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

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

# PARALLELISATION OF THE WIDE-BAND WIDE-FIELD SPECTRAL DECONVOLUTION FRAMEWORK DDFACET ON DISTRIBUTED MEMORY HPC SYSTEM

*Nicolas Monnier*[⋆1]*, Erwan Raffin*[⋆]*, Cyril Tasse*[†]*, Oleg M. Smirnov*[§]*, Jean-François Nezan*[‡]

[⋆]CEPP, Bull Atos, 35700 Rennes, France
[†]GEPI, Observatoire de Paris, CNRS, Univ Paris Diderot, 92190 Meudon, France
[§]SKA South Africa, Pinelands, 7405 South Africa
[‡]Univ Rennes, INSA Rennes, IETR, CNRS UMR6164, 35700 Rennes
[1] Univ Paris-Saclay, Centralesupelec, L2S

## 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 like MeerKAT. DDFacet has been developed and optimized for single node HPC systems. DDFacet is a good candidate for being integrated in the SKA computing pipeline and should therefore be run on a massive multi-node HPC system for real-time performances. The objective of this work is to study the potential parallelism of DDFacet on multi-node HPC systems. This paper presents a new parallelization level for multi-node HPC based on frequency domains. Experimental results with a real data set from LOFAR show an optimal parallelisation of the calculations in the frequency domain allowing to generate a sky image more than 4 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

## 1. 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 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 develering 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 develivered unprecedant resolution and image quality. The raw data provided by MeerKAT in a few hours involves computations on $10^{11} \times 10^{9}$ data to process. Compared with previous telescopes, MeerKAT shows that this new generation of radiotelecopes 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 a shared memory.

The deployement of SKA 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 multinode HPC system will have to be designed.

This paper presents the results obtained when distributing DDFacet on a multinode HPC system. We show how the parallelization of the wideband wide-field direction-dependent spectral deconvolution algotithms of DDFacet has been implemented. The challenge is here to distribute data and com-

putations between several HPC nodes a distributed memory architecture, especially when adressing the balanced between data transfers between compute nodes and performance.

The Layout of this paper is as follow. After the related work presented in section 2, an overview of the wideband wide-field direction-dependent spectral deconvolution framework DDFacet is given in section 3. The distributed memory parallelization scheme we implemented is described in section 4, the experiments we performed and the associated results are presented in section 5. Finally conclusion and future work are found in section 6.

## 2. RELATED WORK

There is a lot of existing Imager in the world of radio-astronomy; one would say one Imager per radio telescope, probably more. The biggest effort and software available in radio-astronomy 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 algorithm is under development to take the most of the huge 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 are done to build new Imagers that could scale and handle the data from the radio telescope SKA and take the Direction Dependent effects into account, such as [6]. This lastest paper present a multi-node multi-GPU parallelization of $uv$-faceting algorithm to deal with wide-field effects of recent radiotelescopes. However, the results of the paper do not concern a full pipeline since they mainly focused their efforts on the griddider and degridder. Moreover, a lot of work has been done to optimize gridding and degridding on CPU and GPU since they are, and they will be the most expensive part of the Imgaging framework in the case of SKA. Recently, Veenboer et al. presented innovative work on Image Domain Gridding [7]. We could also cite work from Young et al. on A-stacking framework [8] or from Sullivan et al. about Fast Holographic Deconvolution [9].

To our knowledge, this paper presents the first $3rd$ generation of calibration and imaging pipeline parallelized both on shared and distributed memory, DDFacet, which is mandatory to handle the data deluge envisionned by the SKA.

## 3. DDFACET: FACET BASED IMAGER

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

### 3.1.

#### *3.1.1. Imaging in 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 $\nu$ 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: $M = nb\_baseline * t\_integ * nb\_channel$, where $t\_integ$ is the number of sample for one baseline during the integration time, $nb\_channel$ is the number of frequency channel.

The Visibility function can be described by the following equation:

$$V(u,v,w) = \iint_{-\infty}^{\infty} \frac{A(l,m)I(l,m)}{n} e^{-2i\pi(ul+vm+w(n-1)} dldm \tag{1}$$

Where A is the Beam pattern, $I$ is the True sky. Under the assumption below; the Van Citter-Zernike Theorem [10] makes a Fourier relation between the observed sky and the visibilities measured. If the field of view is small, the Beam pattern quickly goes to zero, the baselines are coplanar (w small), we're observing infinitely distant sources, the sampling is continuous, and the correlation of the signals appear on a narrow bandwidth, equation (1) becomes:

$$V(u,v,w) = \iint_{-\infty}^{\infty} I(l,m)e^{-2i\pi(ul+vm)} dldm \tag{2}$$

It is here easy to recognize a 2D Fourier Transform relation. However, the cost of a Direct Fourier Transform (DFT) becomes quickly prohibitive (cost $O(MN^2)$, where $N^2$ is the number of pixel in the image) with the increasing number of Visibilities generated by recent radiotelescopes. The Fast Fourier Transform (FFT) algorithm ($O(N^2 logN)$ ) is then a better option. However, an interferometer doesn't make measurements at regular intervals, making impossible the FFT. The Visibilites have to be resampled to fit in a regular grid. This step is called "gridding" (and degridding for the opposite direction). Each Visibility is convolved with an oversampled convolution kernel then resample onto the $uv$ plane. The equation below describes this resampling step:

$$V_{gridded}(u,v) = (V(u,v)S(u,v) * C(u,v))III(u,v) \tag{3}$$

where $S(u,v)$ is the sampling function (dependent of the array configuration), $C(u,v)$ is the convolution kernel and $III(u,v)$ is the Shah function. The "Dirty" image is then obtained from the inverse Fourier Transform of the gridded Visibilities using an FFT.

$$\tilde{y} = \mathcal{F}^{-1}\{V_{gridded}\} \tag{4}$$

The Dirty image can also be expressed as an ill-posed inverse problem:

$$\tilde{y} = PSF * x + \epsilon \quad (5)$$

where $PSF$ is the Fourier Transform of the sampling function of the interferometer array $S(u, v)$, $x$ is the True sky, and $\epsilon$ is the noise assumed to be Gaussian. The best approximation of the True sky $x$ can be obtained by performing a deconvolution step. Most of the time, a CLEAN Cotton-Schwab like algorithm is used[10].

#### *3.1.2. wideband wide-field direction-dependent spectral deconvolution framework*

The new generation of radiotelescope tends to work on a wide field of view, using large bandwidth and high resolution (very long baselines). The previous assumptions making possible the equation (2) break since some terms are not insignificant anymore. Thus, a lot of baseline-time-frequency direction dependant effects on the observation appear and need to be corrected(ie. Faraday rotation due to the ionosphere at low frequency, or complex beam pattern due to the wide field of view). To correct these Direction Dependant Effects (DDEs), a special calibration has to be done on the observation measurements (3rd generation of calibration), and the effects also have to be taken into account during the Imaging process.

The Imager DDFacet[1] (Direction Dependent Face) and its calibration framework killMS[2] are currently the best tools to deal and correct the DDEs of an observation. DDFacet works on the approximation of a wide-field image by facetting the image[ref]. Each facet is considered as a narrow field that is tangent to the celestial sphere at its phase center. One independent grid is used per facet. The $AW$-projection algorithm is used for the gridding step. Algorithm based on the work of Bhalnagar et al.[11] and then applied for the specific case of DDFacet from the work of Tasse et al. [12].

### 3.2. Software implementation

#### *3.2.1. Multi-Core parallel processing*

DDfacet is currently implemented as a functionnal software base 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're working with a set of measured visibility $V_\nu$ covering the frequency band $\nu$ , where $\nu$ can be divided into several frequency channels $\nu_i$ (Each channel covers more or less the same bandwidth). The Imaging framework consists, for each frequency channel, to calculate the visibilities of the sky model $\widehat{\pi}_{\nu_i}$ for that specific frequency ($FFT$ + Degridding). Those are then subtracted from the measured visibilities $V_{\nu_i}$. The residual visibilities are gridded onto a 2D uv-plan (a different plan for each channel). All these plans are then transformed into the time domain ($FFT^{-1}$) forming a residual image $\widetilde{y}$ which is a cube where each slice contains information for a specific frequency channel. Each minor cycle (represented by the Minor Loop in the figure) of the Deconvolution task (based on CLEAN like algorithm) aims to extract the brightness pixel of the residual image, removing also information around this pixel due to the Point Spread Function (PSF). The position of the pixel extracted is then added to our sky model. After few minor cycles, depending on the threshold set by the user (usually depending on the value of the current brightness pixel), an updated model $\widehat{\pi}_{+1}$ of the sky is generated. This entire process is repeated (and called 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 behavior between the computational and I/O phases, based on a mimic of the Concurrent.Futures python module. The main process of the Imager manages a bunch of workers (depending on the number of cores available or set by the user) and dispatches the tasks in a dedicated queue. Thus, each worker can draw in this queue ....[Add something to complete]

DDFacet is working on a facetted image (except for the deconvolution part where all the facets are merged), where each facet can be processed independently of the others. As shown in Fig. 1, for each frequency channel, the $FFT$ is computed per facet. Thus, it can be computed in parallel where one facet is computed per one core of the processor. The gridding, degridding and $FFT^{-1}$ steps use the same parallelization approach.

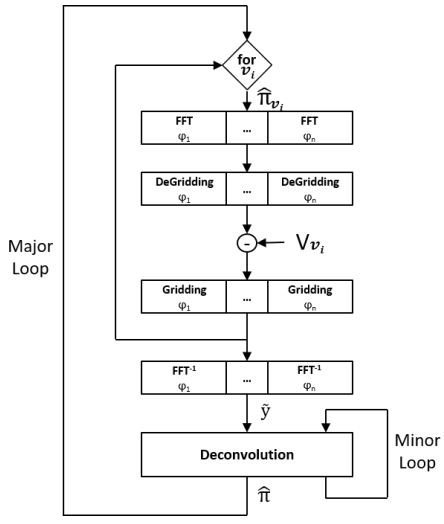


**Fig. 1**. DDFacet Single Node representation. Where $\varphi$ is a facet, $\pi$ is the sky model, gs is the gridding result and $\nu$ contains the visibilies for a set of frequencies $\nu_i$.

[1] https://github.com/saopicc/DDFacet
[2] https://github.com/saopicc/killMS

*3.2.2. 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 $\nu_i$. The size of the sky image generated was 10.000 x 10.000 pixels. The framework DDFacet has been run on a dual-socket node equipped with Intel Skykake generation processors (Intel Xeon Gold 6130). Each of those processors has 16 physical cores that can be viewed as 32 logical cores when hyperthreading is enabled. According to previous performance evaluation, it has been shown that hyperthreading offers a performance gain(see [13]), 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.

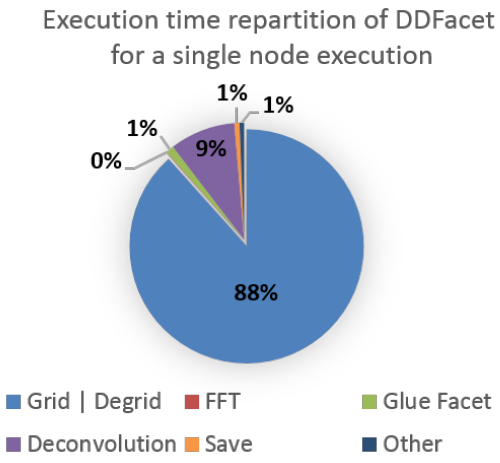


**Fig. 2**. Distribution of the total execution time of DDFacet running on a single node, using logical 64 cores (hyperhtreading enabled) for the intra-node parallelism.

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 timing are less significant regarding the two previous one, thus, we will not discuss in this paper.

Computing result presented in [13] 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 time to solution to obtain a sky image is to distribute the compute over many nodes.

## 4. PARALLELIZATION ON A DISTRIBUTED MEMORY SYSTEM

This section presents the initial ideas to reduce the latency of DDFacet, all based on using several nodes. We'll present three of these ideas and explain the final choice of one of them. Then, we'll discuss in more detail the implementation of this solution.

### 4.1. Choice of the parallelization strategy

*First strategy*

This $1^{st}$ strategy is the simplest one. Instead of working of 1 node to generate 1 image of the sky, we work on N indépendent nodes to generate N independent images. Each node works independently with the other using an independent data set. The advantage of this solution is to be very simple to set up and to highly increase the throughput. 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 a lot of data must be computed in real-time, using only 1 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, and nowadays memory is significantly expensive.

*Second strategy*

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 which will do the Deconvolution step. The advantage of this solution is to dispatch the work of the expensive part of the framework: the Gridding/Degridding (Fig. 1) making possible a potential latency gain. However, this method requires a huge work of refactoring the existing code since we're working on the same parallelization level than the existing one. Our goal here is to modify as less as possible the code so this solution becomes hardly doable. Moreover, each facet needs data (visibilities) from all the frequency channel, therefore each node shall access to each MS file in memory and thus this solution requires a significant amount of memory access.

*Third strategy*

The $3^{rd}$ and last strategy is centering 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. Moreover, unlike the previous strategy, the nodes compute visibilities for specific frequency channels, 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.

The implementation of the third strategy is discussed in the next subsection.

### 4.2. Parallelization implementation

In the previous subsection, we discussed the different ways to parallelize DDFacet over nodes on distributed memory architecture. we concluded by choosing one and its implementation will be the main subject of this subsection. As said before, the visibilities of the different frequency channels can be computed independently of the others. Visibilities for a specific frequency channel are contained in a MS file, thus, these files are computed independently of the others.

Fig.3 makes a better representation of the multi-node parallelization resulting from the independence of some compute. This figure shows us an example where DDFacet is parallelized on $k$ nodes and is computing visibilities for the frequency band $\nu$, subdivided into $N\nu$ frequency channels ($N\nu = n * k$, we want to balance the workload equally over the nodes, so each node should compute the same number of MS file). Each node computes its specific visibilities, from the frequency channels $\nu_i$ ($\nu_i \subset \nu$, $\nu_i$ may be composed by one or few frequency channels), to generate a (sub)residual image $\widetilde{y}_{1..k}$ specific to the node. These (sub)residual images are also hyperspectral cubes containing information only for the frequencies $\nu_i$ (specific to the node) and 0 for all the other frequencies of the total frequency band ${}_nu$. these (sub)residual images are then collected and summed by a single node, which we call "Master node", to create a single residual image $\widetilde{y}$. 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 broadcast to all the nodes. It is important to note that the "per facet" intra-node parallelization is still the same.

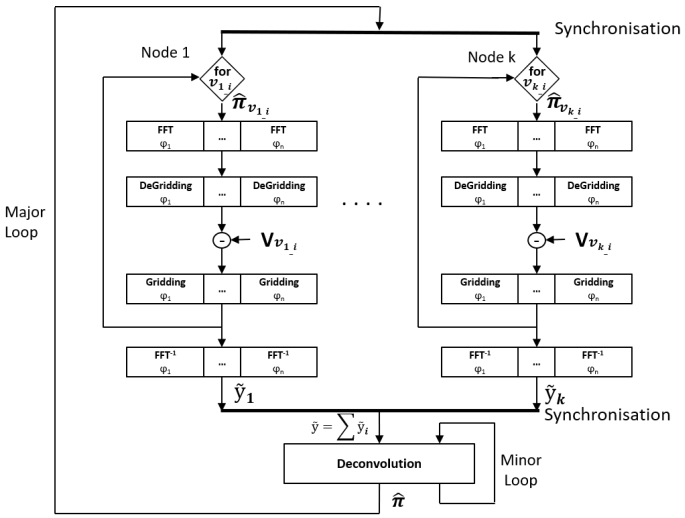


**Fig. 3**. Schema multinode.

The software implementation of this level of parallelism has been done using MPI (mpi4py[3] on python [14]). 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 will be discussed in section 5.

## 5. EXPERIMENTAL RESULTS

The experiments have been performed on the same kind of node as the profiling presented in subsection 3.2.2 (dual socket Intel Xeon Gold 6130). 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 a stopping criteria depending of 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. The speedup result is shown by Fig 4

As shown in the above figure, the speedup increase depending on the number of nodes used, thus the main goal of this study is reached. 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 behavior, we analyzed the distribution of the total execution time on the master node for each case as shown in the Fig 5 below.

On his figure, the step on 1 node corresponds to the profiling shown in Fig 2 in another shape. We can note, from 2

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

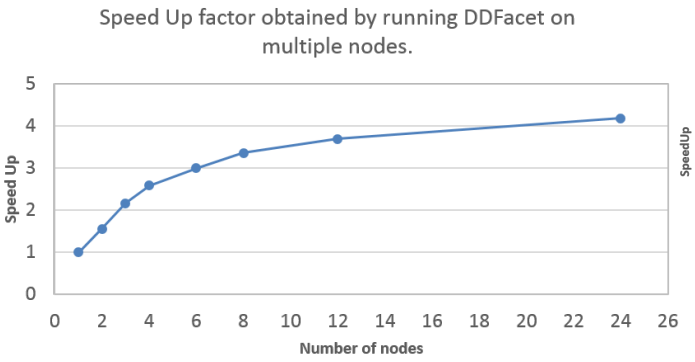


**Fig. 4**. Time to solution speedup of the distributed version of DDFacet over nodes.

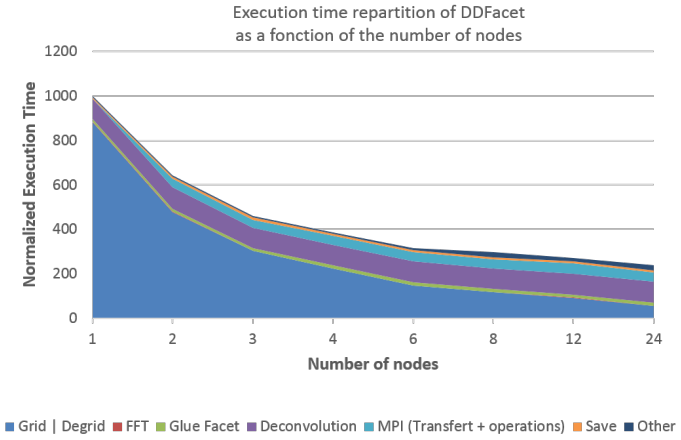


**Fig. 5**. Evolution of the normalized execution time of DDFacet depending of the number of nodes

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, the time per node to compute decreases depending on the number of nodes used. All the other timings are constant. In the 24 nodes case, Gridding/Degridding timing represents 23% of the total execution time (88% on 1 node), Deconvolution represents 40%, MPI 17% and the addition of all the other timings represents 20%.

With these results obtained using our data-set, we can see that, at some point, the efficiency of the speedup is limited by all the timings which are not distributed over nodes. The ideal would be to ensure a speedup similar to the one of the griddind/degridding distributed over nodes, as shown in Fig 6.

To obtain such a speedup, we have to ensure that:

$t_{grid/degrid} >> t_{deconv} + t_{mpi} + t_{other}$

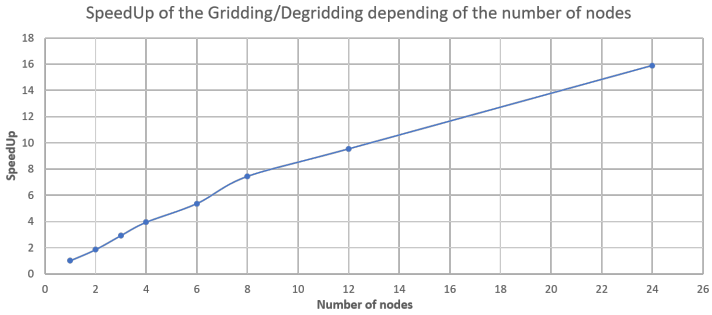


**Fig. 6**. Gridding/Degridding speedup of the distributed version of DDFacet over nodes.

Thus, we would have to use a much bigger test case.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we have presented the parallelization of a wide-band wide-field spectral deconvolution framework DDFacet on distributed memory HPC system, where DDFacet is one of the best candidates as an imager for SKA regarding its high superiority to compute data from new génération of radiotelescope.

In this study, we have shown the possibility to parallelize over nodes an existing software with only minor modifications on the global structure of the software. This parallelization has provided a speedup gain (time to generate an image of the sky from a given data set) but was limited by the size of the data set used. To keep good efficiency for an execution on a higher number of nodes, large data sets are mandatory. Thus, SKA is a perfect candidate.

The final design of the data format generated by SKA will provide better hints to design the best architecture to compute these data.

## 7. REFERENCES


[1] F. Acero et al., "French 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," in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, Oct 2018.

[3] S. Jaeger, "The common astronomy software application (casa)," *Astronomical Data Analysis Software and Systems XVII*, vol. 394, pp. 623, 07 2008.

[4] A. R. Offringa, B. McKinley, N. Hurley-Walker, F. H. Briggs, R. B. Wayth, D. L. Kaplan, M. E. Bell, L. Feng, A. R. Neben, J. D. Hughes, 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, no. 1, pp. 606–619, Aug 2014.

[5] M. Whiting, S. M. Ord, D. Mitchell, M. Voronkov, and J. C. Guzman, "High-performance pipeline processing for askap," pp. 1–1, 2018.

[6] Lao Bao-qiang, An Tao, Yu Ang, and Guo Shao-guang, "Research on parallel algorithms for uv-faceting imaging," *Chinese Astronomy and Astrophysics*, vol. 43, no. 3, pp. 424 – 443, 2019.

[7] Bram Veenboer, Matthias Petschow, and John W. Romein, "Image-Domain Gridding on Graphics Processors," pp. 545–554, May 2017.

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

[9] Sullivan et al., "Fast Holographic Deconvolution: a new technique for precision radio interferometry," *The Astrophysical Journal*, vol. 759, no. 1, pp. 17, Nov. 2012, arXiv: 1209.1653.

[10] A. R. Thompson, James M. Moran, and George W. Swenson, *Interferometry and synthesis in radio astronomy*, Springer, New York, 3nd ed edition, 2017.

[11] S. Bhatnagar, T. J. Cornwell, K. Golap, and Juan M. Uson, "Correcting direction-dependent gains in the deconvolution of radio interferometric images," *Astronomy & Astrophysics*, vol. 487, no. 1, pp. 419–429, Aug. 2008, arXiv: 0805.0834.

[12] 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: An imager for LOFAR," *Astronomy & Astrophysics*, vol. 553, pp. A105, May 2013.

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

[14] Gonzalez et al., "Python code parallelization, challenges and alternatives," in *Astronomical Data Analysis Software and Systems XXVI ASP Conference Series*, Oct 2019.