tests and discussion below is a cluster of nodes, each node with multiple multi-core CPUs. The specific instance of this general hardware we used had two 8-core CPUs per node. The data is served to the computing processes from the Luster file system via a 10 GBit interconnect.

The imaging was done for 2-term Multi-term deconvolution algorithm to account for the frequency dependent sky brightness distribution. This requires holding three full-sized complex-valued grids in each gridder process.

The data used was for a 106-pointing mosaic observation using the EVLA at L-Band (1 – 2GHz). Only one channel from 6 Spectral Windows (SPWs) were used to reduce to total runtime, but also trigger all the relevant computing required for wide-band wide-field imaging.

Two different algorithms were tested for gridding: the Wide-band A-Projection and the W-Projection algorithm. These were chosen since they represent higher compute load that standard gridders and are expected to be used for imaging that does benefit from HPC. The compute load and internal implementation of both these algorithms is quite different, with WB A-Projection giving higher computing load for low resolution and/or higher frequency imaging.

All tests were done via the new "tclean" interface to the imager module in CASA (Rau & The CASA Imaging Team, 2013).

4.2 Runtime Measurements

4.2.1 WB A-Projection for MT-MFS imaging

As mentioned before, the compute load of A-Projection is higher than other gridders. Briefly, this is because the value of N_{sup} is larger for each visibility sample that is gridded. The first gridding cycle, typically to compute the PSF is even more expensive since it also computes the wide-band sensitivity pattern in this cycle. It is therefore a good candidate algorithm which will benefit from HPC. Tests were done using $1-7$ nodes, each with 16 cores (i.e. number of cores used ranged from $16-112$).

Tests were done using the data setup described in Sec. 4.1. A $2K \times 2K$ image was made with two Taylor terms. The convolution functions (CF) required during gridding are also expensive to compute. These are therefore computed and cached on the disk. However in the tests done here, the disk cache of CFs was not used.

Figure 5 show the run times as a function of number of CPU cores used. The curve in green colour (labeled "Make PSF+CFs+avgPB") shows the run time for the first gridding cycle. Since CF computational load is a one-time and a constant cost, the blue curve labeled "Make PSF+avgPB" represents the cost of the first gridding cycle if the disk cache of CFs is used (i.e., CFs are not computed). Cost of CF computations in the current implementation is constant with number of cores used and was $\sim 22sec$ for this run. The curve labeled "Make Residual" shows the run time for making the residual image(s). For comparison, the black curves labeled "T_o/N_{cores}" show the theoretically expected runtime scaling. The curve in red labeled "Select" shows the overhead to setting up the imager processes. This cost, while small keeps increasing with the number of cores used.

The run time for data selection is shown by the two curves at the bottom (in red and black). The red curves show the time difference between sending the data selection commands to all cores and the time when the selection finishes. The black curve labled "Imager Data Selection" is the time the code in C++ takes to finish the data selection. Clearly, most of the time is taken in the C++ data selection code.

4.2.2 Multi-thread W-Projection for MT-MFS imaging

The W-Projection gridder in CASA has been implemented as a multi-threaded gridder using OpenMP. Since it also has higher computing load than standard gridders, it is a good candidate algorithm to test parallelization efficiency. Additionally, since it is also multi-thread, it also offers the opportunity to test the run time performance for the two-level parallelization discussed in section 1, Eq. 4.

The first test was essentially the same, replacing WB A-Projection with W-Projection for gridding. Second test was done by running only one MPI process per node but with 16 threads per process to parallelize gridding. The results of the tests are shown in Fig. 6.

1. **Multi-threaded gridder:** In this test, a single imager was run per node, each with 16 threads. The curves labeled "Make PSF" and "Make Res" show the run time scaling for computing PSF and residual images. The curve in red labeled "Data Selection" shows the cost of setting up the imagers. This cost is significantly smaller (since significantly smaller number of imagers need to be setup) and the cost in fact keeps reducing with number of cores. The curves for theoretical scaling are labeled as "$N_o * 19/N_{cores}$" (see section 5).

2. **Singe-threaded gridder:** In this test, 16 imagers were run on each node, with each gridder running a single thread (same as in the A-Projection tests). The equivalent curves for PSF, Residual computations and cost of setting up the imagers are labeled with "NO_OMP". The cost of PSF and residual image computation is significantly lower but as before, the cost of setting up the imagers increases with number of cores.

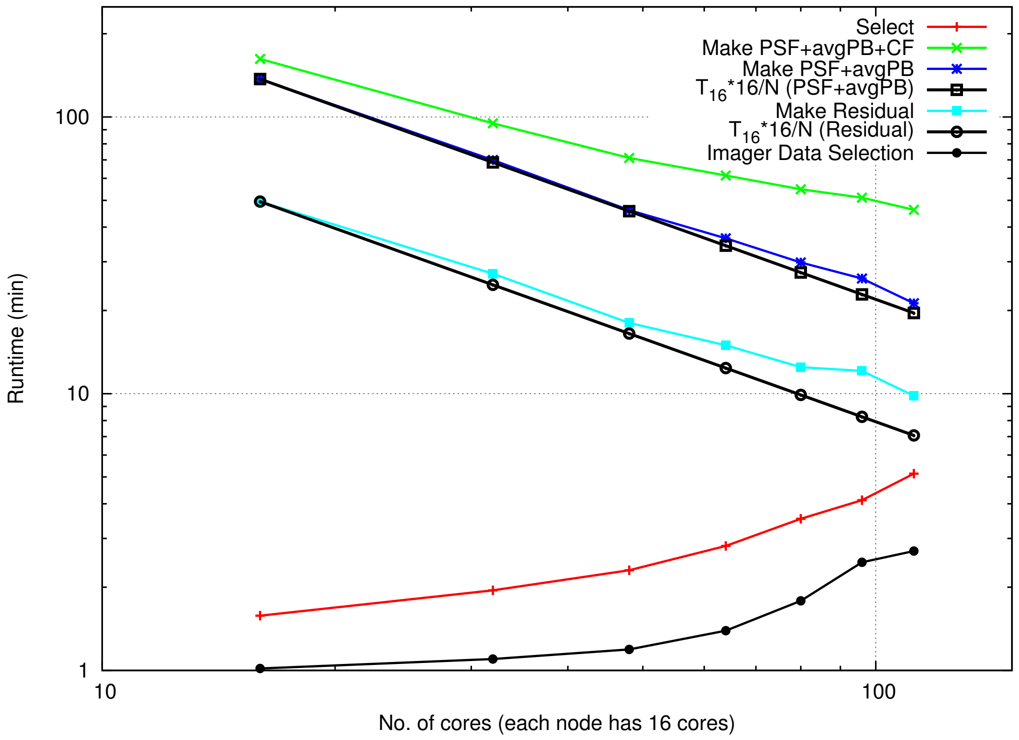

Figure 5: Run time performance for WB A-Projection for 2-term MT-MFS imaging. The curves labled "$T_{16}*16/N$" show the theoretically expected scaling with number of cores. The curve labled "Imager Data Selection" is the time it takes for the C++ code to finish the data selection at each core.

5 Discussion

From Fig. 5 a few conclusions are clear. The run time for gridding scales quite well and quite close to linear with number of cores. It slowly deviates from linear scaling as the number of cores increase. This we think could be due to the increase fraction of run time spent in setting up the imagers, which keeps increasing with number of cores. For large number of cores, the run time becomes comparable to the time it takes to setup the imagers. The cost of setting up the imagers therefore needs to be investigated.

The tests with W-Projection with and without multi-threading shows some surprising results. Firstly, the runtime with using 16 OpenMP threads scales much more close to linear than with A-Projection. This could be (a) due to better utilization of the computing resources, and/or (b) lower and flatter scaling of the cost of setting up the imagers. Note that while the scaling of the run time for PSF and residual computations with OpenMP is closer to linear, the measured curves match the theoretical curves (labeled "$T_o * 19/T_{cores}$") as if there were 19 cores per node as against 16 cores in the hardware. One way to interpret this result is that there is a $\sim$ 15% overhead due to multi-threading (a small overhead, we feel).

However the run time without OpenMP threads for residual computations is significantly lower than that with multi-threaded gridding. The run time for computing the PSF is almost flat with number of cores. Both these results we do not understand yet. The cost of setting up the imagers is similar and rising with number of cores and in fact always higher than the cost of computing residual images (i.e., increasingly more time was spent in setting up the imagers than in making the residual image).

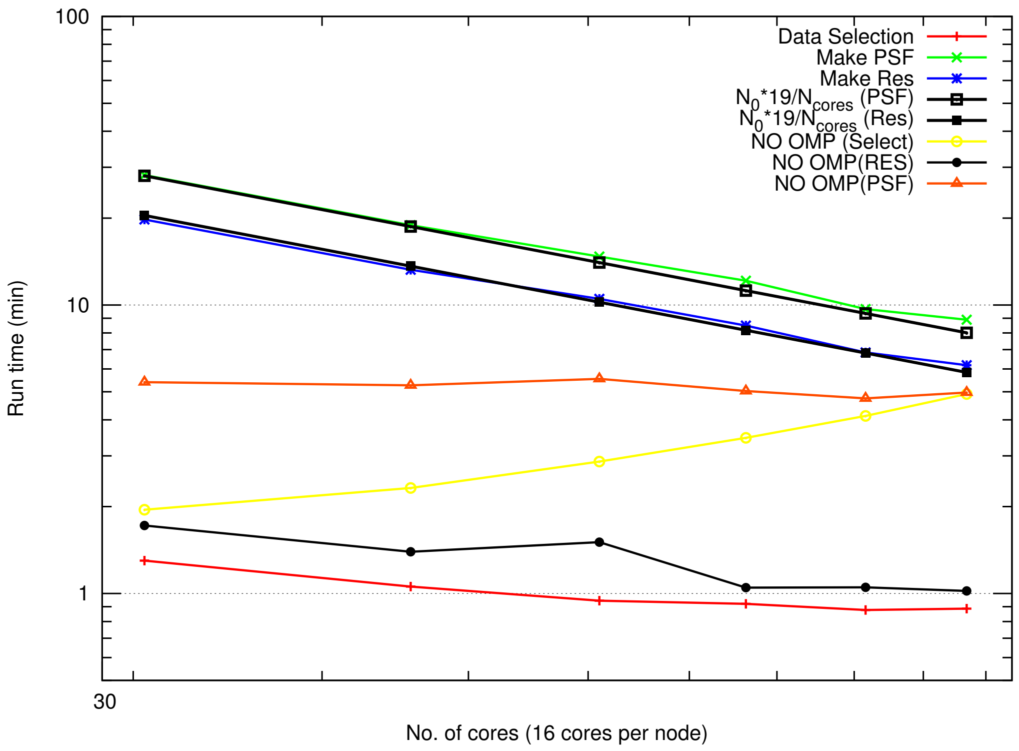

Figure 6: Run time performance for W-Projection for 2-term MT-MFS imaging. The curves in black show the theoretically expected scaling with number of cores. The gridder was run with 16-threads per process. The tests were again run with only one thread per process (same as for A-Projection tests above). The data for the latter tests are labeled with "NO_OMP" in the plot.

6 Conclusions

The OMP multi-threaded gridded offers many architectural, functional and scaling advantages. If it can be shown to work at a comparable level as purely MPI based parallelization, it is clearly the way forward. However it requires more investigation to understand the current data (which perhaps should be taken once again to eliminate the possibility of something the catastrophically wrong when making the current measurements). This requires attention of the appropriate scientists on the team.

The cost data selection by the C++ code in imagers needs to be investigated. As is shown here, that cost becomes comparable to the the gridding cost of large number of cores and in fact higher than the cost of gridding for W-Projection.

References

Bhatnagar. 2009, Report on the findings of the CASA Terabyte Initiative: Single-node tests, Tech. rep., EVLA Memo 132

Bhatnagar, S., Ye, H., & Schiebel, D. 2009, Parallelization of the off-line data processing operations using CASA, Tech. rep., EVLA Memo 133

Golap, K. 2010, Multi-threaded Gridding using OpenMP, Tech. rep., in prep.

Gonzalez, J. 2014, CASA 4.3 Parallel Processing Framework, *https://safe.nrao.edu/wiki/pub/HPC/CasaParallelization/CASA-4.3-MPI-Parallel-Processing-Framework-Installation-and-advance-user-guide-v1.2.pdf*

Rau, U., & The CASA Imaging Team. 2013, The Re-factored Imager interface: tclean, *http://www.aoc.nrao.edu/ rurvashi/ImagingAlgorithmsInCasa/node4.html*

A Appendix: Runtime measurements for WB A-Projection

Table 1: Table of the measusrement of the runtime for imaging using WB A-Projection for 2-term MT-MFS deconvolution.

N_{cores}	T_{setup} (sec)	$T_{PSF+CF+avgPB}$ (sec)	T_{Res} (sec)	$T^{16}_{PSF+CF+avgPB} * 16/N_{cores}$ (sec)	$T^{16}_{Res} * 16/N_{cores}$ (sec)	Approx. time for tool data selection (sec)
16	94.67	9725.51	2969.65	9725.51	2969.65	61.0
32	116.85	5695.16	1628.79	4862.76	1484.83	66.0
48	138.14	4266.38	1081.60	3241.84	989.88	71.5
64	169.01	3686.59	898.05	2431.38	742.41	83.5
80	211.93	3288.32	748.84	1945.10	593.93	107.0
96	247.34	3065.64	725.16	1620.92	494.94	147.5
112	308.48	2772.98	589.64	1389.36	424.24	162.0

B Appendix: Runtime measurements for multi-threaded W-Projection

Table 2: Table of the measusrement of the runtime for imaging using the multi-threaded W-Projection algorithms for 2-term MT-MFS deconvolution.

N_{cores}	T_{setup} (sec)	T_{PSF} (sec)	T_{Res} (sec)	T^{NOOMP}_{setup} (sec)	T^{NOOMP}_{PSF} (sec)	T^{NOOMP}_{Res} (sec)	$T^{32}_{PSF} * 19/N_{cores}$ (sec)	$T^{32}_{Res} * 19/N_{cores}$ (sec)
32	77.99	1691.40	1186.69	117.14	324.20	103.19	1682.66	1226.64
48	63.38	1133.92	794.61	139.32	316.16	83.68	1121.77	817.76
64	56.67	882.70	629.69	172.06	332.63	90.43	841.33	613.32
80	55.28	727.97	508.96	207.76	301.89	62.90	673.06	490.66
96	52.58	580.63	410.73	247.60	284.73	63.04	560.89	408.88
112	53.19	533.58	370.83	294.71	298.52	61.27	480.76	350.47