Parallelisation of the Wide-Band Wide-Field Spectral Deconvolution Framework DDFacet on Distributed Memory HPC System

Nicolas Monnier, Erwan Raffin, Cyril Tasse, Jean-François Nezan, Oleg M Smirnov

Parallelisation of the wide-band wide-field spectral deconvolution framework DDFacet on distributed memory HPC system

Nicolas Monnier*†, Erwan Raffin*, Cyril Tasse‡, Jean-François Nezan§, Oleg M. Smirnov¶

*CEPP - Center for Excellence in Performance Programming, Atos Bull, 35700 Rennes, France
†Univ Paris-Saclay, CNRS, Centralesupelec, L2S, 91912 Gif sur Yvette cedex, France
‡GEPI, Observatoire de Paris, CNRS, Univ Paris Diderot, 92190 Meudon, France
§Univ Rennes, INSA Rennes, IETR, CNRS UMR6164, 35700 Rennes, France
¶SKA South Africa, Pinelands, 7405 South Africa

***Abstract*—The next generation of radio telescopes, such as the Square Kilometer Array (SKA), will need to process an incredible amount of data in real-time. In addition, the sensitivity of SKA will require a new generation of calibration and imaging software to exploit its full potential. The wide-field direction-dependent spectral deconvolution framework, called DDFacet, has already been successfully used in several existing SKA pathfinders and precursors like MeerKAT and LOFAR. However, DDFacet has been developed and optimized for single node HPC systems. DDFacet is a good candidate for being integrated into the SKA computing pipeline and should, therefore, have the possibility to be run on a large multi-node HPC system for real-time performance. The objective of this work is to study the potential parallelism of DDFacet on multi-node HPC systems. This paper presents a new parallelization strategy based on frequency domains. Experimental results with a real data set from LOFAR show an optimal parallelization of the calculations in the frequency domain, allowing to generate a sky image more than four times faster. This paper analyses the results and draws perspectives for the SKA use case.**

***Index Terms*—High Performance Computing, Square Kilometer Array, DDFacet, Parallelization**

I. Introduction

The Square Kilometer Array (SKA) is the World's largest mega-science project [1]. Besides building a radio telescope observatory to make fundamental discoveries about the universe over the next 50 years with the broadest range of science of any facility worldwide, it is also the World's largest Big Data project and the largest international computing collaboration. Thousands of dishes and the million antennas of the SKA equipment will generate unprecedented scale of data to compute in order to convert them into science for the astronomers. The equipment will be built in South Africa's Karoo region and Western Australia's Murchison Shire for many scientific and technical reasons. Processing the vast quantities of data produced by the SKA will require very high-performance central supercomputers capable of in excess of 100 petaflops of raw processing power. The data computations will start on site and distributed all over the World.

As an intermediate step towards SKA, MeerKAT is a radiotelescope already operational, delivering real data to process [2]. First images of the sky have been provided by MeerKAT in March 2017 from 16 dishes and in July 2018 from 64 dishes. It delivered unprecedented resolution and image quality. The raw data provided by MeerKAT in a few hours involves the computations of $10^{11} \times 10^9$ data to process. Compared with previous telescopes, MeerKAT shows that this new generation of radiotelescopes must take into account direction dependant signal distortions in the computing of data in order to benefit from the telescope sensitivity. A new generation of calibration and imaging algorithms are studied by astronomers and implemented in the DDFacet (Direction Dependent Facet) library. As data from MeerKAT can be stored, previous work on DDFacet aims to reduce the computation time, but real-time is not mandatory. Computations are shared inside a single HPC node with shared memory.

The SKA deployment will be done into two phases: SKA1 in 2020/2024 and SKA2 in 2030+. SKA1-LOW will be made up of 130 000 phase antennas in Australia and SKA1-MID of 200 dishes in South Africa. SKA1-MID will produce $10^{13} \times 10^{11}$ data to compute (10 000 times more than Meerkat). SKA2 in 2030+ will scale by a factor 10 the number of dishes. SKA will thus provide too much data to be stored, which means than the computations of the calibration and imaging algorithms will have to be done in real-time. The challenge of SKA in terms of computing is huge, and a dedicated multi-node HPC system will have to be designed.

This paper presents the results obtained when distributing DDFacet on a multi-node HPC system. We show how the parallelization of the wideband wide-field direction-dependent spectral deconvolution algorithms of DDFacet has been implemented. The challenge is here to distribute data and computations on the distributed memory architecture of a multi-node HPC system, especially when addressing the balance between data transfers between compute nodes and performance.

The Layout of this paper is as follows. After the related work presented in section II, an overview of DDFacet is given

in section III. The distributed memory parallelization scheme we implemented is described in section IV, the experiments we performed, and the associated results are presented in section V. Finally, conclusion and future work are found in section VI.

II. Related Work

Many different imager exist in radio-astronomy. Each has specific specifications depending on the targeted radiotelescope or the targeted hardware to compute on. The main reference in the domain is CASA (Common Astronomy Software Applications) [3], which was developed to support data post-processing needs of radio astronomical telescopes such as ALMA and VLA in the 90'. CASA package is also continuously updated to support the most recent scientific advances with the third generation of radiotelescope Imaging. A new generation of algorithms is under development to take the most of the considerable number of antennas in SKA and to take into account real-time and energy constraints of the SKA.

The very popular imagers as the WSClean [4] or the ASKAPSoft [5] are widely used. Moreover, a lot of research and engineering is done to build new imagers that could scale and handle the data from the SKA radio telescope and take the Direction Dependent effects (DDE) into account. For instance, in [6], an uv-faceting algorithm allows a multi-node multi-GPU parallelization to deal with wide-field effects of recent radiotelescopes. Recently, Veenboer et al. presented innovative work with Image Domain Gridding algorithm on GPU [7] and FPGA [8]. The work from Young et al. on A-stacking framework [9] or from Sullivan et al. about Fast Holographic Deconvolution [10] are also alternative ways to deal with these effects. Most of the effort in the domain do not consider a full pipeline but is focused on the gridding/degridding algorithms, which are the most computing demanding algorithms of the Imaging framework in the case of SKA.

This paper presents the first $3rd$ generation of calibration and imaging pipeline parallelized on shared and distributed memory, DDFacet, which is mandatory to handle the data deluge envisioned by the SKA.

III. DDFacet: facet based imager

This section presents the facet based imager DDFacet with its main characteristics. A focus on its software architecture, including the shared memory parallelization scheme already used, is also given.

A. Imaging in radio interferometry

An interferometer array is a network of antennas and dishes distributed spatially on earth (or space). Each antenna's pair is defined as a baseline (physical distance between the antennas). Each baseline, at a time t and frequency ν generate a sample by correlating the signals of these two antennas. This measurement is called visibility. For a non-polarized interferometer, the number of visibilities generated is:
$N_{vis} = nb\_baseline$ x t_integ x $nb\_channel$, where t_integ is the number of sample for one baseline during the integration time, $nb\_channel$ is the number of frequency channel.

The Radio Interferometric Measurement Equation (RIME [11]) describes the relationship between the sky model and the various direction-dependent and direction-independent Jones matrices, map to the measured visibilities as a linear transformation. For a given antenna pair pq, at time t and frequency ν, the RIME is :

$$\mathbf{V}^{meas}_{pq,t\nu} = \mathbf{G}_{pt\nu}\left(\int \mathbf{D}_{pt\nu,s}\mathbf{K}_{p,s}\mathbf{I}_s\mathbf{K}^H_{q,s}\mathbf{D}^H_{qt\nu,s}d\boldsymbol{s}\right)\mathbf{G}^H_{qt\nu} + \epsilon \tag{1}$$

Where $\mathbf{G}_{pt\nu}$ is the 2x2 direction-independent Jones matrix for the antenna p, $\mathbf{D}_{pt\nu,s}$ is the 2x2 direction-dependent Jones matrix, $\mathbf{K}_{p,s}$ is the effect of the array geometry and correlator and is purely scalar, such as

$$\mathbf{K}_{p,s}\mathbf{K}^H_{q,s} = e^{-2i\pi(ul+vm+w(n-1))}$$

, $\mathbf{I}_s$ is the four-polarization sky contribution for the direction $s = (l, m, n)$, and ϵ is a 2x2 random matrix following a normal distribution.

If $\boldsymbol{\mathcal{G}}_{pq} = \mathbf{G}^*_{qt\nu} \otimes \mathbf{G}_{pt\nu}$ and $\boldsymbol{\mathcal{J}}_{pq,s} = \mathbf{D}^*_{qt\nu,s} \otimes \mathbf{D}_{pt\nu,s}$ are the direction-independent and direction-dependent 4x4 Mueller matrices for the baseline pq, at time t and frequency ν, then eq (1) can be written :

$$Vec(\mathbf{V}^{meas}_{pq,t\nu}) = \boldsymbol{\mathcal{G}}_{pq}\int \boldsymbol{\mathcal{D}}_{pq,s}Vec(\mathbf{I}_s)e^{-2i\pi(ul+vm+w(n-1)} + \epsilon \tag{2}$$

Where $Vec()$ is the vectorization operator and $\otimes$ the Kronecker product. The effects in $\boldsymbol{\mathcal{G}}_{pq}$ describe the direction-independent effects such as the individual station electronics, or their clock drifts and offsets. The effects in $\boldsymbol{\mathcal{D}}_{pq,s}$ describes the DDE, which include the ionospheric effect such as the Faraday rotation or phase shift, and the phased array station's beam that depend on time, frequency and antenna.

From eq (2), if the W-term ($e^{-2i\pi w(n-1)}$) and the DDE ($\boldsymbol{\mathcal{J}}_{pq}$) are not taken into account, the relation between the sky and the visibilities becomes a Direct Fourier Transform (DFT). Since the cost of the DFT, $O(N_{vis}N^2_{pix})$, becomes quickly prohibitive, we rather use the Fast Fourier Transform (FFT) algorithm ($O(N^2_{pix}logN_{pix})$). However, the visibilities are not sampled on a uniform grid, which is mandatory to use the FFT. To generate a sky image, the visibilities are first *gridded* on a uniform grid by applying a convolution, and then a 2D inverse FFT is performed on the grid to obtain the so-called dirty image $\widehat{y}$. The w-term and the DDE from eq (2) are taken into account during the gridding step. In a similar way, we define *degridding* as the step to obtain visibilities from the Fourier transform of the discretized sky image.

To obtain the best approximation of the *true* sky, a CLEAN Cotton-Schwab algorithm is often used during the imaging step (gridding and inverse FFT) [12]. After the imaging step, one or more "bright" sources are extracted and added to our sky model $\widehat{\pi}$ through a deconvolution step. From this

sky model, the visibilities are *predicted* and subtracted to the measure visibilities. This process is then repeated until the sky model converges to the *true* sky.

B. wideband wide-field direction-dependent spectral deconvolution framework

It is extremely challenging to synthesize high-resolution images with the new generation of radiotelescope like LOFAR. This latter operates at very low frequency, observing a wide field of view and combining short and long baselines.

The estimation of the true sky $\mathbf{I}$, assuming the Jones terms constant, from eq (2) is done by the imager DDFacet (Direction Dependant Facet) [13] [15]. A specificity of DDFacet framework is to solve the imaging problems due to the DDE by using the faceting approach. The purpose of faceting is to approximate a wide field of view with many small narrow-field images. One independent grid is used per-facet and a constant DDE $\mathcal{D}_{pq,s_\varphi}$ (where s_φ is the direction of the facet φ) is applied to each facet. Experience says that we need to split the sky model into a few tens of direction to be able to describe the spatial variation of the direction-dependent Jones matrices [16].

DDFacet also incorporates a few deconvolution algorithms that natively incorporate and compensate for the DDE. For the next of this paper, every mention of the deconvolution algorithm will reference the Hybrid Matching Pursuit (HPM) algorithm described in [15].

C. Multi-Core parallel processing

DDfacet is currently implemented as a functional software based on python and C for the intensive computational kernels. A naive description of the imaging framework is described by Fig 1. In this example, we are working with a set of measured visibility $\boldsymbol{V}_\nu$ covering the frequency band ν , where ν can be divided into several frequency channels ν_i, each channel covering more or less the same bandwidth. The imaging framework consists of calculating through an FFT and degridding the visibilities for a specific frequency channel from the sky model $\widehat{\pi}_{\nu_i}$ for that specific frequency. Those are then subtracted from the measured visibilities $\boldsymbol{V}_{\nu_i}$ producing residual visibilities. The residual visibilities are then gridded, for each channel, onto a 2D uv-plan with a different plan for each channel. All these plans are then transformed into the time domain through an FFT^{-1}, forming a residual image $\delta\widehat{y}$ which is a cube made up of slices containing information for a specific frequency channel. Each Minor Cycle of the deconvolution task aims to extract the brightness pixel of the residual image and to remove information around this pixel due to the Point Spread Function (PSF). The position of the pixel extracted is then added to the sky model. The number of Minor Cycles depends on a threshold set by the user and usually also on the value of the current brightness pixel. After the execution of Minor Cycles, an updated model $\widehat{\pi}_{+1}$ of the sky is generated. This entire process is repeated in a Major Cycle until the global stopping criteria, also set by the user, is reached.

The framework has a native single-node multiprocess execution hand-written for an optimal asynchronous behaviour between the computational and I/O phases, based on a mimic of the Concurrent Futures python module[1]. The main process of the imager manages a bunch of workers, where each worker is a process, depending on the number of cores available or set by the user. The main process then dispatches the different jobs in a dedicated queue, a compute queue for the computational jobs, or an I/O queue for the I/O relative jobs. Thus, each available worker takes and executes a job from these queues when they are not empty.

DDFacet is working on a facetted image except for the deconvolution part where all the facets are merged. Each facet φ can be processed independently of the others. As shown in Fig. 1, for each frequency channel, the FFT, FFT^{-1}, gridding and degridding are computed per facet. Thus, these algorithms can be computed in parallel on the available processor cores.

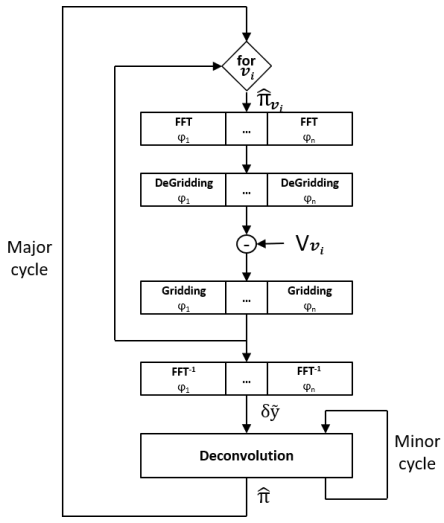

Fig. 1. DDFacet Single Node representation. Where φ is a facet, π is the sky model, gs is the gridding result and ν contains the visibilies for a set of frequencies ν_i.

D. Profiling

We performed a profiling of DDFacet to study the distribution of the total execution time. This profiling has been done using 24 Measurement Sets (or MS file) from the LOFAR survey and covering the 120MHz-165MHz frequency band. Each MS file contains Visibilities and metadata for a specific frequency channel ν_i. The size of the sky image generated was 10.000 x 10.000 pixels. The framework DDFacet has been run on an Atos Bull dual-socket compute node equipped with Intel Skykake processors (Intel Xeon Gold 6130) and 192GB RAM DDR4 memory at 3200 MT/s. Each of those processors has 16 physical cores that can be viewed as 32 logical cores when hyperthreading is enabled, so 64 logical cores per compute node. According to previous performance evaluations, it has

[1]https://docs.python.org/3/library/concurrent.futures.html

been shown that hyperthreading offers a performance gain [15]. Thus we enabled it for our experiments. In total 64 processes (1 process per core) could run in parallel on the node. The profile obtained in such conditions is given in Fig.2.

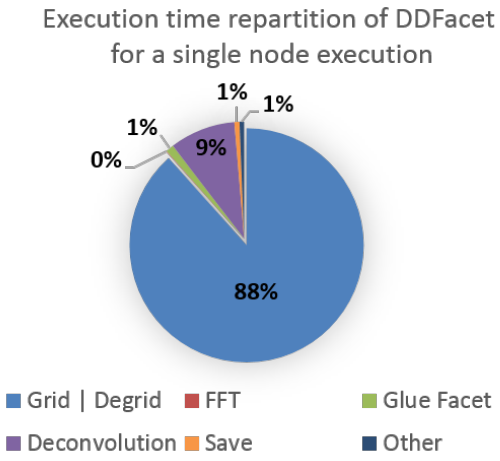

Fig. 2. Distribution of the total execution time of DDFacet running on a single node, using logical 64 cores (hyperhtreading enabled) for the intra-node parallelism.

The total execution time is divided into few main tasks: **gridding/degridding**, FFT (also including FFT^{-1}), **Glue Facet** corresponding to the fusion of all the facets to create a single image, **Deconvolution**, **Save** corresponding to the intermediate and final saving of the sky models or the final image and the last part called **Other** including all the other and not significant computing time. The most critical function is the intensive compute kernel, which does the gridding/degridding computation. Indeed, the execution time of this kernel corresponds to 88% of the total execution time. The second most important function is the deconvolution and corresponds to 9% of the total execution time. All the other times are less significant and will not be discussed in this paper.

Computing result presented in [15] has shown how the existing intra-node parallelization is important and efficient to decrease the gridding time. However, with the huge amount of data to process in a project like SKA, this result is limited by the number of cores in a node and will not be able to scale between. Thus, keeping the same algorithm architecture, a way to reduce the total time to obtain a sky image is to distribute the compute over many nodes.

IV. Parallelization on a distributed memory system

This section presents three strategies to reduce the latency of DDFacet, thanks to multi-node parallelization. Then, we will discuss in more detail the implementation of the chosen one.

A. Choice of the parallelization strategy

The 1^{st} **strategy** is the simplest one, based on batch parallelization. The nodes generate each one independent sky image. Each node works independently with the others using an independent data set. The advantage of this solution is to be very simple to set up and to increase the throughput highly. However, the latency to generate an image is still the same, making this strategy not viable for our study case. Moreover, it is important to note that in the case of SKA, where many data must be computed in real-time, using only one node to generate an image of the sky will be a huge bottleneck. Indeed, a significant amount of memory would be needed to handle all the data, while current memory is significantly expensive.

This 2^{nd} **strategy** is the extension of the existing intra-node parallelization of DDFacet. The idea here is to extend the facet parallelization from multi-core to multi-node and multi-core. In this case, to generate a sky image, each node works on specific facets of the image, then all the nodes merge their results on a single node to process the deconvolution step. In that way, the gridding/degridding most, computing step, is distributed on several nodes and makes potential latency gain. However, this method requires a considerable work of refactoring the existing code since we are working on the same parallelization level than the existing one. Our goal here is to modify the code as less as possible, so this solution becomes hardly doable. Moreover, each facet needs visibility data from all the frequency channels. Therefore each node shall access each MS file in memory, and thus this solution requires a significant amount of memory access.

The 3^{rd} and **last strategy** is focused on the independence of the computes between each frequency channel. In this case, each node will compute visibilities from one or several specific frequency channels of the same data set. The whole frequency channels covering the frequency band of our data set. Then, the result of each node will be merged into one node, which will do the deconvolution step. The advantage of this solution is, like the previous one, to dispatch the gridding and degridding work onto several nodes making possible a potential latency gain. But, unlike the previous strategy, the nodes compute visibilities for specific frequency channels; therefore, each node only needs to access specific MS files corresponding to these frequencies. Thus, memory access is limited. Finally, the parallelization is done at another level than the existing one, making the implementation much easier.

B. Parallelization implementation

The third strategy is based on the data independence between frequency channels as the data of each frequency channel is stored in an MS file. Thus, MS files are computed independently. Fig.3 represents the multi-node third strategy parallelization. This figure shows an example where DDFacet

is parallelized on k nodes, and visibilities for the frequency band ν are subdivided into $N\nu$ frequency channels. $N\nu$ is set to be a factor of the number of node k. Thus, each node computes the same number of MS files in order to balance the workload equally over the nodes. Each node computes its specific visibilities, from the frequency channels ν_i, where $\nu_i \subset \nu$, ν_i may be composed by one or few frequency channels, to generate a subresidual image $\delta\widehat{y}_{1..k}$ specific to the node. These subresidual images $\delta\widehat{y}_k$ are also hyperspectral cubes containing information only for the frequencies ν_i (specific to the node) and 0 for all the other frequencies of the total frequency band ν. These subresidual images are then collected and summed by a single node, which we call "master node", to create a single residual image $\delta\widehat{y}$. This residual image is identical to the one generated by the single-node version of DDFacet. The master node is in charge of the deconvolution step to update the sky model. This model is then broadcasted to all the nodes. It is important to note that the "per facet" intra-node parallelization is still the same.

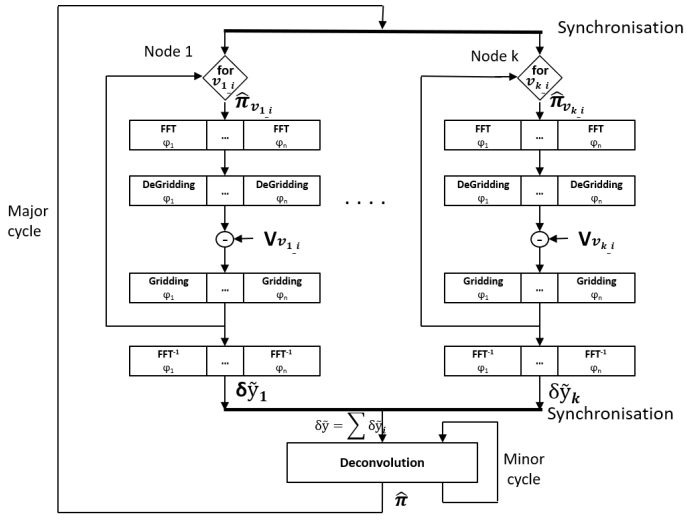

Fig. 3. Multi-node parallelization on independent frequency channels.

The software implementation of this level of parallelism has been done using MPI through mpi4py[2] on python [19]. Then, each node is seen as a specific rank from 0 to $k-1$, where rank 0 corresponds to the "Master node".

The multi-node parallelization has been achieved with success. All the results are discussed in section V.

V. Experimental results

The experiments have been performed on the same type of nodes as for the profiling presented in subsection III-D with dual-socket Intel Xeon Gold 6130. The nodes were interconnected using a Mellanox EDR interconnect. The same parameters and stopping criteria have been set for each execution of DDFacet: a final image size of 10000x10000 pixels, an intra-node parallelization on the 64 logical cores, 24 MS files, and stopping criterion depending on the quality of the final image. The performance tests have been done on a number of nodes multiple of the number of MS files used, which corresponds in our case to 2, 3, 4, 6, 8, 12, and 24 nodes, representing a total of 1536 logical cores. The speedup result is shown by Fig 4.

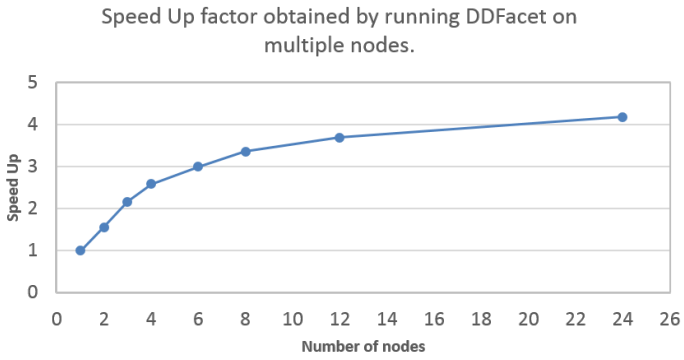

Fig. 4. Speedup of the distributed version of DDFacet over nodes.

As shown in the above figure, the speedup increases depending on the number of nodes used. However, the curve is not linear, and the gain on 24 nodes (speedup of 4.182) is questionable regarding the gain on 12 nodes (speedup of 3.699). To explain this global behaviour, we analyzed the distribution of the total execution time on the master node for each case, as shown in the Fig 5.

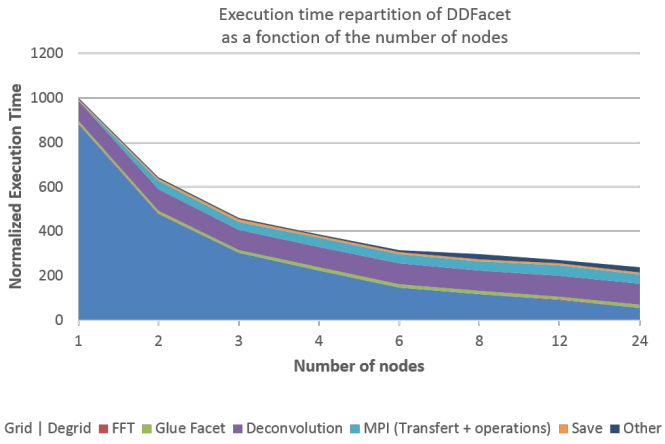

Fig. 5. Evolution of the normalized execution time of DDFacet depending on the number of nodes

On Fig 5, the step on one node corresponds to the profiling shown in Fig 2 in another shape. We can note, from two nodes, a new MPI time (light blue in the figure) corresponding to the time spends in MPI communication and all the waiting time due to the synchronized communications. As the gridding (dark blue in the figure) is the only part distributed over nodes,

[2]https://mpi4py.readthedocs.io/en/stable/

the time per node to compute the gridding and degridding operations decrease depending on the number of nodes used. On a 24 nodes server, the gridding/degridding step represents 23% of the total executing time, while it represented 88% on the single node implementation. Deconvolution represents 40%, MPI 17%, and the addition of all the other timings represents 20%.

These results show that the speedup is limited by all the timings which are not distributed over nodes. The ideal speedup equal to the number of nodes is hardly obtained due to data transfers. The gridding/degridding distributed over nodes shown in Fig 6 is close to the ideal speedup being linear with the number of nodes and being 16 for an execution onto 24 nodes. To obtain such a speedup to the whole imager, we have to ensure that: $t_{grid/degrid} >> t_{deconv} + t_{mpi} + t_{other}$, what is expected when working on a much bigger test case like SKA1.

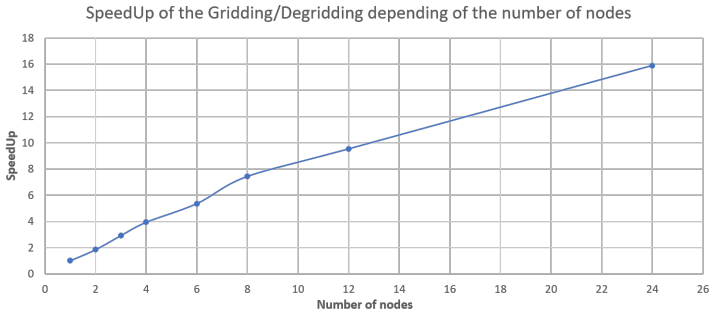

Fig. 6. Gridding/Degridding speedup of the distributed version of DDFacet over nodes.

VI. Conclusion and Future Work

In this paper, we have presented the parallelization of DDFacaet, a wideband wide-field spectral deconvolution framework on distributed memory HPC system. DDFacet is one of the best candidates as an imager for SKA regarding its capacities to deal with DDEs from new generation of radiotelescope.

The results of our MPI parallelization over several nodes on the existing shared memory parallel implementation with a real data set from LOFAR have been analyzed. The obtained hybrid parallelization has speedup the image generation of the radio sky up to a factor of 4.2 for the whole program and up to 16 for the Gridding/Degridding kernel on 24 compute nodes. The acceleration of the whole program is limited by the size of the data set used for this study, which won't be the case with the large data sets of SKA. We showed in this paper that the parallelization potential of DDFacet on multi-node HPC systems is very high. This potential is a crucial feature for the selection of DDFacet in the SKA final computing system. Finally, our experiments also show the importance of the data format for the multi-node parallelization, which is currently discussed in the project.

For future work, we plan to explore the use of accelerators as GPUs in perspective with this new hybrid parallelization scheme to even more reduce the time to solution. Moreover, we also plan to focus on the algorithmic part of the imager. Especially on the actual partitioning of the gridding and degridding steps that make them two independent part of the algorithm, which could be modify to become a single step.

References

[1] Acero, F. and others, "IFrench SKA White Book - The French Community towards the Square Kilometre Array," 2017.

[2] O. Smirnov, "Modern Radio Interferometric Imaging Challenges: From MeerKAT Towards the SKA," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), unpublished.

[3] Jaeger, S., "The Common Astronomy Software Application (CASA)," Astronomical Data Analysis Software and Systems XVII, vol. 394, pp 623, 2008.

[4] Offringa, A. R. and McKinley, B. and Hurley-Walker, and et al., "wsclean: an implementation of a fast, generic wide-field imager for radio astronomy," Monthly Notices of the Royal Astronomical Society, vol. 444, pp 606–619, 2014.

[5] M. Whiting and S. M. Ord and D. Mitchell and M. Voronkov and J. C. Guzman, "2018 2nd URSI Atlantic Radio Science Meeting (AT-RASC), 2018.

[6] Lao Bao-qiang and An Tao and Yu Ang and Guo Shao-guang, "Research on Parallel Algorithms for uv-faceting Imaging," Chinese Astronomy and Astrophysics, vol. 43, pp 424–443, 2019.

[7] Veenboer, Bram and Petschow, Matthias and Romein, John W., "Image-Domain Gridding on Graphics Processors," 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 545–554, may 2017.

[8] YVeenboer, Bram and Romein, John W., "Radio-Astronomical Imaging: FPGAs vs GPUs," Euro-Par 2019: Parallel Processing, vol. 11725, pp 509–521, 2019.

[9] Young, A. and Wijnholds, S. J. and Carozzi, T. D. and Maaskant, R. and Ivashina, M. V. and Davidson, D. B., "Efficient correction for both direction-dependent and baseline-dependent effects in interferometric imaging," Astronomy & Astrophysics, vol. 577, pp A56, may 2015.

[10] Sullivan et al., "Fast Holographic Deconvolution: a new technique for precision radio interferometry)," The Astrophysical Journal, vol. 759, pp 17, 2012.

[11] O.M. Smirnov, "Revisiting the radio interferometer measurement equation. I. A full-sky Jones formalism," A&A, vol. 527, pp. A106, January 2011.

[12] A.R. Thompson, James M. Moran, and George W. Swenson, "Interferometry and synthesis in radio astronomy 3^{rd} edition," 2017.

[13] DDFacet, https://github.com/saopicc/DDFacet

[14] killMS, https://github.com/saopicc/killMS

[15] C. Tasse, B. Hugo, M. Mirmont, O. Smirnov, M. Atemkeng, L. Bester, M.J. Hardcastle, R. Lakhoo, S. Perkins, and T. Shimwell, "Faceting for direction-dependent spectral deconvolution," A&A, vol. 611, pp. A87, 2018.

[16] Shimwell, T. W. and Tasse, C. and Hardcastle, M. J. and Mechev, A. P. et al., "The LOFAR Two-metre Sky Survey - II. First data release," A&A, vol. 622, pp. A1, 2019.

[17] S. Bhatnagar, and T. J. Cornwell,and K. Golap, and Juan M. Uson, "Correcting direction-dependent gains in the deconvolution of radio interferometric images," Astronomy & Astrophysics, vol. 487, pp 419–429, 2008.

[18] C. Tasse, S. van der Tol, J. van Zwieten, G. van Diepen and S. Bhatnagar, "Applying full polarization A-Projection to very wide field of view instruments," Astronomy & Astrophysics, vol. 553, pp A105, 2013.

[19] Gonzalez et al, "Python Code Parallelization, Challenges and Alternatives," Astronomical Data Analysis Software and Systems XXVI ASP Conference Series, vol. 521, pp 515, october 2019.