Multi-core multi-node parallelization of the radio interferometric imaging pipeline DDFacet

Nicolas Monnier, David Guibert, Cyril Tasse, Nicolas Gac, François F. Orieux, Erwan Raffin, Oleg M Smirnov, Benjamin V Hugo

Multi-core multi-node parallelization of the radio interferometric imaging pipeline DDFacet

Nicolas Monnier*, David Guibert†, Cyril Tasse‡, Nicolas Gac*, François Orieux*,
Erwan Raffin†, Oleg M. Smirnov§, Benjamin V. Hugo§

*Laboratoire des Signaux et Systèmes, Univ. Paris-Saclay, CNRS, CentraleSupélec, 91 190 Gif-sur-Yvette, France
†CEPP - Center for Excellence in Performance Programming, Atos Bull, 35 700 Rennes, France
‡GEPI, Observatoire de Paris, CNRS, Univ. Paris Diderot, 92 190 Meudon, France
‡South African Radio Astronomy Observatory, Cape Town, Western Cape, 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 been successfully used in several existing SKA pathfinders and precursors like MeerKAT and LOFAR. This imager allows a multi-core execution based on facets parallelization and a multi-node execution based on observations parallelization. However, because of the amount of data to be computed, the data on a single observation will have to be distributed on several nodes. This paper proposes the first two-level parallelization of DDFacet in the case of a single observation. A multi-core parallelization based on facets and a multi-node parallelization based on frequency distribution grouped in Measurement Sets. We show that this multi-core multi-node parallelization has successfully reduced the total execution time by a factor of 5.7 on a LOFAR dataset.**

***Index Terms*—HPC, DDFacet, Parallelization, Distributed memory, SKA**

I. Introduction

The Square Kilometer Array (SKA) is one of the World's largest science projects [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 largest Big Data project and the largest international computing collaboration. With thousands of dishes and millions of antennas, the SKA equipment will generate an unprecedented amount of data. The equipment will be built in South Africa's Karoo region and Western Australia's Murchison Shire. Processing the vast quantities of data produced by the SKA will require high-performance central supercomputers capable of more than 100 petaflops of raw processing power. The data computations will start on-site and be distributed all over the World.

As an intermediate step towards SKA, MeerKAT or LOFAR are radiotelescopes already operational, delivering real data to process [2]. The raw data provided by MeerKAT in a few hours involves the computations unprecedent amount of data. Compared with previous generation of radio telescopes, MeerKAT and LOFAR show that this new generation must consider direction-dependent signal distortions (DDEs), such as ionospheric distortion or antenna beams, in computing data to benefit from the telescope sensitivity.

Each antenna's pair is defined as a baseline (physical distance between the antennas). Each baseline, at a time t and frequency ν generates a sample by correlating the voltages of these two antennas. This measurement is called visibility. These antennas are combined into a single virtual telescope using a mathematical approach called *aperture synthesis* [3]. This technique makes images of high resolution and high sensitivity that a single extremely large receiving element could not make. However, it is also computationally intensive, with a complexity dependent on the number of visibilities generated by the radio interferometer and the quality of the desired result.

An imaging pipeline is composed of many algorithmic bricks that are easily differentiated from each other. Those responsible for the interpolation of data, applying at the same time direction-dependent corrections, called gridding and degridding, are the most expensive. Tasse [4] has shown that they can reach 80% of the total execution time. There is a lot of literature about how to reduce this computation time. On the side of the standard methods, we find computation optimization on CPU parallelization [5], as well as GPU and multi-GPU parallelization [6], [7]. At the same time, some methods allow reducing the memory consumption linked to these methods [8]. The literature also proposes alternative methods, such as Image Domain Gridding (IDG), allowing to speed up the computation while keeping a high accuracy in the results [9]. We also find the implementation of these methods on accelerator platforms such as the GPU or FPGA [10], [11].

Because of the large amount of data to be processed by current radio telescopes, computational methods have to be integrated on a larger scale by complete imaging pipelines. The main reference is CASA (Common Astronomy Software Applications) [12], 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. WSClean [13] is another more advanced imager integrating the IDG

algorithm. Both of these pipelines allow a distributed execution using MPI [14]. We can also find a pipeline, from the data generation to the image, based on the workflow execution system DALiuGE [15], allowing large-scale execution [16]. We can also find imaging pipeline based on the Dask[1] framework, such as *Codex-Africanus* [17]. Most of these pipelines and workflows have in common to dispatch computationnaly intensive jobs such as *gridding*, *degridding* and *FFT*.

A new generation of calibration and imaging algorithms, also called the 3rd generation of calibration and imaging [18], is implemented in DDFacet[2] (Direction Dependent Facet) framework to compensate and correct the DDEs. This paper presents the first two-level parallelization paradigm for the DDFacet imager to distribute the computation of observation over multi-core and multi-node. The first level of parallelization uses Concurrent Futures for multi-core parallelization on a shared memory system [4]. The second level allows multi-node parallelization on a distributed or shared memory system. The challenge is 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. An overview of DDFacet and radio interferometry is given in section II. The shared and distributed memory parallelization scheme we implemented is described in section III. The experiments we performed and the associated results are presented in section IV. Finally, the conclusion and future work is found in section V.

II. DDFacet and Radio Interferometry Imaging

This section describes the mathematical basis of radio interferometric imaging, taking into account direction-dependent effects. In a second step, we present DDFacet, an imaging pipeline that considers these effects.

A. Radio Interferometry Imaging

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 generates a sample by correlating the voltages 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 [19]) 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 :

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

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

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

, $\boldsymbol{x}_s$ 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 :

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

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. In order to generate a sky image, the visibilities are first *gridded* on a uniform grid by applying a convolution. 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. Similarly, we define *degridding* as the step to obtain visibilities from the Fourier transform of the discretized sky image.

Usually, a CLEAN major/minor cycle deconvolution algorithm is used to reconstruct the *true* sky. This kind of algorithm in the case of DDFacet is described in sec III-A.

B. DDFacet

It is extremely challenging to synthesize high-resolution images with the new generation of radio telescopes like LOFAR or MeerKAT. 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) [4]. The specificity of the DDFacet

[1]https://dask.org/

[2]https://github.com/saopicc/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 shows that we need to split the sky model into a few tens of directions to be able to describe the spatial variation of the direction-dependent Jones matrices [20]. Moreover, in order to reduce the amount of data to be processed, an averaging of visibilities is performed using the BDA algorithm [21], [22]. Depending on the baseline and a decorrelation factor considered acceptable, the visibilities are more or less averaged together.

DDFacet incorporates a few deconvolution algorithms natively incorporated and compensated for the DDE. For instance, one can use the standard Multi-Scale Multi-Frequency (MSMF) CLEAN algorithm. As it is well-known that the CLEAN-like algorithms are not robust in deconvolution of extended emission, the SubSpace Deconvolution algorithm (SSD) is implemented. As described in [4], one can also find an extension of SSD using a genetic algorithm (SSDGA). For the next of this paper, every mention of the deconvolution algorithm will reference the MSMF algorithm described in [4].

III. Parallelization

This section describes the parallelization paradigms implemented in DDFacet. First, we will see the simplified sequential algorithm to introduce notations. Then, we will see the multi-core parallelization with a modified implementation compared to the native version to simplify the implementation of the multi-node parallelization. Finally, we will see the multi-node parallelization to reduce the latency to generate a reconstructed sky image.

A. Sequential algorithm

The DDFacet imaging algorithm is described by Alg 1. For simplicity, the visibilities of a MS file are represented by $\mathbf{v}_{MS_j}$. The imager is an iterative algorithm going to K major cycles, using J MS files to reconstruct a hypercube $\widehat{\boldsymbol{x}}$ divided into I facets.

B. Multi-core parallelization

In order to facilitate the implementation of the distributed memory parallelization, a new implementation of the multi-core parallelization has been made using Concurrent-Futures[3]. This new implementation, following the same parallelization paradigm as the native version [4], is illustrated in Fig 1.

In the case of DDFacet, the image is divided into facets whose calculations are independent of each other. Thus each loop implying facets of Alg 1 is parallelized. The parallelization framework is based on an asynchronous behavior between the computational and I/O phases. 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 is dedicated for the computational jobs like *gridding*, *degridding*, and *FFT* for each facet φ, and an I/O queue for the I/O relative jobs.

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

Algorithm 1: DDFacet sequential imaging pipeline.

Data: ($\widehat{\boldsymbol{x}}$, **v**, I, J, K, PSF)
Initialization(**v**);
for *k in K_MajorCycles* **do**
  for *j in J_MS* **do**
    for *i in I_Facets* **do**
      $\widehat{\boldsymbol{g}}_{\varphi_i,MS_j} = FFT(\widehat{\boldsymbol{x}}, i, j)$;
      $\widehat{\mathbf{v}}_{\varphi_i,MS_j} = Degrid(\widehat{\boldsymbol{g}}_{\varphi_i,MS_j}, i, j)$;
    end
    $\delta\mathbf{v}_{MS_j} = \widehat{\mathbf{v}}_{MS_j} - \mathbf{v}_{MS_j}$;
    for *i in I_Facets* **do**
      $\boldsymbol{g}_{\varphi_i,MS_j} = Grid(\delta\mathbf{v}_{MS_j}, i, j)$;
    end
  end
  for *i in I_Facets* **do**
    $\delta\boldsymbol{y} += FFT\_inv(\boldsymbol{g}_{\varphi_i}, i, j)$;
  end
  $\widehat{\boldsymbol{x}} = Deconvolution(\delta\boldsymbol{y}, PSF)$;
end

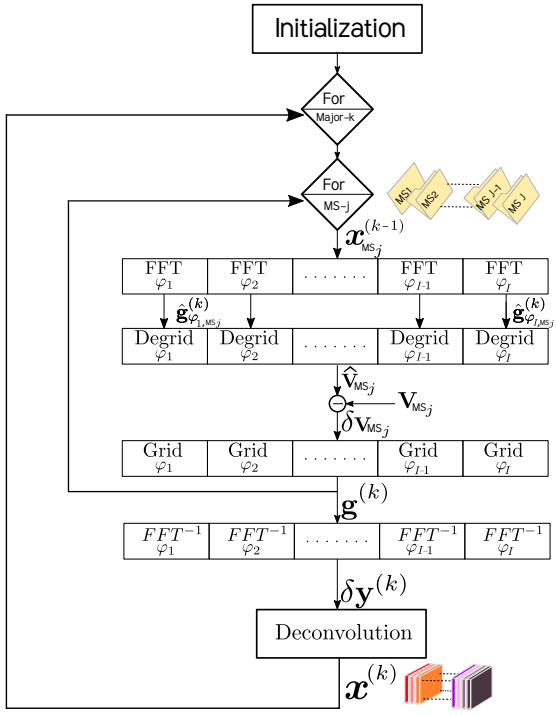

Fig. 1. Multi-core parallelization

C. Multi-node parallelization with data distribution

The naive multi-node parallelization paradigm used to increase the image production rate is to distribute observation on each node. However, this method, illustrated by Fig 2, only increases the throughput of image production. Indeed, in the

case of a large dataset, each image can take several hours/days to produce. Hence the need to reduce the latency of an image.

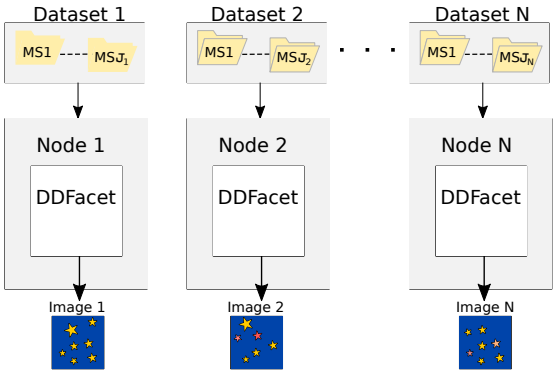

Fig. 2. Existing DDFacet multi-node parallelization. Each node generates an independent image using different dataset of a specific observation.

Our approach, illustrated by Fig 3, distributes the computation of a dataset composed of several MS files on several nodes to generate the corresponding image. The dataset is equally distributed over a number N of nodes to balance the computational load. The nodes gather the necessary metadata after an initialization phase specific to each sub-dataset such as BDA computation and weight computation. After that, the computational core of each node is identical, using the multi-core parallelization described in Sec III-B. The FFT, degridding, and gridding operators will be applied depending on the dataset. After the gridding of all MS for each node, the subgrids $\boldsymbol{g}_{1..N}^{(k)}$ are transferred and reduced to the master node, also called rank0, such as

$$\boldsymbol{g}^{(k)} = \sum_{i=1}^{N} \boldsymbol{g}_i^{(k)} \tag{3}$$

Only the master node is in charge of moving into the spatial domain and applying the deconvolution algorithm to the residual image $\delta \boldsymbol{y}$. The resulting hypercube $\boldsymbol{x}^{(k)}$ is then broadcast to all nodes, such that each node receives only the slices whose frequency band is associated with its subset in order to reduce IO communication.

The software implementation has been done in python using the Concurrent Futures interface of mpi4py[4] based on MPI [23]. The master MPI process is in charge of spawning MPI processes on one or several nodes.

Thus, this multi-node implementation makes two levels of parallelism of DDFacet. The first native one is based on facets parallelization and shared memory. The second one is based on Measurement Set parallelization, which allows for a distributed memory system. Moreover, this implementation allows better use of the computational resources within a node. Indeed, when the processor has a large number of cores or when we use few facets to reconstruct an image, such as

$$\frac{I\_facets}{nb\_cores} < 1, \tag{4}$$

[4]https://mpi4py.readthedocs.io/en/stable/mpi4py.futures.html

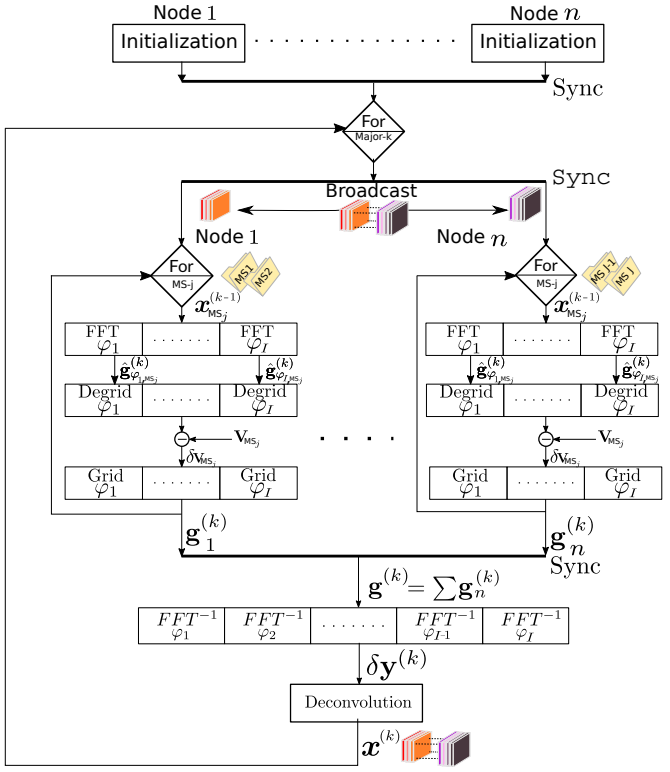

Fig. 3. Multi-node and multi-node parallelization based on independent MS files using distributed memory system.

it results in an under-utilization of the computational resources. Therefore, we can take advantage of this two-level parallelization by spawning several MPI processes per compute node. In this case, each MPI processes share the computation resources equally and is optimal when

$$\frac{I\_facets}{(\frac{nb\_cores}{nb\_mpi})} \geq 1. \tag{5}$$

IV. Experiments

This experiment aims to show the interest in using a distributed version of DDFacet to reduce the latency to obtain a result. A data set from the LOFAR radiotelescope artificially inflated to obtain 48 MS files for a total of 1.1TB of data is used. The experiments were performed on a BullSequana X400 system based on AMD EPYC 7742 64-Core processor. Each node, a dual-socket, is composed of 256 logical cores.

The purpose of this test case is to reconstruct a 10.000×10.000 pixels image after 20000 iterations of MSMF CLEAN, i.e. three major cycles, and the result is shown by Fig 4. The image is decomposed into $11 \times 11 = 121$ facets. The intra-node multi-core parallelization thus allows using 121 cores of our dual-socket processor. It results in an under-utilization of this processor. Our experiment will use 1, 2, and 4 MPI processes per node to maximize the computational cores' use and study each case's scalability and memory impact. Thus, in the case of one MPI process per node, 256 logical cores are allocated for the MPI process. For two MPI processes per

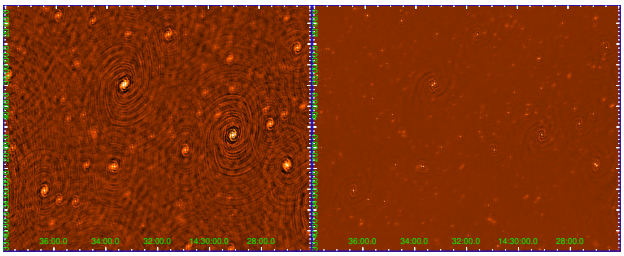

Fig. 4. Left : Dirty Image. Right : Restored image after 20.000 iterations of MSMF Clean.

node, 128 cores are allocated per MPI process, and in the case of four MPI processes per node, 64 logical cores are allocated per MPI process.

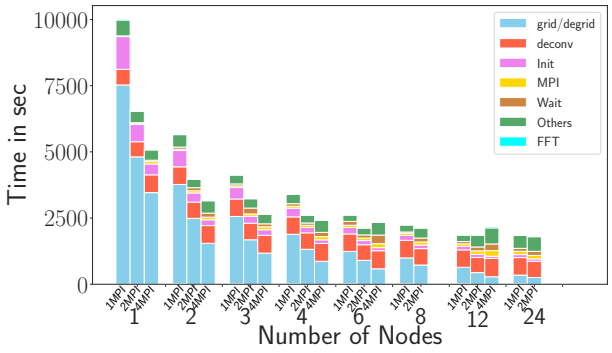

Fig. 5. Profiling of the master MPI process for a complete execution regarding the number of MPI processes per node (1, 2, and 4), and the number of nodes used for the parallel execution.

The total execution time and the profiling of the different steps of the algorithm are illustrated by Fig 5. For the case of 1 MPI process on one node, the gridding and degridding steps represent 73%, which confirms the result of [4]. This result highlights the need to distribute this core computation to reduce the total execution time. As expected, the total execution time tends to decrease with distribution on several nodes, demonstrating the interest of this implementation to reduce the latency. In this profiling, we find two types of computational blocks. Blocks with variable computational cost, such as gridding/degridding and initialization, whose computational cost decreases with the distribution on several nodes, following the parallelization scheme presented in Fig 3. And blocks with a fixed computational cost, such as Deconvolution, or Others, which gather all non-profiled timing, including some I/O and saving times. Additionally, the MPI communication time remains very low but increases with the number of MPI processes. The MPI communication waiting time varies with the balanced distribution of MS files among MPI processes.

The scalability factor of the distributed part on several MPI processes, i.e. the gridding, degridding, and FFT steps depending on the number of nodes used, is defined as

$$\alpha(p,n) = \frac{t_{p,1}}{t_{p,n}}, \tag{6}$$

where $t_{p,n}$ is the time of the master for p MPI processes per node distributed on n nodes. The scalability curves for 1 MPI process per node and 4 MPI processes per nodes are shown by Fig 6 and Fig 7. In both cases, the experimental scalability curve follows the theoretical one, showing the good distribution of the computation on several nodes.

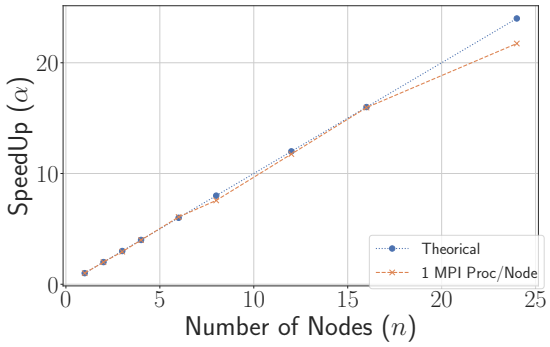

Fig. 6. Scaling of the gridding/degridding using one MPI process per node.

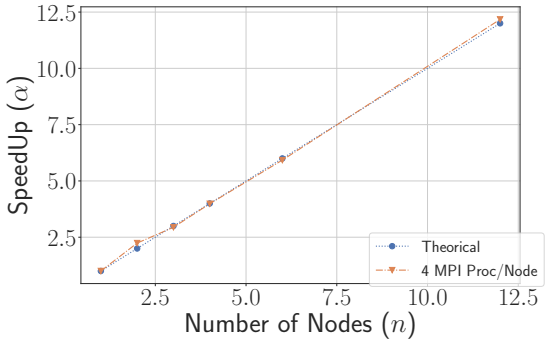

Fig. 7. Scaling of the gridding/degridding using four MPI processes per node.

Fig 8 shows the speedup of the total execution time regarding the execution time on one node with 1 MPI process, such as the speedup is

$$\beta(p,n) = \frac{t_{1,1}}{t_{p,n}}. \tag{7}$$

The speedup reaches an asymptote at 5.3 with a peak for $\beta(1,16) = 5.7$. Moreover, on a lower number of nodes, the higher the number of MPI processes, the higher the speedup. For instance, on one node $\beta(4,1)$ reaches 2.01.

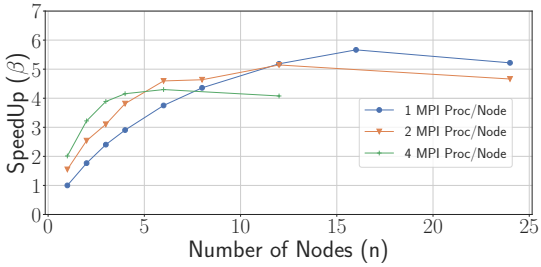

Fig. 8. Total execution time SpeedUp regarding the number of nodes used.

Fig 9 shows the gridding speedup and the total execution time for each node between 4 MPI processes and 1 MPI process, such as the speedup is defined as

$$\gamma(n) = \frac{t_{1,n}}{t_{4,n}}. \quad (8)$$

The gridding speedup remains constant with a factor γ around 2, which demonstrates an improvement of the multi-core parallelization. However, the total execution time speedup decreases. We can therefore conclude that the main limiting factor for the reduction of the execution time is the fixed cost times.

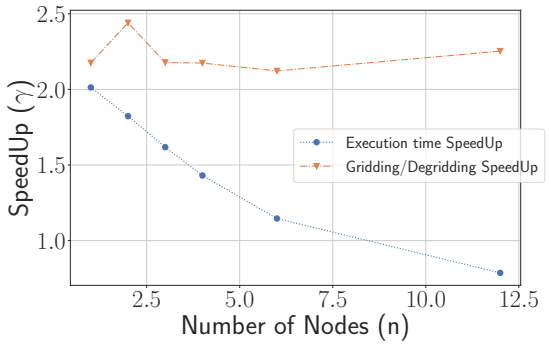

Fig. 9. Speedup of gridding/degridding and total execution time between 4 MPI processes and 1 MPI process regarding the number of nodes.

Finally, We can also observe an increase in memory consumption during gridding and degridding, as indicated by the Tab I. This increase is expected because we load in memory the data necessary for the processing of each MPI process.

	1 MPI/Node	2 MPI/Node	4 MPI/Node
Mem(GB)	27.4	34.1	68.1

TABLE I
MAXIMUM MEMORY CONSUMPTION DURING GRIDDING/DEGRIDDING.

V. CONCLUSION AND FUTURE WORK

This paper presents the parallelization of DDFacet, a wide-band wide-field spectral deconvolution framework on a distributed memory HPC system.

We have presented a two-level parallelization paradigm with facet-based parallelization for the multi-core shared memory aspect and MS-based parallelization for the multi-node distributed-memory aspect. This implementation allowed for distributing the most expensive computational blocks, such as gridding and degridding, ensuring perfect scalability. Using a LOFAR dataset, the total time to generate a reconstructed image was reduced to a factor up to 5.7. Finally, we showed that this implementation could also improve the multi-core parallelization depending on the number of facets and the processor used. The perspective for the future is to continue working on ways to reduce the execution time. The first track is to apply our multi-node framework to distribute the deconvolution computation on several nodes. Moreover, to optimize the computational load between several nodes because of the compression applied by the BDA algorithm. Finally, to implement optimization for gridding and degridding on accelerator card for DDFacet.

ACKNOWLEDGMENT

This work was supported by grants from Région Ile-de-France and DARK-ERA (ANR-20-CE46-0001-01).

REFERENCES

[1] F. Acero and al., *French SKA White Book - The French community towards the Square Kilometer 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*, 2018.
[3] W. Brouw, "Aperture synthesis," *Methods in Computational Physics*, vol. 14, no. 10, pp. 684–692, 1975.
[4] C. Tasse and al., "Faceting for direction-dependent spectral deconvolution," *Astronomy & Astrophysics*, vol. 611, p. A87, 2018.
[5] A. H. Barnett and al., "A parallel non-uniform fast Fourier transform library based on an "exponential of semicircle" kernel," *SIAM Journal on Scientific Computing*, Apr. 2019.
[6] J. W. Romein, "An efficient work-distribution strategy for gridding radio-telescope data on GPUs," in *Proceedings of the 26th ACM international conference on Supercomputing - ICS '12*. ACM Press, 2012, p. 321.
[7] B. Merry, "Faster GPU-based convolutional gridding via thread coarsening," *Astronomy and Computing*, vol. 16, pp. 140–145, 2016.
[8] M. B., "Approximating W projection as a separable kernel," *Mon. Not. R. Astron. Soc.*, vol. 456, no. 2, pp. 1761–1766, Feb. 2016.
[9] S. van der Tol and al., "Image domain gridding: a fast method for convolutional resampling of visibilities," *Astronomy & Astrophysics*, vol. 616, p. A27, 2018.
[10] B. Veenboer and al., "Image-domain gridding on graphics processors," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 545–554.
[11] B. Veenboer and J. W. Romein, *Radio-Astronomical Imaging: FPGAs vs GPUs*. Cham: Springer International Publishing, 2019, vol. 11725.
[12] S. Jaeger, "The Common Astronomy Software Application (CASA)," *ADASS ASP Conference Series*, vol. 394, p. 4, 2008.
[13] A. R. Offringa and al., "WSClean: an implementation of a fast, generic wide-field imager for radio astronomy," *MNRAS*, vol. 444, no. 1, pp. 606–619, 2014.
[14] W. Gropp and al., *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.
[15] C. Wu and al., "DALiuGE: A graph execution framework for harnessing the astronomical data deluge," *Astronomy and computing*, 2017.
[16] R. Wang, R. Tobar, and M. Dolensky, "Processing full-scale square kilometre array data on the summit supercomputer," p. 12, 2020.
[17] S. J. Perkins and al., "A dask distributed radio astronomy reduction framework," in *2021 XXXIVth URSI GASS*. IEEE, 2021, pp. 1–3.
[18] V. Parekh and al., "Third-Generation Calibrations for MeerKAT Observation," *Galaxies*, vol. 9, no. 4, p. 90, Nov. 2021.
[19] O. M. Smirnov, "Revisiting the radio interferometer measurement equation. i. a full-sky jones formalism," *A&A*, vol. 527, p. A106, 2011.
[20] M. P. van Haarlem and al., "LOFAR: The LOw-frequency ARray," *Astronomy & Astrophysics*, vol. 556, p. A2, 2013.
[21] M. T. Atemkeng and al., "Using baseline-dependent window functions for data compression and field-of-interest shaping in radio interferometry," *MNRAS*, vol. 462, no. 3, pp. 2542–2558, 2016.
[22] S. J. Wijnholds, A. G. Willis, and S. Salvini, "Baseline-dependent averaging in radio interferometry," *Monthly Notices of the Royal Astronomical Society*, vol. 476, no. 2, pp. 2029–2039, 2018.
[23] J. Gonzalez and al., "Python Code Parallelization, Challenges and Alternatives," *ADASS XXVI*, vol. 521, p. 4, 2019.