Fast Grid to Grid Interpolation for Radio Interferometric Imaging

Nicolas Monnier[a], François Orieux[a], Nicolas Gac[a], Cyril Tasse[b], Erwan Raffin[c], David Guibert[c]

[a]*Laboratoire des Signaux et Systèmes, Univ. Paris-Saclay, CNRS, CentraleSupélec, 91 190 Gif-sur-Yvette, France*
[b]*GEPI, Observatoire de Paris, CNRS, Univ. Paris Diderot, 92 190 Meudon, France*
[c]*CEPP - Center for Excellence in Performance Programming, Atos Bull, 35 700 Rennes, France*

Abstract

Backward and forward interpolations on a Fourier grid are computationally expensive operations for radio interferometric imaging algorithms. By merging these operations, we propose the Grid to Grid (G2G) method which aims to reduce the computational cost and memory footprint. We have also shown that the oversampling factor used for the convolution function in the G2G method strongly impacted the accuracy and computational speed. Acceleration on graphics processing units (GPU), well suited for this embarrassingly parallel algorithm, has been studied mainly for backward operation. Thus, we propose a GPU and CPU implementation of the G2G method on Nvidia A100 and Intel Ice Lake processors. Experiments have shown a GPU performance improvement with a Fourier-point throughput better than up to 37% regarding standard gridder and degridder.

Keywords: Radioastronomy, Inverse Problem, Interpolation, GPU, Gridding

Contents

1. Introduction

In recent years, large projects involving astronomers, computer scientists, and engineers have developed new generations of radio telescopes to improve antenna sensitivity, resolution, and data quality. These improvements have led to a significant increase in data generation, involving a higher volume of data to process. Because the volume of data is directly related to the number of antennas, telescopes with many antennas will have more data to process.

Imaging is a critical phase of the data processing pipelines. It consists of adding the data generated by the telescope, called visibilities, on a Fourier-transformed grid to create an image of the sky. This step, called gridding, and its adjoint, the degridding, are critical steps in the imaging phase because they are computationally very expensive. Existing methods for sky reconstruction have all the common points of being iterative and require to grid and degrid the visibilities at each iteration. Even if algorithms like Cotton-Schwab CLEAN (Schwab, 1984) limit the number of these operations, the computational cost remains extremely high.

These gridding and degridding operators are good candidates to be parallelized on many-core accelerators like the GPU. Therefore, a lot of work has been done, mainly on the gridding operator, to decrease the main memory bandwidth and not rely on memory caches. Several works, such as (Merry, 2016), (Romein, 2012a), and (Muscat, 2014), used the advantage of the proximity of the coordinates between two data samples on the grid to reduce the memory accesses. However, most of the existing methods only focus on the gridding step.

This paper presents a method to reduce the algorithmic complexity of gridding and degridding operators by merging them to reduce the amount of data used. This method, based on FHD (Sullivan and al., 2012), also has the advantage of reducing the global memory footprint of these operations. Finally, this paper presents a GPU implementation of this new operator, taking into account the extension of this work with the *w*-correction,

as well as the addition of a degridding operator implementation.

This paper is structured as follows. Section 2 presents the basics of imaging in radio astronomy. Section 3 explains in a theoretical way the Grid to Grid (G2G) method. Section 4 is dedicated to the parallelization of the method on GPU, compared to the state-of-the-art on the same subject. Section 5 shows the performances of this method on GPU. Finally, we conclude this work by discussing future works.

2. Radio Interferometric Imaging

2.1. Forward problem and synthesis imaging

A radio interferometer array is a network of antennas that generates measures of the radio emission of the observed sky $\boldsymbol{x}$. Each antenna's pair is defined with a baseline $\boldsymbol{b} = (u, v, w)$ where u, v, and w are the coordinates in unit of wavelength λ. The measurement of one antenna pair of baseline $\boldsymbol{b}$, or so-called *visibility*, is defined as

$$\mathrm{v}(u, v, w) = \iint \frac{x(l, m, n)}{n} e^{-2i\pi(ul+vm+w(n-1))} \,\mathrm{d}l\,\mathrm{d}m \tag{1}$$

where x is the sky brightness distribution using a coordinate system (l, m, and $n = \sqrt{1 - l^2 - m^2}$) that indicates an angular position.

As the earth's rotation causes the baseline to change over time, the position of generated visibilities in the UVW space also changes during the time. Thus, the total number of visibilities generated during an observation is

$$M = \frac{N_{bl} \times N_{ch} \times N_{pol} \times T}{\Delta t}. \tag{2}$$

where N_{bl} is the number of baselines, N_{ch} the number of frequencies, N_{pol} the number of polarization, T the duration of the observation, and Δt the integration time for a sample. As current radio telescopes, such as SKA, have many antennas (197 for SKA-mid) and have to produce large images up to 30.000×30.000 pixels, the radio interferometric imaging problem is considered a large-scale problem.

If the array of antennas is coplanar (all visibilities fall on an arbitraly plane in the *uvw* space) and the FOV is small ($n \approx 1$), each visibility $\mathrm{v}(u, v)$ is the 2D spatial Fourier transform of the sky distribution at frequency (u, v). This result is known as the van Cittert-Zernike theorem (Thompson et al., 2001), and Eq. (1) becomes

$$\mathrm{v}(u, v) = \iint x(l, m) e^{-2i\pi(ul+vm)} \,\mathrm{d}l\,\mathrm{d}m. \tag{3}$$

Since the array cannot cover the full (u, v) plane, radio interferometric imaging aims to recover the sky from incomplete visibility measurements, leading to an ill-posed linear inverse problem.

Imaging synthesis computes the so-called dirty image, which is the inverse Fourier Transform of the measured visibilities such as

$$\boldsymbol{y}(l, m) = \sum_{i=1}^{M} \mathbf{v}_i e^{2i\pi(u_i l + v_i m)}. \tag{4}$$

The most straightforward approach to do it, using a Direct Fourier Transform (DFT), is computationally too expensive. The computational cost of the DFT for M visibilities in $N_p \times N_p$ pixels image is

$$C_{DFT} = O(N_p^2 M) \tag{5}$$

On the other hand, the FFT needs the visibilities to be uniformly sampled, which is not. The most common solution is to interpolate visibilities with non-uniform *uv*-coverage into a uniform one. This procedure is called gridding. Its computational cost for coplanar arrays and the FFT is

$$C_{grid} = O(N_p^2 log_2 N_p + C_{supp}^2 M) \tag{6}$$

where C_{supp}^2 is the size of the convolution function in 2D.

After discretization, the dirty image can be represented by $N_p \times N_p = P$ pixels grid as $\boldsymbol{y} \in \mathbb{R}^P$, and the visibility measurement as a vector $\boldsymbol{v} \in \mathbb{C}^M$. Relation between $\boldsymbol{y}$ and $\boldsymbol{v}$ is described by

$$\boldsymbol{y} = \boldsymbol{F}^\dagger \boldsymbol{S}^\dagger \mathbf{v} \tag{7}$$

where $\boldsymbol{F}^\dagger \in \mathbb{C}^{P \times P}$ is the inverse Fourier transform matrix, $\boldsymbol{S}^\dagger \in \mathbb{C}^{P \times M}$ maps M visibilities to P Fourier coefficients and is the *gridding* operator. The discrete forward problem is the adjoint of eq. 7 such as

$$\mathbf{v} = \boldsymbol{S}\boldsymbol{F}\boldsymbol{x} + \boldsymbol{n} \tag{8}$$

where $\boldsymbol{F} \in \mathbb{C}^{P \times P}$ is the Fourier transform, $\boldsymbol{S} \in \mathbb{C}^{M \times P}$ is the *degridding* operator, adjoint of $\boldsymbol{S}^\dagger$, and $\boldsymbol{n} \in \mathbb{C}^M$ is an i.i.d. Gaussian noise.

2.2. Gridding operator

To use the FFT algorithm, the gridding (and similarly the degridding) operator aims to interpolate data with non-uniform coordinates on a uniform grid. This kind of problem is common in different fields of study. This method is, for example, known as the Non-Uniform Fast Fourier Transform (NUFFT) in the medical domain, such as MRI (Fessler and Sutton, 2003),(Barnett et al., 2019),(Zhili Yang and Jacob, 2009).

In such interpolation, the true position $\boldsymbol{b}_i = (u_i, v_i)$ of the visibilities are approximated to the nearest neighbor on an oversampled grid of resolution (Δ_u, Δ_v). If K is the oversampling factor, the FFT steps of the grid $\boldsymbol{g}$ are $(K\Delta_u, K\Delta_v)$. The approximated position of $\boldsymbol{b}_i$ becomes

$$\widetilde{\boldsymbol{b}}_i = (p_i \Delta_u, q_i \Delta_v). \tag{9}$$

In that case, using the gridding operator $\boldsymbol{S}^\dagger$ to compute the coordinate $\boldsymbol{b}_k = (p_k K\Delta_u, q_k K\Delta_v)$ of the grid $\boldsymbol{g}$ is equivalent to

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{b}_k) &= \sum_{i=1}^{M} C^\dagger(\boldsymbol{b}_k - \widetilde{\boldsymbol{b}}_i)\mathrm{v}_i \\ &= \sum_{i=1}^{M} C^\dagger(p_k K\Delta_u - p_i\Delta_u, q_l K\Delta_v - q_i\Delta_v)\mathrm{v}_i \end{aligned} \tag{10}$$

where $C^\dagger$ is a discrete precomputed 2D kernel of size $C_{supp} \times C_{supp}$ in FFT step which contains $K^2 \times C_{supp} \times C_{supp}$ values. The choice and the size of this interpolator to avoid aliasing effects and maximize accuracy is a well-known subject in the literature, see for instance (Fessler, 2007),(Beatty et al., 2005),(Thévenaz et al., 2000). In our case, we used the Kaiser-Bessel function with a support $C_{supp} = 7$.

2.3. Sky estimation

The sky $\boldsymbol{x}$ reconstruction for incomplete data $\mathbf{v}$ is an ill-posed inverse problem. Therefore, interferometric imaging is usually defined as

$$\widehat{\boldsymbol{x}} = \arg\min_x \frac{1}{2}\|\mathbf{v} - \boldsymbol{S}\boldsymbol{F}\boldsymbol{x}\|^2 + R(\boldsymbol{x}) \tag{11}$$

where $\|\mathbf{v} - \boldsymbol{S}\boldsymbol{F}\boldsymbol{x}\|^2$ is the data fidelity term and R a regularizer (Giovannelli and Coulais, 2005; Thiébaut and Young, 2017; Bester et al., 2021). Algorithms used to solve this optimization problem require computing many gradients of the criterion during their iterative processes. The computing cost of a gradient of the data fidelity term

$$\nabla J(\boldsymbol{x})_{STD} = \boldsymbol{F}^\dagger \boldsymbol{S}^\dagger (\boldsymbol{v} - \boldsymbol{S}\boldsymbol{F}\boldsymbol{x}) \tag{12}$$

can be very high since it requires the computation of the gridding and degridding. The gradient computation $\nabla J(\boldsymbol{x})_{STD}$ corresponds to the residual image computation $\delta\boldsymbol{y}$ of the historical Cotton-Schwab CLEAN algorithm (Schwab, 1984). We proposed in this paper a new formulation to reduce the computing cost of the gradient evaluation without additional approximation. The gradient computation corresponds to the so-called "major loop". The deconvolution, called "minor loop," used in the overall reconstruction methods like (Schwab, 1984) Cotton-Schwab CLEAN is not studied here.

3. Grid to Grid interpolation

In this section, we propose a decomposition of the gridding and degridding operators in order to facilitate their fusion into the Grid to Grid operator. We also present a generalization and the particularities of the G2G method.

3.1. Gridding and degridding decomposition

A way to implement the gridding operator $\boldsymbol{S}^\dagger$ is to decompose it into three sub-operators : accumulation, convolution, and subsampling.

1. **Accumulation $\boldsymbol{A}^\dagger$.**
   Let's define $\boldsymbol{g'}$ a vectorized oversampled 2D grid of size $KN_p \times KN_p = K^2P$ with thin pixel resolution (Δ_u, Δ_v). The oversampled pixel value associated with the coordinate $\boldsymbol{b}_k$ on the thin grid is

   $$\boldsymbol{g'}(\boldsymbol{b}_k) = \begin{cases} \sum_i \mathrm{v}_i & \text{if } \widetilde{\boldsymbol{b}}_i = \boldsymbol{b}_k, \\ 0 & \text{otherwise.} \end{cases}$$

   Thus, $\boldsymbol{g'}$ is a grid of accumulated visibilities that are approximated at the same position on the thin grid. We therefore define an accumulation operator $\boldsymbol{A}^\dagger \in \{0,1\}^{K^2P \times M}$ such as $\boldsymbol{g'} = \boldsymbol{A}^\dagger \mathbf{v}$. $\boldsymbol{A}^\dagger$ is a sparse operator with one '1' per column, '0' otherwise.
2. **Convolution $\boldsymbol{C}^\dagger$.**
   Considering a 2D grid $\widetilde{\boldsymbol{g}}$ of size $KN_p \times KN_p$ with thin step resolution (Δ_u, Δ_v) (same size as $\boldsymbol{g'}$). This step is a two-dimensional discrete convolution between the thin grid $\boldsymbol{g'}$ and the convolution function $C^\dagger$, such as the value associated to the coordinate $\boldsymbol{b}_k$ is

   $$\widetilde{\boldsymbol{g}}(\boldsymbol{b}_k) = \sum_i \sum_j C^\dagger\Big((p_k - p_i)\Delta_u, (q_k - q_j)\Delta_v\Big)\boldsymbol{g'}(\boldsymbol{b}_{ij}).$$

   Thus, we define the discrete convolution operator $\boldsymbol{C}^\dagger \in \mathbb{C}^{K^2P \times K^2P}$ such as $\widetilde{\boldsymbol{g}} = \boldsymbol{C}^\dagger \boldsymbol{g'}$.
3. **Sub-sampling $\boldsymbol{O}^\dagger$.**
   The FFT grid is coarser by a factor K regarding the oversampled grid. We define the subsampling operator $\boldsymbol{O}^\dagger \in \{0,1\}^{P \times K^2P}$ that reduces a grid of size K^2P into a grid of size P. Thus, the FFT grid $\boldsymbol{g}$ is $\boldsymbol{g} = \boldsymbol{O}^\dagger \widetilde{\boldsymbol{g}}$. The subsampling operator has one '1' per line, '0' otherwise.

Combining these three sub-operators, we can define the FFT grid as $\boldsymbol{g} = \boldsymbol{O}^\dagger \boldsymbol{C}^\dagger \boldsymbol{A}^\dagger \mathbf{v}$. Thus, the gridding operator can be written as

$$\boldsymbol{S}^\dagger = \boldsymbol{O}^\dagger \boldsymbol{C}^\dagger \boldsymbol{A}^\dagger. \tag{13}$$

This decomposition of gridding into sub-operators is illustrated in Fig. 1. In practice, these operators are not instantiated as a matrix but are implicit, being applied by dedicated code.

The degridding operator $\boldsymbol{S}$ is the adjoint of the gridding $\boldsymbol{S}^\dagger$. Thus, from Eq. (13), we can define the degridding decomposition as

$$\boldsymbol{S} = \boldsymbol{A}\boldsymbol{C}\boldsymbol{O}, \tag{14}$$

where $\boldsymbol{O} \in \{0,1\}^{K^2P \times P}$ is an oversampling operator filling the oversampled grid with zeros, $\boldsymbol{C} \in \mathbb{C}^{K^2P \times K^2P}$ is a 2D discrete convolution where the convolution function C is the flipped conjugate of $C^\dagger$, and $\boldsymbol{A} \in \{0,1\}^{M \times K^2P}$ is a sampling mapping operator that builds M visibilities at their approximate coordinates from K^2P coefficients. If $\widehat{\boldsymbol{g}}$ is the Fourier grid to be degridded, the visibility vector $\mathbf{v}$ is $\mathbf{v} = \boldsymbol{A}\boldsymbol{C}\boldsymbol{O}\widehat{\boldsymbol{g}}$. This decomposition is illustrated in Fig. 2.

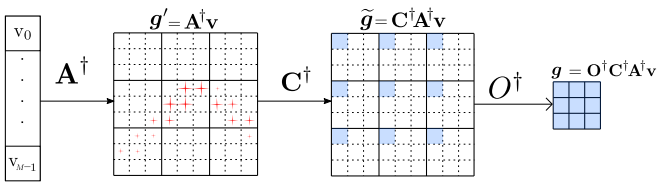

Figure 1: Gridding decomposition. First, accumulation of visibilities on a 2D thin grid of size $KN \times KM$, then a 2D discrete convolution with a kernel $C^\dagger$, and finally, a K subsampling.

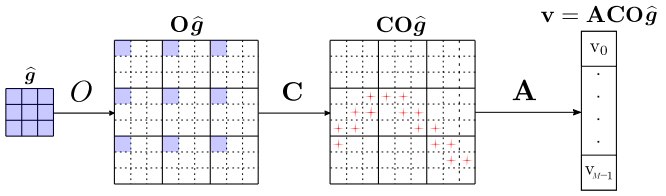

Figure 2: Degridding decomposition. First, oversampling of a map $\boldsymbol{g}$ with a factor K, then a 2D discrete convolution with a kernel C, and finally, a mapping operator $\boldsymbol{A}$ maps the Fourier coefficients to M visiblities.

3.2. Gradient computation with G2G

The standard method to compute the gradient operator described by Eq. (12) can be developed such as it becomes

$$\begin{aligned}\nabla J(\boldsymbol{x}) &= \boldsymbol{F}^\dagger \boldsymbol{S}^\dagger \boldsymbol{v} - \boldsymbol{F}^\dagger \boldsymbol{S}^\dagger \boldsymbol{S}\boldsymbol{F}\boldsymbol{x} \\ &= \boldsymbol{y} - \boldsymbol{F}^\dagger \boldsymbol{S}^\dagger \boldsymbol{S}\boldsymbol{F}\boldsymbol{x}.\end{aligned} \tag{15}$$

The Fast Holographic Deconvolution (FHD) (Sullivan and al., 2012) algorithm used this decomposition. We can note that $\boldsymbol{F}\boldsymbol{S}^\dagger\boldsymbol{v}$ is the dirty image $\boldsymbol{y}$, which is computed only once. In this method, they build the *holographic* map $\boldsymbol{H} \in \mathbb{C}^{P\times P}$ such as $\boldsymbol{H} = \boldsymbol{S}^\dagger\boldsymbol{S}$. This method has the advantage of being fast because a simple matrix product is needed to compute the gradient. However, the computation time and the memory footprint required to compute $\boldsymbol{H}$ are prohibitive.

In our case, we use the same development of Eq. 15, as well as the gridding and degridding decomposition. Thus, the succession of degridding and gridding operators can be rewritten as

$$\boldsymbol{S}^\dagger\boldsymbol{S} = \boldsymbol{O}^\dagger\boldsymbol{C}^\dagger\boldsymbol{A}^\dagger\boldsymbol{A}\boldsymbol{C}\boldsymbol{O}. \tag{16}$$

from this equation, we built $\boldsymbol{A}^* = \boldsymbol{A}^\dagger\boldsymbol{A}$, and $\boldsymbol{A}^* \in \mathbb{N}^{K^2P\times K^2P}$ a diagonal matrix, such as $\boldsymbol{a} = \mathrm{diag}(\boldsymbol{A}^*)$ is a vector of size K^2P. The value of each element a_i of $\boldsymbol{a}$ corresponds to the number of visibilities whose positions have been approximated on the corresponding oversampled pixel. Therefore, it is sufficient to do a term-wise product between $\widetilde{\boldsymbol{g}}$ and $\boldsymbol{a}$ instead of going through the visibility vector $\mathbf{v}$. This method, illustrated by Fig. 3, will be referred as the Grid to Grid method (G2G), and the gradient computation becomes

$$\nabla J(\boldsymbol{x})_{G2G} = \boldsymbol{y} - \boldsymbol{F}^\dagger G2G(\boldsymbol{F}\boldsymbol{x}), \tag{17}$$

where $G2G(\bullet) = \boldsymbol{O}^\dagger\boldsymbol{C}^\dagger\boldsymbol{A}^*\boldsymbol{C}\boldsymbol{O}\bullet$.

In a typical continuum imaging application, the STD method uses a different number of subbands for gridding and degridding. Typically, gridding requires fewer subbands than degridding. However, with the G2G method, the number of subbands must be identical and set as the greater of the two.

3.3. G2G compression ratio

We define M' as the number of non-zero elements in $\boldsymbol{a}$. With M the number of raw data, practical cases show that $M > M'$ or $M \gg M'$ depending on the oversampling factor K. Consequently, the G2G method avoids processing the vector of M raw visibilities but instead processes a smaller vector of size M'. Moreover, unlike FHD (Sullivan and al., 2012) who builds and stores $\boldsymbol{S}^\dagger\boldsymbol{S} \in \mathbb{C}^{P\times P}$ as a matrix, we only need to build and store the sparse vector $\boldsymbol{a}$. Thus, from Eq. (10) and Eq. (16) the output grid $\boldsymbol{g}$ at coordinates $\boldsymbol{b}_k = (u_k, v_k)$ becomes

$$g(\boldsymbol{b}_k) = \sum_{i'=1}^{M'} C^\dagger\Big(\boldsymbol{b}_k - \boldsymbol{b}_{i'}\Big)a_{i'} \sum_{i,j} C\Big(\boldsymbol{b}_{i'} - \boldsymbol{b}_{ij}\Big)\widetilde{g}(\boldsymbol{b}_{ij}). \tag{18}$$

For the same oversampling factor, the gradient computed by the G2G method Eq. (15) and the STD method Eq. (12) is the same. Furthermore, the computing cost of the gradient using the G2G method is given by

$$C_{G2G} = O(2M'C^2_{supp} + M' + 2P\log P + P) \tag{19}$$

while the computing cost of the standard approach is

$$C_{STD} = O(2MC^2_{supp} + 2P\log P + M). \tag{20}$$

Hence, for the same K, the presented method is more efficient without additional errors when

$$M' < \frac{2MC^2_{supp} + M - P}{2C^2_{supp} + 1} \tag{21}$$

With a data set for a standard sky observation (8h), empirical measurements show that $MC^2_{supp} >> P - M$. Thus, Eq. 21 can be simplified such that

$$M' \lesssim M. \tag{22}$$

In cases where $MC^2_{supp} >> P - M$ is false, we can no longer simplify, and Eq. 21 must be respected to ensure a decrease in the computational cost.

We define as the compression ratio, the ration between M' and the raw number of visibilities M, such as

$$c_\alpha = \frac{M'}{M} \times 100. \tag{23}$$

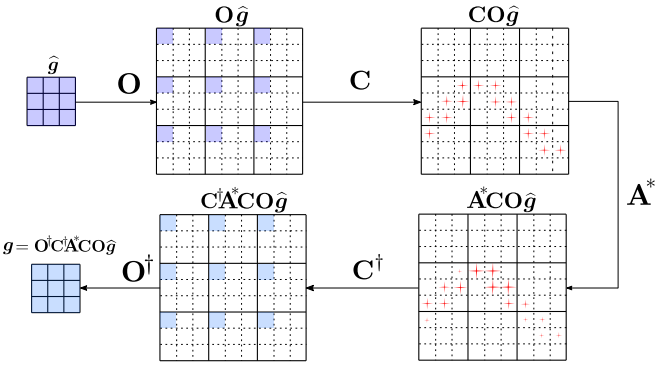

Figure 3: Degridding and gridding decomposition using the G2G method. An element-wise multiplication is done between the diagonal of the operator $\boldsymbol{A}^*$ and the oversampled grid.

Note that the compression provided by the G2G method can be seen as a generalization of the compression provided by the Baseline Dependent Averaging method (Wijnholds et al., 2018; Atemkeng et al., 2016). The BDA method averages visibilities over time as a function of baseline, whereas G2G averages visibilities over their position in the Fourier plane. Moreover, BDA's compression leads to an approximation that can be chosen for higher or weaker compression. With G2G, the compression comes directly from the model.

3.4. Extension for w-term

Taking into account the effects related to the non-coplanarity of the antenna array (visiblities fall on different planes in the *uvw* space), the visibility equation from Eq. 1 is

$$\mathrm{v}(u,v,w) = \iint \frac{x(l,m,n)}{n} e^{-2i\pi(ul+vm)} G(l,m,w)\,\mathrm{d}l\,\mathrm{d}m \quad (24)$$

with

$$G(l,m,w) = e^{-2i\pi\left(w(\sqrt{1-l^2-m^2}-1)\right)}. \quad (25)$$

(Cornwell et al., 2008) has shown that a reprojection is allowed to and from any position in the (u,v,w) space to and from $w = 0$ plane by convolution using a known kernel. This technique, called w-projection, uses the convolution theorem from the Eq. 24 such as

$$\mathbf{v}(u,v,w) = \mathbf{v}(u,v,w=0) * \tilde{G}(u,v,w), \quad (26)$$

where $\tilde{G}(u,v,w)$ is the Fourier transform of $G(l,m,w)$. This convolution can be done during the gridding, which makes the convolution function more complex. In order to avoid computing the convolution function on the fly for each w value, the closest w-layer is chosen. A vast number of convolution functions are precomputed and stored. Moreover, the support size of the kernel depends on the image size and the w value (Tasse et al., 2013). Thus, the computational cost of the convolution increases regarding this particular correction.

The G2G interpolation method has no significant changes by including the w correction. Indeed, using the decomposition used in Fig. 3, only the discrete convolution operators $\boldsymbol{C}^\dagger$ and $\boldsymbol{C}$ change depending on the convolution function, which is related to w. As illustrated in Fig. 4, a matrix $\boldsymbol{A}^*$ must be computed for each layer W, such that $\boldsymbol{A}^*_{w_i}$ matches all w values approximated at w_i.

As shown in Fig. 5, the values of w vary smoothly over time. The construction of the sparse diagonals of $\boldsymbol{A}^*_{w_i}$ is, therefore, not a problem.

3.5. Compression approximation

The plots of the variation of baseline length in the (u,v,w) space are curves. The projections of these curves in the (u,v) space become arcs. The u, v, and w coordinates are given from the cartesian coordinates system (X,Y,Z), a terrestrial reference frame that never change from a local observer on earth, of the baselines as

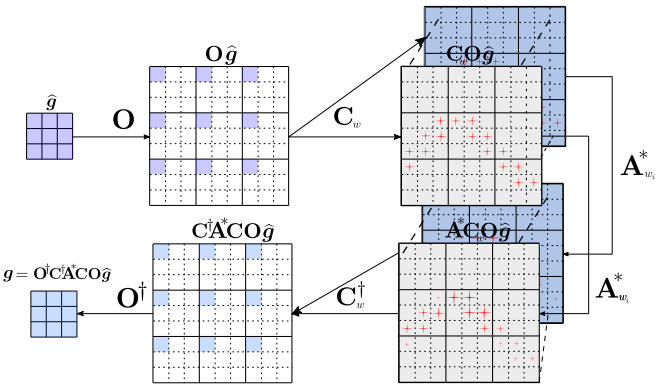

Figure 4: Extension of the grid to grid method including the w correction.

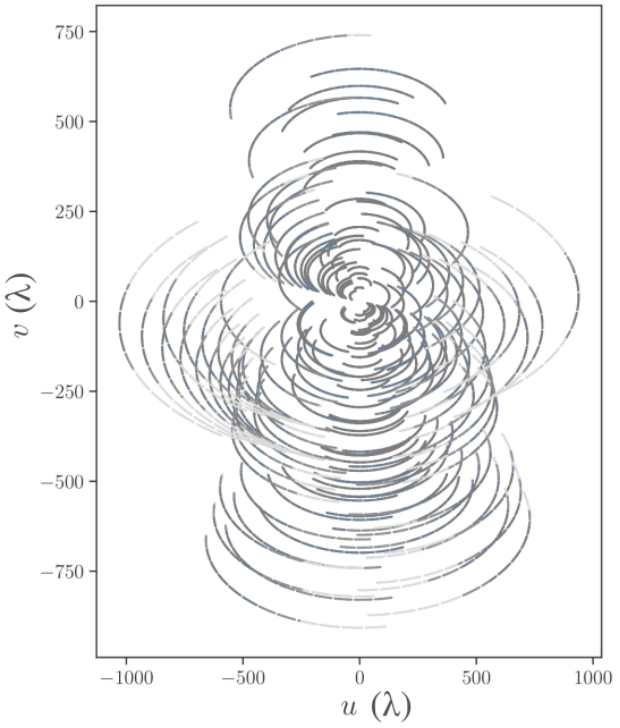

Figure 5: Variation of the w coordinate value in the uv coverage. Different shades of grey are used for a different set of w values. w varies slowly in time during the observation. A simulation of a VLA-D 8 hours observation pointed at RA=19 : 25 : 59, Dec= 21.06.26.

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} \sin H_0 & \cos H_0 & 0 \\ -\sin\delta_0 \cos H_0 & \sin\delta_0 \sin H_0 & \cos\delta_0 \\ \cos\delta_0 \cos H_0 & -\cos\delta_0 \sin H_0 & \sin\delta_0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (27)$$

where (H_0, δ_0) are the coordinates of the observation's phase center on the celestial sphere. It has been shown by (Thompson et al., 2001) that the track formed in time by the baseline projected in the (u, v) space is an ellipse. Thus, by transforming Eq. 27 and removing the time aspect, we obtain the following ellipse equation

$$u^2 + \left[\frac{v - \frac{Z}{\lambda}\cos\delta_0}{\sin\delta_0}\right]^2 = \left[\frac{X}{\lambda}\right]^2 + \left[\frac{Y}{\lambda}\right]^2. \quad (28)$$

The portion of the ellipse traced during the observation depends on the azimuth, the elevation and the latitude of the baseline, the declination of the source, and the hour angle covered during the observation. Fig. 6 shows the superposition of the *uv* track and the ellipse built with the same parameters. We can

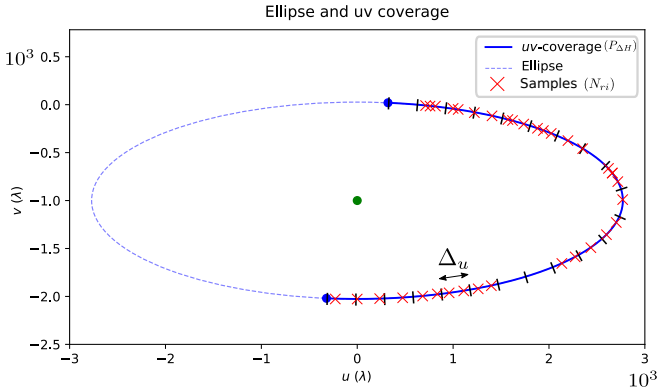

Figure 6: *uv*-coverage for a single baseline and the corresponding ellipse (dash-line).

estimate the ellipse's perimeter formed from Eq 28 by taking the semimajor axis $a = \sqrt{\frac{X^2}{\lambda} + \frac{Y^2}{\lambda}}$ and the semiminor axis $b = sin(\delta_0)\sqrt{\frac{X^2}{\lambda} + \frac{Y^2}{\lambda}}$. The different formulas of Ramanujan (Villarino, 2005) allow us to approximate the ellipse's perimeter P with very low errors depending on a and b. We can, therefore, estimate the theoretical compression factor $\widehat{c_\alpha}$ as a function of the ellipse's perimeter for an observation time $P_{\Delta H}$ and the resolution Δ_u that depends on the oversampling factor.

In the case of a single baseline i at a specific frequency, the idea is to divide the number of intervals in the portion of the ellipse's perimeter $\frac{P_{\Delta H,i}}{\Delta_u}$ by N_{ri}, the number of visibility acquired during the observation period for this specific baseline. As the number of intervals might be higher than the number of samples, meaning no compression, the compression number is limited by 1. Thus, the compression estimation is

$$\widehat{c_{\alpha,i}} = min(\frac{(\frac{P_{\Delta H,i}}{\Delta_u})}{N_{ri}}, 1) \times 100. \quad (29)$$

To estimate the compression for all baselines, take the average of all $\widehat{c_{\alpha,i}}$ such as

$$\widehat{c_\alpha} = mean(\widehat{c_{\alpha,i}}). \quad (30)$$

4. GPU Implementation

In this section, we will introduce in more detail the state of the art of different GPU implementations of gridding algorithms. We will then see the specificity of the implementation of the G2G method on GPU.

4.1. Related work

Generic image-processing convolution on GPU has been studied intensively for years and is even studied in many tutorials to learn programming (Cheng et al., 2014). However, in radio astronomy imaging, the gridding and degridding are rather a sampled convolution than an image convolution. As a result, memory access patterns are less predictable, and thus, creating an image is much more computationally expensive.

GPU gridding implementations started with the democratization of the GPU for scientific computing around the 2010s. (Edgar et al., 2010) have developed a GPU gridder, focused on NVidia platforms with CUDA language, for the Murchison Widefield Array (MWA) telescope. They recognized that accumulating visibilities convoluted on the grid was not thread safe. Moreover, adding the visibilities directly on the grid significantly increases the device-memory access cost. Their implementation linked a CUDA thread for each output grid point (a pixel). Then, each thread searches for all the visibilities which one has an impact on the corresponding grid point by checking them one by one. However, this approach was inefficient, and only 60 visibilities out of 130,000 were concerned. To overcome this problem, they sorted the visibilities according to the (u, v) coordinates and grouped them into bins so the threads could search for the right visibilities more efficiently.

(Van Amesfoort et al., 2009) focused on maximizing the bandwidth device memory obtained. Therefore, in their implementation, each thread block writes to a private grid stored on the device memory to avoid atomic writes. However, this technique is limited by the size of the grids that can be used and the GPU memory.

(Romein, 2012b) is considered as the reference for recent GPU implementations and introduced a more efficient work distribution strategy for gridding. This strategy was made to minimize device memory access. The grid is divided into subgrids, where the size of each subgrid is the size of the convolution function. For each CUDA block, the number of threads created is equal to the number of grid points in the subgrid. Each thread is focused on a large number of grid points in the grid but only one in the subgrid. Because visibilities coordinates move slowly in time for the same baseline, looping over visibilities in the CUDA kernel also makes the convolution function move slowly. Each thread accumulates the results of the convolved visibility on registers until the grid point is no longer in the subgrid. When the grid point is out of the subgrid, the register is added to the device memory using an atomic add operation.

Thus, this strategy minimizes writing in the device memory by using local registers.

The main improvement of (Merry, 2016) is the introduction of thread coarsening for large convolution functions. Thread coarsening is similar to loop unrolling but applied to parallel work items instead of sequential ones. In its implementation, each thread handles several adjacent grid points within a subgrid. Thanks to thread coarsening, memory accesses within a thread improved because the data for each grid point is close to each other in the device memory. However, this method requires significantly increasing the number of registers per thread.

Finally, the IDG also made a GPU implementation (Veenboer et al., 2017). This is one of the few works proposing implementations for gridder and degridder. However, the IDG algorithm is fundamentally different from the classical gridder. The principle is to do many NUFFT on small grids with very little data each time (van der Tol et al., 2018). In this case, we are not doing a sample convolution on the image but rather a succession of sine, cosine, and shift. This method, therefore, allows DDEs (primary beams, ionospheric screens, etc.) to be corrected transparently and without additional computational cost.

Algorithm 1: G2G - sequential pseudo code.

Data: (...,K, $Igrid$, $Ogrid$, $nhit$)

for i ***in*** *1:M'* **do**
  u, v, w, a_i, ch = getNhit(i, $nhit$) ;
  c_i = (0,0);
  for $u_{tap} \leftarrow [-half\_size]$ *to* $[+half\_size]$ **do**
    for $v_{tap} \leftarrow [-half\_size]$ *to* $[+half\_size]$ **do**
      convU, convV = getConv(u_{tap}, v_{tap}, K);
      wc = convFctConj[w][convU][convV] ;
      c_i += Igrid[$u + u_{tap}$][$v + v_{tap}$][ch] × wc;
    end
  end
  $c_\mathrm{i} = c_\mathrm{i} \times a_i$;
  for $u_{tap} \leftarrow [-half\_size]$ *to* $[+half\_size]$ **do**
    for $v_{tap} \leftarrow [-half\_size]$ *to* $[+half\_size]$ **do**
      convU, convV = getConv(u_{tap}, v_{tap}, K);
      w = convFct[w][convU][convV] ;
      $Ogrid[u + u_{tap}][v + v_{tap}]$[ch] += $c_\mathrm{i} \times$ w;
    end
  end
end

4.2. Grid to Grid sequential details

First, we present a sequential implementation of G2G that serves as the basis for the CPU code. Alg. 1 is the sequential pseudo-code for working with data and grids for one polarization. The diagonal $diag(\boldsymbol{A}^*)$ is stored in an array *nhit* which contains the value of each non-zero element a_i with its corresponding coordinates on the diagonal. The **getNhit** function get the information of the A^* operator through the **nhit** array regarding the for loop iteration.

The algorithm processes sequentially, in the order of storage, the M' non-zero elements $diag(\boldsymbol{A}^*)$. The distance between two consecutive samples for the same baseline is very small during an observation. Therefore, the distance between $diag(\boldsymbol{A}^*)$ elements is also very small, which allows a low spatiality between two iterations.

4.3. Grid to Grid GPU implementation details

we present the work distribution strategy, which implements the G2G method with a grid in the Fourier plane as input and a new grid in the Fourier plane as output. Our STD and G2G GPU implementation are based on Romein's previous implementation (Romein, 2012a) to reduce device-memory access. Additionally, we implement a reduction method to decrease the synchronizations between threads by optimizing the SIMD synchronous instruction within a warp. The G2G method inputs a Fourier grid and outputs another Fourier grid. On the other hand, gridding takes a visibility vector as input and outputs a Fourier grid. The degridding takes a Fourier grid as input and produce visibilities.

The strategy is as follows. We decompose the input and output grids into subgrids, where each subgrid has the size of the convolution function. In the example of Fig. 7, the grids are of size 12×12, and the convolution function is 4×4. In reality, both are much larger. We create a number of threads equal to the number of grid points covering the size of the convolution function, 16 threads in this case (the case of larger kernels will be discussed later). Each thread manages the grid points assigned to it. In our case, a thread manages the ⋆ points and another the • points, which are equivalent grid points on the input grid and the output grid, as illustrated by Fig. 7. As seen in Sec. 4.2, the coordinates of two consecutive a_i elements of the *nhit* array are close to each other. Therefore, the convolution function moves slowly as well, and the grid points of the input grid can be stored in registers. In the same way, the output grid points are first accumulated in registers before being updated in global memory once the thread has to deal with another grid point. With a convolution function of size $n \times n$, this method allows reading and writing with atomicAdd instruction only $2 \times n$ grid points simultaneously. Indeed, in the case of a diagonal move with visibilities close to each other, at most one row and one column will come out of the new grid points, so there would be $2 \times n$ reading and $2 \times n$ writing.

In order to accumulate the value of the compressed visibility on registers, it is necessary to reduce the portion of visibility calculated by each thread. However, the reduction has a low parallelization potential and can become a bottleneck in our algorithm. To minimize the time spent on this reduction, we need a number of threads equal to a power of 2. CUDA has designed several types of methods to optimize the reduction operator (Harris et al., 2007). We have chosen a strategy to optimize this reduction to minimize the mandatory synchronization steps.

The hardware limits the number of threads per block and is often 1024 threads. The W-projection algorithm requires convolution functions of size $C^2_{supp} > 1024$. In this case, like (Merry, 2016), each thread handles several grid points adja-

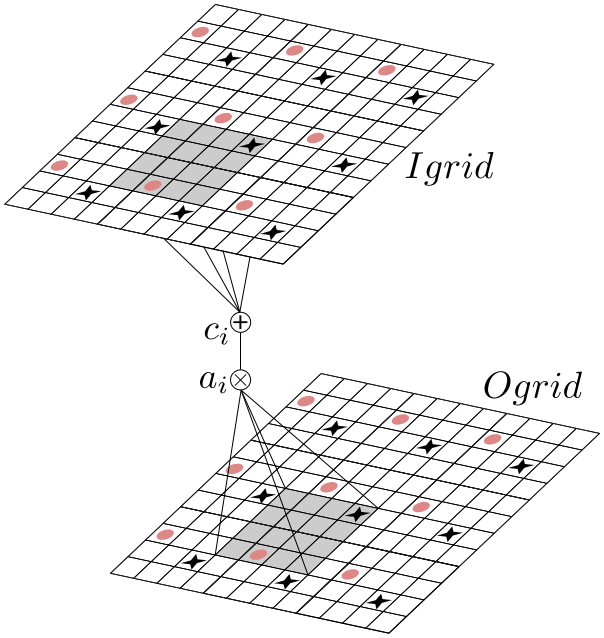

Figure 7: Mapping of work item matching the input and output grids. Each CUDA thread manages the same grid point in the convolution kernel area for the input and output grid. For example, one CUDA thread is in charge of all the grid points tagged by •, and another CUDA thread by **x**.

Algorithm 2: Kernel G2G - pseudo code.

Data: (..., *Igrid*, *Ogrid*, *nhit*)
in_i = (0,0) ;
add_i = (0,0);
c_i = (0,0);
for *i* ***in*** *n* **do**
  u_t, v_t, *w*, a_i, *ch* = getNhit(*i*, *nhit*) ;
  gridU, *gridV* = getGrid(u_t, v_t);
  convU, *convV* = getConv(u_t, v_t);
  if *gridU*! = *prev_gridU* ***or*** *prev_gridV*! = *gridV* **then**
    atomicAdd(*Ogrid*[*prev_gridU*][*prev_gridV*], add_i);
    in_i = *Igrid*[*gridU*][*gridv*] ;
  end
  w = convFct[*w*][*th_convU*][*th_convV*];
  wc = convFctConj[*w*][*th_convU*][*th_convV*];
  c_{xx} = reduce(in_i×wc);
  add_i = c_i × w × a_i ;
end
atomicAdd(*Ogrid*[*prev_gridU*][*prev_gridV*], add_i);

cent within a subgrid. Moreover, the memory accesses are done in loop unrolling to benefit from better memory access for the adjacent grid points. However, the increase in the number of registers prevents us from doing full thread coarsening.

We used a separable approximation of the convolution function. Moreover, like Merry and Romein, we assumed that the convolution function was polarization-independent. The results can, therefore, not be directly applied to the A-projection algorithm. A tunable factor is the number of data of diag A used per thread block. Arbitrarily, we have used 1024 elements per block. As Merry pointed out, higher numbers make more use of spatial coherence between adjacent elements but reduce parallelism.

A pseudo-code of the G2G GPU implementation is represented by Alg. 2. This simplified code shows the case with a single polarization. The pseudo-kernel loops the 1024 $diag(\boldsymbol{A}^*)$ elements, such that Nhit is an array that contains the coordinates and the non-zero values of the diagonal elements.

It is important to note that the STD gridder, STD degridder, and G2G implementations all have the same level of kernel optimization. Thus, comparing the performance of these implementations in the result part can be done without bias.

5. Results

This section aims to show the performance of the G2G algorithm implemented on GPU and CPU. We will study the accuracy and the memory footprint related to the oversampling factor. Then, we will see the performances of the implementation on CPU and GPU before comparing it to other state-of-the-art algorithms. Finally, we will study the roofline model of our implementation in order to show the different possible improvements.

5.1. Experiment setup

For the GPU part, the experiments have been done on a BullSequana XH2000 with an NVIDIA A100-SXM GPU using CUDA. The number of CUDA threads per block used depends on the size of the convolution kernel. Each thread block handles 1024 visibilities, so the total number of blocks is the number of visibilities, 4 000 000, divided by the number of visibilities per block 1024 (as the result is not an integer, the last block deals with less than 1024 visibiltites). For the CPU part, the experiments have been done on BullSequana XH2000 system based on Intel(R) Xeon(R) Platinum 8358 32 cores dual socket. Thus, the code was parallelized on 64 cores.

We will not study the pre-processing necessary for constructing $diag(\boldsymbol{A}^*)$, as it can be done while building the dirty image. Nevertheless, we found that pre-processing performance to make the $diag(\boldsymbol{A}^*)$ matrix is highly dependent on ordering the data by baseline, as in any other gridder implementation.

The data set used a simulation of a 8h observation with the VLA-D with a 1s integration time for 64 frequency channels. The observation has 4 000 000 quad-polarized raw visibilities per frequency channel. The size of the grid used is 1280×1280 pixels following the CASA tutorial[1]. The G2G method is advantageous for compact configurations like VLA-D or NenuFAR. It would be less beneficial for instruments like VLBI.

[1] https://casaguides.nrao.edu/index.php?title=VLA_CASA_Imaging-CASA6.2.0

5.2. Oversampling factor K accuracy

The performance of the G2G method is directly related to the oversampling factor K used for the resolution of the oversampled grid, the *uv* coverage, and the convolution function. For the same oversampling factor, the gradient computed with the STD method is the same as one with the G2G method, such that

$$\nabla J_{G2G}(\boldsymbol{x}_{K=j}) = \nabla J_{STD}(\boldsymbol{x}_{K=j}). \tag{31}$$

Thus, the computational cost is reduced without depreciating the quality of the computed gradient.

First, we looked at the accuracy related to the oversampling factor. The oversampling factor $K = 64$ is the default oversampling factor used in the WSClean imager (Offringa and al., 2014). We will therefore take the gradient from Eq. 15 computed for this K as a reference. Fig. 8 and 9 show the maximum error

$$\text{MaxAE}(K) = \max(\left|\nabla J_{G2G}(\boldsymbol{x}_{K=64}) - \nabla J_{G2G}(\boldsymbol{x}_{K=j})\right|) \tag{32}$$

and the mean absolute error between gradients

$$\text{MAE}(K) = \frac{1}{P}\sum_{i=0}^{P} |\nabla J_{G2G}(\boldsymbol{x}_{i,K=64}) - \nabla J_{G2G}(\boldsymbol{x}_{i,K=j})|. \tag{33}$$

There is a direct impact between the reconstruction quality of the gradient and the oversampling factor. The smaller K is, the stronger the approximation to the true coordinates of the visibilities will be. But we know that a significant approximation tends to decorrelate the data and introduce artifacts in the image (Thompson et al., 2001). It is a known and already observed result (Offringa et al., 2019). Moreover, the error is also dependent on the chosen convolution function.

The problem of accuracy for recent radiotelescopes such as LOFAR or MeerKAT, and soon SKA, is general for both the G2G and STD methods. Indeed, these two methods achieve identical accuracy for the same oversampling factor as they use the same convolution function for the interpolation. New gridding techniques have recently been developed to increase accuracy with a very low oversampling factor (Ye et al., 2022; Barnett et al., 2019).

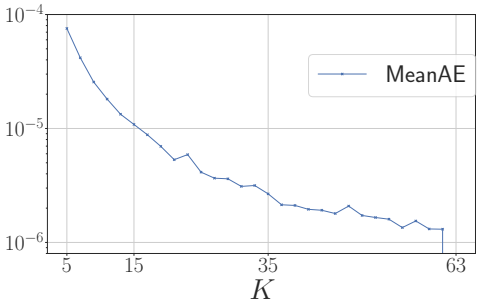

Figure 8: Mean absolute error of the gradient computation regarding the oversampling factor K.

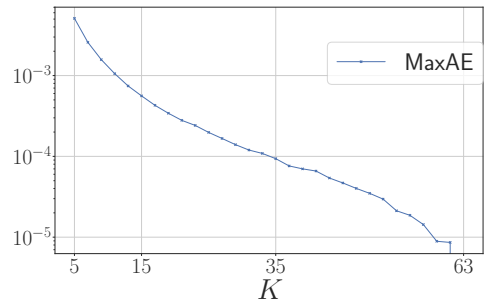

Figure 9: Maximum absolute error of the gradient computation regarding the oversampling factor K.

5.3. Memory footprint

We compare the memory footprint between the G2G method Eq. 17 and the STD method Eq. 12. In the case of STD, the memory footprint is mainly composed of visibilities for all polarity at all frequencies and their respective *uvw* coordinates. The memory footprint is estimated by

$$Mem_{\mathbf{STD}} = (N_r.N_{ch}.N_{pol} + 3.N_r).N_{bytes}, \tag{34}$$

with N_{bytes} the number of bytes used for the precision of the data (single or double float), N_r the number of visibilities for a single frequency channel and a single polarization, N_{ch} the number of frequency channels, and N_{pol} the number of polarizations. The memory footprint of the G2G method is mainly occupied by the dirty image as well as the compressed $diag(\boldsymbol{A}^*)$ calculated in part 3.2. This compressed array has five columns: three for the approximate *uvw* coordinates, one for the slice number, and one for the value of the corresponding element. Therefore, the memory footprint is estimated by

$$Mem_{\mathbf{G2G}} = (N_{Slices}.N_{\text{x}}.N_{\text{y}}.N_{pol}.N_{bytes}) + (5.n_{coo}.N_{int32}) \tag{35}$$

with n_{coo} is the length of the compressed $diag(\boldsymbol{A}^*)$ matrix, N_{Slices} is the number of slices in the hypercube, N_{bytes} is the number of bytes for the floating numbers, and N_{int32} is the number of bytes for an $int32$. The experimental results are presented in table 1 for several oversampling factors.

	K	**8**	**16**	**24**	**32**	**40**	**64.**
STD	*single*	3.8	3.8	3.8	3.8	3.8	3.8
	double	7.7	7.7	7.7	7.7	7.7	7.7
G2G	*single*	1.7	1.85	1.99	2.13	2.27	2.66
64 slices	*double*	3.27	3.42	3.56	3.70	3.83	4.22
G2G	*single*	0.17	0.32	0.46	0.60	0.73	1.12
1 slice	*double*	0.19	0.34	0.48	0.62	0.76	1.15

Table 1: Memory footprint needed to compute the gradient in GB.

The memory footprint remains constant regardless of K for the STD method. For the G2G method, the memory footprint varies with K. The smaller the K, the smaller the memory footprint because c_α will be smaller. Furthermore, we compare two

cases. The first is with a dirty image with as many slices as frequency channels. For each K, the memory footprint is smaller than the STD method. The main weight comes from the dirty image. In the case of the dirty image with one slice for all frequencies, the memory footprint decreases drastically. For each K, the main weight of the memory footprint is α. With $K = 8$, the memory footprint used with this method represents 4.4% of the memory footprint used with the STD method.

The memory footprint results follow the same logic as the compression ratio results in Tab. 2. The lower the sampling factor, the higher the compression. For $K = 64$, the required data volume is 27.8% of the initial data set. For $K = 8$, c_α is only 3.4%. The difference between c_α and $\widehat{c_\alpha}$ is very small, so the theoretical compression gives reliable information of the gain more quickly.

K	**8**	**16**	**24**	**32**	**40**	**64.**
$\widehat{c_\alpha}$	3.9%	7.7%	11.4%	15.1%	18.6%	28.8%
c_α	3.4%	6.9%	10.4%	13.9%	17.3%	27.8%

Table 2: Theoretical $\widehat{c_\alpha}$ and experimental c_α compression ratio.

The G2G method obtains compression gain under $M' < M$. In practice, this is the case for most radio telescopes. The more compact the array, like VLA or NenuFar, the greater the compression gain. For less compact arrays, such as LOFAR or SKA, a compression gain is still expected, but less by an order of magnitude. On the other hand, the memory cost for non-compact networks working on large images can be dominated by saving the dirty image on several frequency bands. As a result, despite lossless data compression and reduced computational cost, the memory cost with some networks can be very high.

5.4. GPU performances

We will use three metrics to measure the CPU and GPU implementation performance. The first metric is the execution time in seconds. The second metric, define as giga Grid Point per Second (GPS), is the number of pixels updated or read per second during the execution of the kernel such that

$$GPS_{G2G}(K) = \frac{\#\text{FourierPoint}_{G2G}.C_{supp}^2}{t_{G2G}(K)}.\times 10^{-9}, \qquad (36)$$

with $t_{G2G}(K)$ is the GPU execution time for the G2G kernel regarding K, and $\#\text{FourierPoint}_{G2G} = M'.N_{pol}$ is the number of Fourier-point to be gridded and degridded. The last metric define as the compressed Million Fourier-point Throughput (MFT) is the number of data processed in the G2G kernel, such as

$$MFT_{G2G}(K) = \frac{\#\text{FourierPoint}_{G2G}}{t_{G2G}(K)} \times 10^{-6}. \qquad (37)$$

Fig. 10 and Fig. 11 show the execution time of the G2G method for several convolution function sizes as a function of the oversampling factor for the CPU and the GPU implementations. In both cases, we can see the influence of the oversampling factor in the execution time because the number of elements to be processed depends on K. Moreover, the increase in execution time as a function of the convolution function is globally well proportional to its size. The exception is the case with $C_{supp} = 64$ in GPU, where each thread takes care of several pixels and tends to accelerate the CUDA kernel. We can notice an order of magnitude difference in favor of the GPU implementation compared to the 64-core parallelized CPU implementation. This result confirms the interest in GPU implementation, even when we can use high-performance CPUs with many cores.

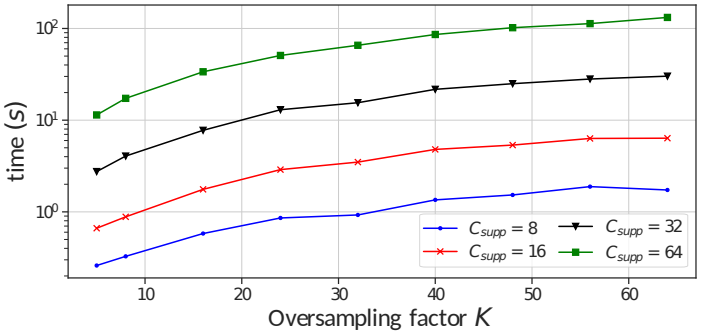

Figure 10: CPU - G2G computation time regarding the oversampling factor K.

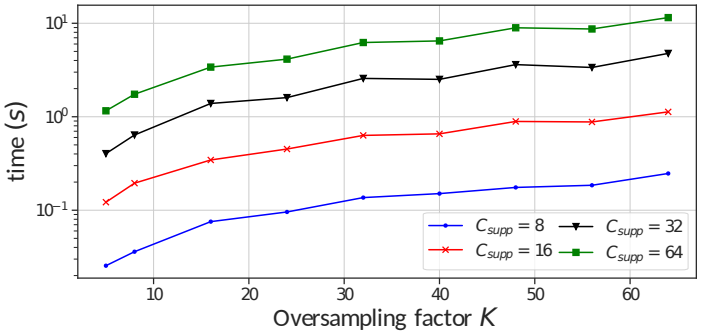

Figure 11: GPU kernel - G2G computation time regarding the oversampling factor K.

Fig. 12 shows the evolution of GPS as a function of K for several convolution function sizes. For each C_{supp}, the overall trend is an increase in performance as the oversampling factor is higher. However, the evolution remains choppy, and some K performs better than others. Surprisingly, this is not with K being a power of 2. The performances are separated into two blocks. The first one is with $C_{supp} = 16$ and $C_{supp} = 32$, whose performances are slightly lower than the others. The second with $C_{supp} = 8$ and $C_{supp} = 64$, whose last one confirms the good performances on execution time.

Fig. 13 illustrates the data throughput of the G2G method as a function of K for several convolution functions. As expected, the larger the convolution function, the lower the data throughput. Indeed, more CUDA threads are allocated to process a

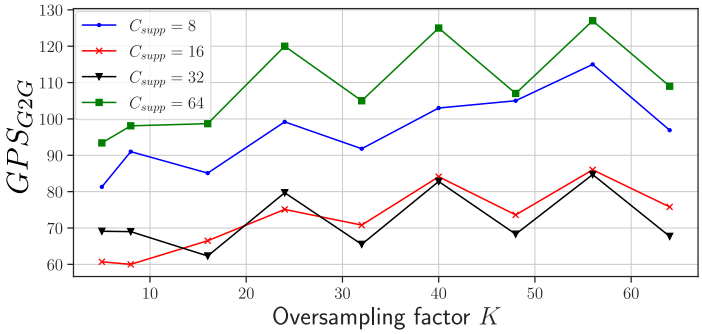

Figure 12: GPU kernel - GPS_{G2G} regarding the oversampling factor K.

sample. As a result, the number of blocks executed in parallel is lower, thus reducing the throughput. On the other hand, the throughput remains relatively stable as a function of K.

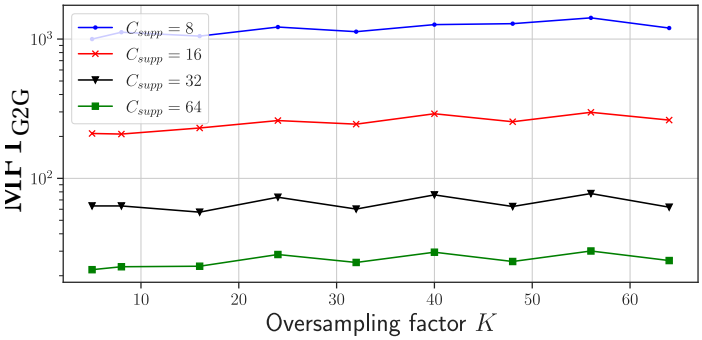

Figure 13: GPU kernel - MFT_{G2G} Fourier-point throughput of G2G method regarding the oversampling factor K.

5.5. Comparison with STD

We also compare the GPU state-of-the-art gridder and degridder implementations following the Romein implementation with the G2G method. In this comparison, the STD method is built from the GPU implementations of these gridder and degridder. Moreover, we will only take into account the CUDA kernel times without taking into account the memory transfer times. We take this assumption for the case where the kernel times would cover the memory transfer costs. The metrics used to compare the performance of the G2G method with the STD method are the same as the previous one. The GPS is calculated as

$$GPS_{STD} = \frac{\#\text{FourierPoint}_{STD}.C_{supp}^2}{t_{grid} + t_{degrid}} \times 10^{-9}, \qquad (38)$$

with $\#\text{FourierPoint}_{STD} = N_r.N_{ch}.N_{pol}$ is the number of visibilities to be gridded and degridded. The Fourier-point throughput is calculated as

$$MFT_{STD} = \frac{\#\text{FourierPoint}_{STD}}{t_{grid} + t_{degrid}} \times 10^{-6}. \qquad (39)$$

For the G2G and STD methods, we take an oversampling factor $K = 8$ for the convolution functions.

Fig. 14 compared the performances of the two methods. For $C_{supp} = 64$, the STD method is slightly superior by 2%. For all other cases, the G2G method is better. For the case of $C_{supp} = 8$, the MFT is 38% better for the G2G method than for the STD method, and goes down to 14% better for $C_{supp} = 32$. It should be noted that these experiments greatly benefit the STD method since the raw visibilities used are taking great advantage of the GPU implementation to reduce the memory bandwidth. Finally,

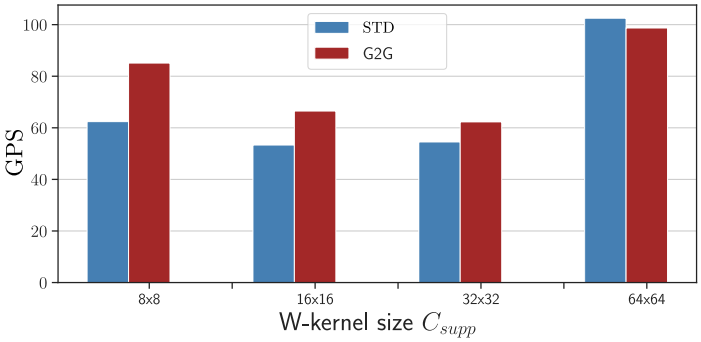

Figure 14: GPU kernel - GPS comparison between G2G and STD methods for $K = 8$.

we compare the Fourier-point throughput processed per second for the different methods. The results are presented in Fig. 15. The performances for STD and G2G follow the previous result, and G2G has better performances for small convolution functions.

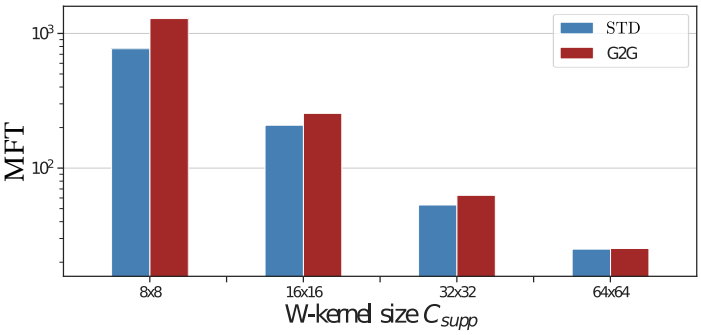

Figure 15: GPU kernel - Fourier-point throughput comparison between G2G, STD.

5.6. Roofline model

The roofline model (Williams et al., 2009) is a tool to visualize the possible limitations of an algorithm with respect to the maximum theoretical performance of the target architecture. The model is characterized by the device's peak performance in FLOPs and by the attainable memory bandwidth. This model aims to indicate the possible bottleneck of the specific algorithm. The roofline model for the GPU implementation of the G2G, the gridding and degridding algorithms are represented by Fig. 16 and describes the achievable performances in FLOPS compared to the Arithmetic Intensity (AI) such as

$$AI = \frac{\#\text{OP}}{\#\text{memory access}} \tag{40}$$

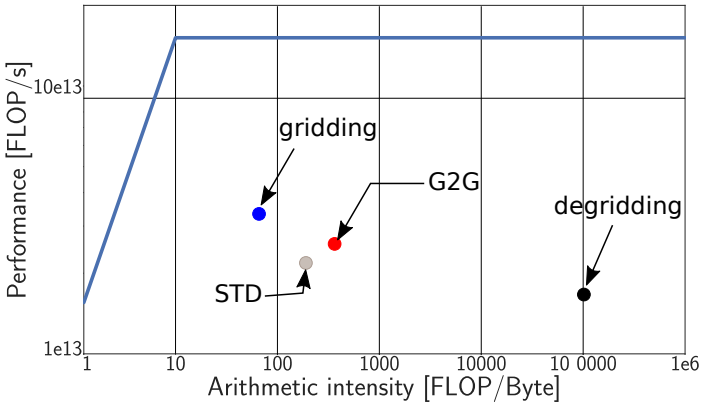

Figure 16: Roofline model.

The three points are in the compute-bound area without being limited by the theoretical power of the GPU. The gridding reaches 22% of the theoretical performance, the degridding 11%, the STD 14%, and G2G between with 17%. The STD point was built from the gridding and degridding points. G2G improves the arithmetic intensity compared to STD. The result also shows a large margin of performance improvement before reaching the theoretical peak since the bandwidth does not limit the methods. In order to improve the performance, it is necessary either to increase the number of operations performed in the kernel time or to decrease the kernel time without changing the number of operations. The memory access time to access the convolution function coefficients and the lack of parallelism of the reduction operation are known to be bottlenecks. We can therefore expect much better results by improving these two points. One possible improvement is integrating the full Merry's implementation, Merry (2016), with coarsening threads.

6. Conclusion

We presented the Grid to Grid method used in radio interferometric imaging that can be seen as a generalization of the BDA. The gridding and degridding operators have a high computational cost. The G2G method shows the computational cost reduction by merging these two operators in a theoretical and experimental way. The main cost reduction factor is the oversampling factor used for the convolution function and the fineness of the approximation of true *uv* coordinates of the visibilities. This method makes no degradation of the reconstruction quality compared to classical gridding and degridding algorithms for the same oversampling factor K. Moreover, it can be used in addition to other averaging methods.

We have shown the impact of the oversampling factor on the precision and the error generated by it. We have also shown its impact on the memory footprint and execution time. Consequently, a compromise to be made on the oversampling factor to balance the computation acceleration while limiting the error. The G2G method showed a compression gain for all types of radio telescopes, especially if the array was compact. For non-compact arrays, we still expect a compression gain and a lower computational cost but potentially a higher memory cost due to dirty image storage. GPU implementation of the G2G methods is well suited by increasing the performance with a factor 10 regarding the CPU implementation. Finally, we showed that the GPU implementation of G2G was more efficient than the GPU implementation of a gridding/degridding w-projection algorithm with the same optimization level. The Fourier-point throughput is up to a 37% increase. On the other hand, the limitations come from the absence of direction-dependent effects processing as with A-projection. This algorithm requires much more memory access to load the convolution functions, which are time, frequency, baseline dependent, and not separable.

In future work, we plan to improve the CUDA kernel at several levels. The first is using texture to store convolution functions to improve accuracy. The second is to fully implementing Merry's GPU optimization is also a promising direction. In addition, we plan to implement the w-stacking method (Offringa and al., 2014), which allows using a low oversampling factor and thus works on larger images. Moreover, a higher level improvement is the possibility of using a hybrid version of w-projection and w-stacking. Finally, port the CUDA code to HIP, (Kondratyuk et al., 2021), in order to use the code on Nvidia or AMD platforms in a transparent way.

Acknowledgment

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

References

Atemkeng, M.T., Smirnov, O.M., Tasse, C., Foster, G., Jonas, J., 2016. Using baseline-dependent window functions for data compression and field-of-interest shaping in radio interferometry 462, 2542–2558. doi:10.1093/mnras/stw1656.

Barnett, A.H., Magland, J.F., Klinteberg, L.a., 2019. A parallel non-uniform fast Fourier transform library based on an "exponential of semicircle" kernel. arXiv:1808.06736 [cs, math] ArXiv: 1808.06736.

Beatty, P., Nishimura, D., Pauly, J., 2005. Rapid gridding reconstruction with a minimal oversampling ratio. IEEE Transactions on Medical Imaging 24, 799–808. doi:10.1109/TMI.2005.848376.

Bester, H.L., Repetti, A., Perkins, S., Smirnov, O.M., Kenyon, J.S., 2021. A practical preconditioner for wide-field continuum imaging of radio interferometric data `arXiv:2101.08072`.

Cheng, J., Grossman, M., McKercher, T., 2014. Professional CUDA c programming. John Wiley & Sons.

Cornwell, T.J., Golap, K., Bhatnagar, S., 2008. The non-coplanar baselines effect in radio interferometry: The w-projection algorithm. IEEE Journal of Selected Topics in Signal Processing 2, 647–657.

Edgar, R.G., Clark, M.A., Dale, K., Mitchell, D.A., Ord, S.M., Wayth, R.B., Pfister, H., Greenhill, L.J., 2010. Enabling a high throughput real time data pipeline for a large radio telescope array with GPUs. Computer Physics Communications 181, 1707–1714. doi:10.1016/j.cpc.2010.06.019.

Fessler, J., Sutton, B., 2003. Nonuniform fast fourier transforms using min-max interpolation. IEEE Trans. Signal Process. 51, 560–574. doi:10.1109/TSP.2002.807005.

Fessler, J.A., 2007. On NUFFT-based gridding for non-cartesian MRI. Journal of Magnetic Resonance 188, 191–195. doi:10.1016/j.jmr.2007.06.012.

Giovannelli, J.F., Coulais, A., 2005. Positive deconvolution for superimposed extended source and point sources. Astronomy & Astrophysics 439, 401–412. doi:10.1051/0004-6361:20047011, arXiv:astro-ph/0507691.

Harris, M., et al., 2007. Optimizing parallel reduction in cuda. Nvidia developer technology 2, 70.

Kondratyuk, N., Nikolskiy, V., Pavlov, D., Stegailov, V., 2021. Gpu-accelerated molecular dynamics: State-of-art software performance and porting from nvidia cuda to amd hip. The International Journal of High Performance Computing Applications 35, 312–324.

Merry, B., 2016. Faster GPU-based convolutional gridding via thread coarsening 16, 140–145. doi:10.1016/j.ascom.2016.05.004, arXiv:1605.07023.

Muscat, D., 2014. High-Performance Image Synthesis for Radio Interferometry. Ph.D. thesis. ArXiv: 1403.4209.

Offringa, A.R., al., 2014. WSClean: an implementation of a fast, generic wide-field imager for radio astronomy. Monthly Notices of the Royal Astronomical Society 444, 606–619. doi:10.1093/mnras/stu1368, arXiv:1407.1943.

Offringa, A.R., Mertens, F., van der Tol, S., Veenboer, B., Gehlot, B.K., Koopmans, L.V.E., Mevius, M., 2019. Precision requirements for interferometric gridding in the analysis of a 21 cm power spectrum. Astronomy & Astrophysics 631, A12. doi:10.1051/0004-6361/201935722.

Romein, J.W., 2012a. 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. p. 321. doi:10.1145/2304576.2304620.

Romein, J.W., 2012b. 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, San Servolo Island, Venice, Italy. p. 321. doi:10.1145/2304576.2304620.

Schwab, F.R., 1984. Relaxing the isoplanatism assumption in self-calibration; applications to low-frequency radio interferometry. The Astronomical Journal 89, 1076. doi:10.1086/113605.

Sullivan, I., al., 2012. Fast holographic deconvolution: a new technique for precision radio interferometry. The Astrophysical Journal 759, 17. doi:10.1088/0004-637X/759/1/17, arXiv:1209.1653.

Tasse, C., van der Tol, S., van Zwieten, J., van Diepen, G., Bhatnagar, S., 2013. Applying full polarization A-Projection to very wide field of view instruments: An imager for LOFAR. A&A 553, A105. doi:10.1051/0004-6361/201220882.

Thiébaut, E., Young, J., 2017. Principles of image reconstruction in optical interferometry: tutorial. Journal of the Optical Society of America A 34, 904. doi:10.1364/JOSAA.34.000904.

Thompson, A.R., Moran, J.M., Swenson, G.W., 2001. Interferometry and synthesis in radio astronomy. 2nd ed ed., Wiley.

Thévenaz, P., Blu, T., Unser, M., 2000. Image interpolation and resampling , 39.

van der Tol, S., Veenboer, B., Offringa, A.R., 2018. Image domain gridding: a fast method for convolutional resampling of visibilities 616, A27. doi:10.1051/0004-6361/201832858.

Van Amesfoort, A.S., Varbanescu, A.L., Sips, H.J., van Nieuwpoort, R.V., 2009. Evaluating multi-core platforms for HPC data-intensive kernels, in: Proceedings of the 6th ACM conference on Computing frontiers - CF '09, ACM Press, Ischia, Italy. p. 207. doi:10.1145/1531743.1531777.

Veenboer, B., Petschow, M., Romein, J.W., 2017. Image-domain gridding on graphics processors, in: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE. pp. 545–554. doi:10.1109/IPDPS.2017.68.

Villarino, M.B., 2005. Ramanujan's perimeter of an ellipse. arXiv preprint math/0506384 .

Wijnholds, S.J., Willis, A.G., Salvini, S., 2018. Baseline-dependent averaging in radio interferometry 476, 2029–2039. doi:10.1093/mnras/sty360.

Williams, S., Waterman, A., Patterson, D., 2009. Roofline: an insightful visual performance model for multicore architectures. Commun. ACM 52, 65–76. doi:10.1145/1498765.1498785.

Ye, H., Gull, S.F., Tan, S.M., Nikolic, B., 2022. High accuracy wide field imaging method in radio interferometry. Monthly Notices of the Royal Astronomical Society 510, 4110–4125. doi:10.1093/mnras/stab3548. arXiv: 2101.11172.

Zhili Yang, Jacob, M., 2009. Efficient NUFFT algorithm for non-Cartesian MRI reconstruction, in: 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, IEEE, Boston, MA, USA. pp. 117–120. doi:10.1109/ISBI.2009.5192997.