Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **476,** 4527–4542 (2018)
Advance Access publication 2018 February 16

doi:10.1093/mnras/sty409

GALARIO: a GPU accelerated library for analysing radio interferometer observations

Marco Tazzari,[1]★ Frederik Beaujean[2] and Leonardo Testi[2,3]

[1] *Institute of Astronomy, University of Cambridge, Madingley Road, CB3 0HA Cambridge, UK*
[2] *C2PAP, Excellence Cluster Universe, Ludwig-Maximilians-Universität München, Boltzmannstr 2, D-85748 Garching, Germany*
[3] *European Southern Observatory, Karl-Schwarzschild-Str 2, D-85748 Garching, Germany*

Accepted 2018 February 10. Received 2018 February 18; in original form 2017 September 19

ABSTRACT

We present GALARIO, a computational library that exploits the power of modern graphical processing units (GPUs) to accelerate the analysis of observations from radio interferometers like Atacama Large Millimeter and sub-millimeter Array or the Karl G. Jansky Very Large Array. GALARIO speeds up the computation of synthetic visibilities from a generic 2D model image or a radial brightness profile (for axisymmetric sources). On a GPU, GALARIO is 150 faster than standard PYTHON and 10 times faster than serial C++ code on a CPU. Highly modular, easy to use, and to adopt in existing code, GALARIO comes as two compiled libraries, one for Nvidia GPUs and one for multicore CPUs, where both have the same functions with identical interfaces. GALARIO comes with PYTHON bindings but can also be directly used in C or C++. The versatility and the speed of GALARIO open new analysis pathways that otherwise would be prohibitively time consuming, e.g. fitting high-resolution observations of large number of objects, or entire spectral cubes of molecular gas emission. It is a general tool that can be applied to any field that uses radio interferometer observations. The source code is available online at http://github.com/mtazzari/galario under the open source GNU Lesser General Public License v3.

Key words: methods: numerical – techniques: interferometric – submillimetre: general.

1 INTRODUCTION

In the quest for high angular resolution and high sensitivity, radio astronomy has been developing the use of interferometry since the late 1940s. Unlike single dishes, which directly measure the sky brightness and produce an image of it, radio interferometers measure *visibilities*, the complex valued samples of the Fourier transform of the sky brightness (Thompson 1999). The locations in the Fourier plane where these samples are taken are given by the spatial distribution of the antennas on the ground and the direction of the source being observed. Modern interferometers like Atacama Large Millimeter and sub-millimeter Array (ALMA) and the Karl G. Jansky Very Large Array (VLA) have developed advanced pipelines that not only calibrate the observed visibilities, but also produce for the end users spectrally resolved images of the sky brightness distribution.

Comparing a model prediction to an interferometric data set is typically done in one of the following two ways: either in the image plane by comparing a model image to the image of the sky reconstructed from the visibilities, or in the Fourier plane by directly comparing the observed visibilities to synthetic ones computed from the model image. The first approach is more intuitive but it is intrinsically limited: it relies on estimating the true sky brightness distribution from the observed visibilities. Unfortunately, the observations can only provide a finite number of samples of the visibilities, implying that a unique reconstruction of the sky brightness is not possible. In addition, to remove the effects of discrete sampling, non-linear deconvolution algorithms (e.g. the traditional CLEAN by Högbom 1974; Clark 1980 or MEM by Cornwell & Evans 1985) are applied to perform image reconstruction, which may introduce a variety of artefacts. Moreover, while each of the observed visibility point has associated a well-behaved Gaussian noise (with equal variance in the real and imaginary part), the pixels in the reconstructed image have correlated noise whose properties are poorly constrained (due to the non-linear reconstruction algorithms). Ultimately, model comparison to the reconstructed images is thus affected in the image plane by the sampling of the sky visibility, the non-linear algorithms applied, and the correlated noise on the images, which reflects in the difficulty to correctly estimate the observational uncertainty (Cornwell, Braun & Briggs 1999). The second approach – comparing observed to model

★ E-mail: mtazzari@ast.cam.ac.uk

visibilities – is much more straightforward as it operates in the domain where the observations were made and the uncertainties are better understood (Pearson 1999).

Comparing a model image computed on a regular grid to observed visibilities that are scattered across the Fourier plane involves a series of 1D and 2D array operations such as Fourier transforms, transpositions, and interpolations (Briggs, Schwab & Sramek 1999). The size of the arrays used to properly model the visibilities is set by the properties of the interferometer at observation time: spatial and spectral resolution, sensitivity, number, and distribution of antennas. ALMA and the VLA have delivered tremendous improvements in terms of longer baselines, higher sensitivity, and more uniform Fourier plane coverage. This implies that the spectral and spatial sampling of the visibilities has increased enormously.

Inferring a model from the observations – either using a Bayesian Markov chain sampler or a classical χ^2 optimizer – requires an adequate exploration of the parameter space. Performing the inference in the Fourier plane requires the computation of synthetic visibilities from the model image in each likelihood evaluation. The enhancements in the quality of the visibilities delivered by modern interferometers have increased the computational effort required to model them accordingly, at a point where modelling medium-resolution observations can take one or more days of multicore computation.[1]

The large software packages designed for the calibration and the management of interferometric data sets – e.g. CASA,[2] AIPS,[3] MIRIAD (Sault, Teuben & Wright 1995), GILDAS[4] – usually offer dedicated tasks for modelling the visibilities. However, although handy for a first characterization of the observed sources, these tasks are often very limited in terms of flexibility: they typically require the user to choose among a very restricted set of simplified models for the source brightness, do not allow the user to specify what statistics should be used for the exploration of the parameter space, and cannot be easily incorporated into external modelling codes without a heavy performance penalty. In this context, the CASA-based UVMULTIFIT library (Martí-Vidal et al. 2014) constitutes a more flexible solution as it allows the user to model the visibilities with an indefinite number of parametric source components that can be personalized. We note that all the codes named so far are purely designed for CPUs, and only a few of them (e.g. CASA and UVMULTIFIT) can benefit from multicore operations.

A breakthrough in the computing capabilities is needed in order to fully and timely exploit the wealth of information that the new interferometers make available. In this paper we present GALARIO, a computational library that provides the necessary speed-up. Unlike the central processing units (CPUs) that are composed of at most a few tens of cores, the graphical processing units (GPUs) have thousands of cores that, although less powerful than CPU cores, effectively outperform CPUs in embarrassingly parallel tasks (Nickolls et al. 2008), of which the operations needed to compute the synthetic visibilities are eminent examples.

GALARIO is a library that uses GPUs or alternatively multiple CPU cores to speed up the computation of synthetic visibilities from a model image, and has been designed to achieve the best performance and still to be easy to use and to adopt in existing code. In the context of a fit, GALARIO can be easily adopted as a drop-in replacement to accelerate the computation of the χ^2 between the model predictions and the observed visibilities. Moreover, thanks to its modular structure, GALARIO can be included in any likelihood computation, leaving to the user the choice of the statistical tool used for the parameter space exploration. The GPU version of GALARIO is about 150 times faster than standard PYTHON implementations that rely on the widely used SCIPY and NUMPY packages, and 10 times faster than serial C code. From the user perspective, GALARIO can be called directly in C or C++ and easily imported in PYTHON code as a normal package.

To our knowledge, there is only another code, MONTBLANC (Perkins et al. 2015) that exploits the power of GPUs to compare models directly to observed visibilities. However, GALARIO differs from MONTBLANC in many aspects. First, MONTBLANC models the source brightness only through parametrized models (e.g. a point source, or a Gaussian ellipse) and does not support, as yet, unparametrized radio sources. Instead, GALARIO allows the user to compute synthetic visibilities from a generic 2D image of the sky brightness that can be the result, e.g. of a complex radiative transfer computation as well as of a simple parametric profile. Secondly, while MONTBLANC is dedicated to GPUs, all the functions of GALARIO are implemented both for GPU and CPU, on which the acceleration is achieved with OpenMP. Moreover, since the GPU and CPU functions in GALARIO have the same interfaces, it is easy to write reusable code that can be executed on the GPU or on the CPU just by changing which library is linked in (C) or imported (PYTHON).

The contexts in which GALARIO can be used are manifold. Originally developed in the field of protoplanetary discs, GALARIO implements a general computation of the synthetic visibilities that makes it suitable for application in any field dealing with observations from radio interferometersfor a wide range of wavelengths and angular resolutions.

GALARIO has already been used in a few studies to fit moderate- and high-resolution observations of protoplanetary discs. In Testi et al. (2016) and Tazzari et al. (2017) GALARIO was used to fit the visibilities of the disc continuum emission with a physical model to characterize the disc structure. Tazzari et al. (2016) used GALARIO to study the properties of the dust grains through the simultaneous fit of visibilities at multiple sub-mm, mm, and cm wavelengths. In the domain of extreme high-resolution observations GALARIO has been used to characterize the shape of the multiple rings appearing in the continuum emission of the AS 209 protoplanetary disc seen by ALMA (Fedele et al. 2018). It is worth noting that the speed-up that GALARIO delivers naturally translates into the capability to extend the visibility analysis to many objects on much reduced timescales, thus making it ideal to fit surveys of many sources (e.g. as has been done in Tazzari et al. 2017). Furthermore, a new pathway opened by the acceleration of GALARIO is the possibility to fit simultaneously entire spectral cubes of molecular-gas emission, allowing the kinematics of the object – a protoplanetary disc or a galaxy – to be characterized consistently.

The paper is organized as follows. Section 2 provides an overview of the code illustrating key functionalities and relevant use cases. Section 3.1 introduces the theoretical definitions and equations of Synthesis Imaging and discusses the limitations of the current release of GALARIO. Section 4 describes the CPU and the GPU implementation of GALARIO and Section 5 presents the results of accuracy checks. In Section 6 we analyse the performance of GALARIO and in Section 7 we draw our conclusions. Appendix A

[1] A representative fit of a single wavelength continuum map at 0.1 arcsec resolution, assuming 4096×4096 matrix size, 10^6 visibilities, and 0.5 s to compute them with a standard PYTHON code, 10^6 likelihood evaluations to achieve convergence, running on 32 CPU cores, needs 49 wall-clock hours (excluding the model computation).

[2] https://casa.nrao.edu

[3] http://www.aips.nrao.edu

[4] http://www.iram.fr/IRAMFR/GILDAS

summarizes the steps needed to obtain and install GALARIO. Appendix B reports additional performance tests analogous to those discussed in Section 6. Appendix C shows the results of additional accuracy checks carried out against the CASA package. Appendix D presents the PYTHON implementation of some reference functions.

2 CODE OVERVIEW

In this section we aim to give a quick overview of GALARIO for typical use cases that the reader might find immediately useful, deferring to Section 3.1 the definition of the quantities and equations involved. GALARIO has been designed to accelerate the fundamental task of comparing a model prediction (fundamentally, a brightness image) with an interferometric observational data set, which typically consists of a collection of complex visibilities V_k ($k = 1...M$) defined as the samples of the source visibility V in discrete locations (u_k, v_k). The key functionality of GALARIO is the computation of synthetic visibilities given (i) a model image (or a radial brightness profile) and (ii) a collection of uv-points (u_k, v_k) representing the interferometric baselines sampled by the observations.

The core of GALARIO is written in C++ (for the CPU version) and CUDA C++ (for the GPU version). This allows GALARIO to achieve the best performances and to offer the same core functionalities in both versions. The PYTHON wrappers written in CYTHON are available for the main functions to facilitate the adoption of GALARIO in existing code.

On machines where no GPU is available, GALARIO can still provide a speed-up through OpenMP on multiple CPU cores. If compiled and executed on machines with a CUDA enabled GPU,[5] GALARIO delivers a dramatic speed-up with respect to normal CPU code, up to 150 times faster than a standard PYTHON implementation that uses the NUMPY and SCIPY packages (more details in Section 6).

2.1 Selection of the version

Both the CPU and GPU versions of GALARIO are compiled in single and double precision. After installation, the CPU and GPU versions can be imported in PYTHON with

```
from galario import double      # CPU
from galario import double_cuda # GPU
```

The single- and double-precision libraries in both the CPU and GPU versions offer the same functions with identical interfaces, thus making it easy to write reusable code. Our recommended default is double precision. To use the single-precision versions, replace `double` → `single` in the above commands. The functions described below can be imported from any of these four libraries.

2.2 Basic usage

The computation of the synthetic visibilities $V_{\rm mod}$ of a model image, sampled at some uv-points (u_k, v_k) can be done with `sampleImage`:

```
from double_cuda import sampleImage
Vmod = sampleImage (image, dxy, u, v)
```

[5] The updated list of CUDA enabled GPUs is available at https://developer.nvidia.com/cuda-gpus

where the `image` is a 2d array in Jy pixel^{-1} units and its coordinate system is the same as that of the sky (East to the left, North to the top), `dxy` is the size (in radians) of the image pixel (assumed square), `u` and `v` are linear arrays containing the coordinates u_k, v_k (expressed in units of the observing wavelength λ), and the returned array `Vmod` is a complex array containing the synthetic visibilities (in Jy).

`sampleImage` makes no assumptions on the symmetry of the 2D input `image` and therefore can be used to compute the visibilities of any image. However, in case the model image has an axisymmetric brightness distribution, GALARIO offers a faster version of `sampleImage` called `sampleProfile` that exploits the symmetry of the image and takes as input the brightness profile $I_\nu(r)$ defined on a radial grid and computes internally the 2D image by azimuthally sweeping the profile over 2π:

```
from double_cuda import sampleProfile
Vmod = sampleProfile(I, Rmin, dR, Nxy, dxy, u, v)
```

where `I` is a 1d array containing the radial brightness profile $I_\nu(R)$ (in Jy sr^{-1}), `Rmin` and `dR` are the innermost radius and the cell size of the radial grid expressed in radians, and `Nxy` is the number of pixels on each image axis. Fig. 1 summarizes the workflow of `sampleProfile`: the radial brightness profile (left-hand panel) is used to produce an axisymmetric 2D image (central panel) which is then Fourier transformed and sampled in the specified uv-points (right-hand panel).

The instruction above produces a face-on 2D image out of the profile $I_\nu(r)$. Producing an image with an inclination `inc` (radians) along the line of sight can be done by specifying the optional parameter `inc`:

```
Vmod = sampleProfile (I, Rmin, dR, Nxy, dxy, u, v,
inc=inc)
```

as shown in the example in Fig. 1 for an inclination of 45°.

In the context of a fit, GALARIO provides handy functions to compute directly the likelihood of the model in terms of a χ^2, both in the case the input is a model image or an axisymmetric brightness profile:

```
chi2 = chi2Image   (image, dxy, u, v, Re_Vobs,
                    Im_Vobs, w)
chi2 = chi2Profile (I, Rmin, dR, Nxy, dxy,u, v,)
                    Re_Vobs, Im_Vobs, w)
```

where `Re_Vobs`, `Im_Vobs` are the real and imaginary part of the observed visibilities and `w` their associated weights.

All the functions described so far support optional parameters useful to rotate and translate the model image given. It is possible to rotate the model image by a position angle PA, and to translate it by angular offsets in Right Ascension and Declination direction (ΔRA, ΔDec.) by specifying the optional parameters:

```
Vmod = sampleImage(image, dxy, u, v,
       PA=PA, dRA=ΔRA, dDec=ΔDec)
```

where PA, (ΔRA, ΔDec.) are all expressed in radians and the offsets are defined in sky coordinates, i.e. positive ΔRA and ΔDec. translate the image towards east and north, respectively. Fig. 2 illustrates these definitions. The same optional parameters can be specified in `sampleProfile`, `chi2Image`, and `chi2Profile`. As described in Section 3.3, to achieve better performances the image rotation and translation are not applied to the model image but to the synthetic visibilities.

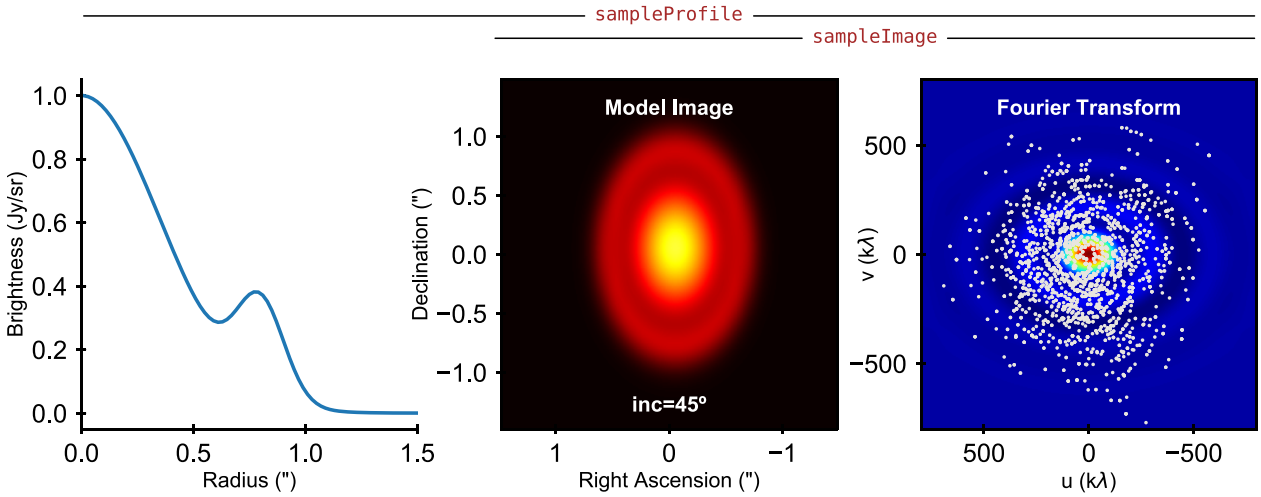

Figure 1. Workflow of the `sampleProfile` and `sampleImage` functions. `sampleImage` takes in input a 2D model image (central panel) and produces the synthetic visibilities by sampling its Fourier transform at the specified *uv*-points locations (right-hand panel). `sampleProfile` computes the synthetic visibilities in the same way as `sampleImage`, but takes in input a radial brightness profile (left-hand panel) from which the model image is internally computed assuming axisymmetry and a line-of-sight inclination (45° in this example).

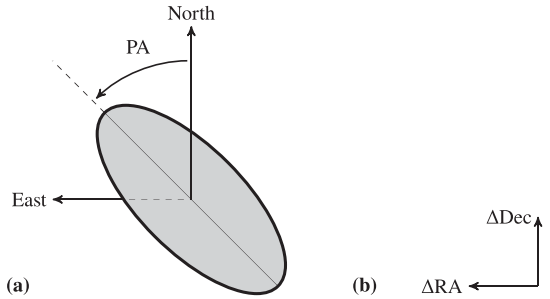

Figure 2. Conventions used in GALARIO. (a) Definition of inclination and position angle (PA). A circular disc is inclined by 55° and rotated by the position angle PA. The inclination is performed with a tilt along the north–south axis before rotating by PA (see e.g. the central panel in Fig. 1). PA is the angle between the north–south axis and the line of nodes – the intersection of the plane of the object with the north-east plane – and is measured counter-clockwise (east of north). (b) Definition of the angular offsets. ΔRA and ΔDec. are positive for offsets towards East and North, respectively.

Hereafter we will refer to the `sampleProfile` and `sampleImage` functions by `sample*`, and analogously by `chi2*` for the `chi2Profile` and `chi2Image` functions. In a similar way, we will use `*Profile` and `*Image` to indicate the related functions.

Details on how to install GALARIO are given in Appendix A and, more thoroughly, in the online documentation at https://mtazzari.github.io/galario/, which also contains code examples showing how GALARIO can be used in typical data analysis workflows.

3 VISIBILITY MODELLING

The response of a synthesis array like ALMA and the VLA to the brightness distribution of a source in the sky is a collection of measurements called *complex visibilities*. In this section we introduce the basic equations needed to define the *visibility* (a more thorough derivation can be found in Wilson, Rohlfs & Hüttemeister 2013), we illustrate how they can be implemented in a computer code, and we discuss the use cases and the limitations that follow from the adopted assumptions.

3.1 Basic equations of synthesis imaging

To define the *visibility* measurement, we first derive the response of a two-element interferometer, the fundamental receiving unit of the array. It consists of a correlator that combines, or multiplies and time averages, the signals received by the two antennas. A diagram of a two-element interferometer can be found in fig. 2-1 in Thompson (1999).

Let us introduce some definitions and a system of coordinates, following standard conventions as in Thompson (1999). Let $I_\nu(\boldsymbol{s})$ be the source brightness in direction $\boldsymbol{s}$ at frequency ν. $I_\nu(\boldsymbol{s})$ is a spectral brightness and is measured in erg s^{-1} cm^{-2} Hz^{-1} sr^{-1} or Jy sr^{-1}. Let us assume the two antennas are identical, with response pattern $A(\boldsymbol{\sigma})$ defined as the effective collecting area in direction $\boldsymbol{s}$. The radiation power collected from each of the antennas in direction $\boldsymbol{s}$ and received from the source element dΩ in the frequency range $\Delta\nu$ is then $I_\nu(\boldsymbol{s})A(\boldsymbol{s})\Delta\nu\,\mathrm{d}\Omega$. Let us call $\boldsymbol{b}$ the baseline vector connecting the two antennas on the ground and $\boldsymbol{s}$ the unit vector – identical for both the antennas – pointing towards the source. Under the simplifying assumption that the source brightness extends over a small region of the celestial sphere (Clark 1999), it is useful to rewrite $\boldsymbol{s} = \boldsymbol{s}_0 + \boldsymbol{\sigma}$, where $\boldsymbol{s}_0$ is a unit vector representing the *phase centre* of the synthesized field of view and $|\boldsymbol{\sigma}| \ll 1$. As a result, $\boldsymbol{\sigma}$, which is perpendicular to $\boldsymbol{s}_0$, lies in the plane tangent to the celestial sphere in $\boldsymbol{s}_0$.

Assuming that the source is in the far field of the interferometer (the incoming wave fronts are plane parallel) and its emission is incoherent (different parts of the source emit uncorrelated radiation), it can be shown (Clark 1999; Thompson 1999) that the response of a two-element interferometer to a source of brightness $I_\nu(\boldsymbol{s})$ is

$$V(\boldsymbol{b}) = \int_{\Omega_S} \mathcal{A}(\boldsymbol{\sigma}) I_\nu(\boldsymbol{\sigma})\, \mathrm{e}^{-2\pi \mathrm{i} \nu\, \mathbf{b}\cdot\boldsymbol{\sigma}/c} \mathrm{d}\Omega\,, \tag{1}$$

where Ω_S is the angular size of the source and $\mathcal{A}(\boldsymbol{\sigma}) = A(\boldsymbol{\sigma})/A_0$ is the normalized antenna response pattern, with A_0 being the antenna response at the centre of the beam. The central Gaussian-like

feature of the antenna pattern is usually termed primary beam and is characterized by a full width at half-maximum

$$\theta_{\rm FWHM} = K\frac{\lambda}{D}\,, \tag{2}$$

where D is the antenna diameter and k is a numerical factor close to unity. For reference, ALMA antennas have a measured $K = 1.13$ (ALMA Partnership et al. 2017). The primary-beam full width at half-maximum serves as the *field of view* of single-pointing observations. Equation (1) – derived assuming a bandwidth $\Delta\nu$ small enough so that I_ν and $\mathcal{A}$ can be considered effectively constant with ν – defines the *complex visibility* of the source with respect to the chosen phase centre $\boldsymbol{s}_0$.

In order to express equation (1) in a practical form, it is useful to define a system of coordinates such that the baseline vector $\boldsymbol{b}$ has coordinates (u, v, w) where u points towards the East, v towards the North, and w is parallel to the direction of interest (i.e. $\boldsymbol{s}_0$, the phase centre). The coordinates (u, v, w) are measured in units of the observing wavelength $\lambda = c/\nu_0$, with ν_0 measured at the centre of the bandwidth. We also introduce a coordinate system on the sky (l, m, n) with its origin in the phase centre and with (l, m, n) being the direction cosines with respect to u and v such that

$$\frac{\boldsymbol{b}\cdot\boldsymbol{s}}{\lambda} = ul + vm + wn\,. \tag{3}$$

The (l, m) plane is usually called *image* plane because it is the plane on which the source brightness $I_\nu(l, m)$ is defined. We note that inside the code the (l, m) coordinates are termed (x, y) to ease readability. With these definitions we can rewrite the complex visibility (1) as

$$V(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \frac{\mathcal{A}(l,m)I_\nu(l,m)}{\sqrt{1-l^2-m^2}} \mathrm{e}^{-2\pi\mathrm{i}(ul+vm+w\sqrt{1-l^2-m^2}-1)}\mathrm{d}l\,\mathrm{d}m. \tag{4}$$

Following Thompson (1999), for small-field imaging, i.e. $|(l^2 + m^2)w| \ll 1$, the above expression simplifies to

$$V(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathcal{A}(l,m)I_\nu(l,m)e^{-2\pi i(ul+vm)}\mathrm{d}l\,\mathrm{d}m\,, \tag{5}$$

where the ranges of the integrals have been extended to infinity since the integrand $\mathcal{A}I_\nu$ is expected to be zero for $l^2 + m^2 > 1$. Under the small-field imaging assumption, equation (5) shows that the visibility V of a source of brightness I_ν is the two-dimensional Fourier transform of its modified brightness distribution $\mathcal{A}I_\nu$.

For arrays with non-coplanar baselines ($w \neq 0$), the small-field imaging assumption introduces a phase error $\pi(l^2 + m^2)w$ for radiation coming from the (l, m) direction. Thompson (1999) and Cornwell, Golap & Bhatnagar (2008) show that this error is small in the region of the image plane centred in $(l, m) = (0, 0)$ with angular diameter

$$\theta_{\rm F} \lesssim \frac{\sqrt{\theta_{\rm res}}}{3}\,, \tag{6}$$

where $\theta_{\rm res}$ is the full width at half-maximum of the synthesized beam (expressed in radians). For a reference observation at a resolution $\theta_{\rm res} = 0.1$ arcsec, this corresponds to a region $\theta_{\rm F} \lesssim 48$ arcsec in the image plane. If the field of view of the observations (equation 2) is smaller than $\theta_{\rm F}$, then the small-field imaging assumption will be valid for single-pointing observations.

The complex-valued visibility function $V_{\rm obs}(u, v)$ is defined everywhere in the (u, v) plane but it is only measured at the discrete locations (u_k, v_k) that correspond to the projected baselines at the moment of observation. These sampling locations are usually termed uv-points. In more general terms, the visibility measurements made by the interferometer can be written as

$$V_{\rm obs}(u_k, v_k) = S\,V_{\rm obs}(u, v)\,, \tag{7}$$

where $S(u, v)$ is the visibility *sampling* function defined as

$$S(u,v) = \sum_{k=1}^{M}\delta(u-u_k, v-v_k)\,, \tag{8}$$

where δ is the Dirac delta distribution.

In order to compare a model prediction to some observed visibilities $V_{\rm obs}(u_k, v_k)$, we need to compute the synthetic visibilities of the model brightness $I_{\nu\,\rm mod}$ using equation (5):

$$V_{\rm mod}(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} \mathcal{A}(l,m)I_{\nu\,\rm mod}(l,m)e^{-2\pi i(ul+vm)}\mathrm{d}x\,\mathrm{d}y\,, \tag{9}$$

and then sample $V_{\rm mod}$ at the same uv-points where the observations were taken. The model likelihood, i.e. the probability of obtaining the observed data assuming the model is correct, can be estimated by means of a Gaussian likelihood (Pearson 1999) $\mathcal{L} \propto \exp(-\chi^2/2)$ where

$$\chi^2 = \sum_{k=1}^{M}\chi_k^2 = \sum_{k=1}^{M}|V_{\rm obs}(u_k, v_k) - V_{\rm mod}(u_k, v_k)|^2\,w_k\,, \tag{10}$$

where w_k is the weight associated with the k-th observed visibility. The weights are computed theoretically as described in Wrobel & Walker (1999) and should reflect the standard deviation σ_k of the measurements of $V(u_k, v_k)$ such that $w_k = 1/\sigma_k^2$.

3.2 Summary of the assumptions in the first release

In this section we discuss some relevant assumptions made in the first released version of the code:

(i) Small-field imaging: the first release of GALARIO uses equation (5) to compute the visibilities, thus neglecting the non-coplanarity of the baselines. This restricts the usage of the code to the cases in which the region modelled with `*Image` or `*Profile` lies within the region defined in equation (6).

(ii) Primary-beam correction: the `*Image` functions take as input an image of the primary-beam corrected brightness $\mathcal{A}I_\nu(l, m)$. In the cases in which the region of interest in the image plane is small compared to the primary beam and close to its centre, one can approximate $\mathcal{A}I_\nu \approx I_\nu$ and apply the `*Image` functions directly to the brightness without significant deviations. The choice whether to apply this approximation is left to the user. We note, however, that in the first released version of the code the `*Profile` functions – which take as input a profile $I_\nu(R)$ and internally compute $I_\nu(l, m)$ – do not apply the primary beam correction.

(iii) Frequency dependence of $\mathcal{A}$ and I_ν: both the antenna pattern and the source brightness are frequency-dependent quantities. As stated in the previous section, the definition in equation (1) holds for small bandwidths $\Delta\nu$ over which the integrand can be assumed constant. For this reason, in the first release of GALARIO, the visibilities are assumed all at the same average frequency ν_0. This implies that, in order to compare synthetic visibilities to observed ones (e.g. through equation 10 with the `chi2*` functions), the observed visibilities (typically consisting of multiple measurements over several hundreds of spectral channels) must be channel-averaged[6] into a

[6] This can be achieved, e.g. with the `split` command of the Common Astronomy Software Application (CASA) package.

single channel at frequency ν_0 and characterized by a small $\Delta\nu$. We note that the effect of channel averaging is to combine the brightness measurements over a region with angular extent $\frac{\Delta\nu}{\nu_0}\sqrt{l^2+m^2}$ along the radial direction. Often termed *bandwidth smearing*, this effect is not negligible at the distances $\sqrt{l^2+m^2}$ where its angular extent becomes comparable with the synthesized beam. The user can choose $\Delta\nu$ in order to control the bandwidth smearing within the image plane region of interest.

The computation of synthetic visibilities of a field of view with multiple sources can be done in basically two ways: either by applying `*Image` to an image of $\mathcal{A}I_\nu(l, m)$ containing all the sources, or by summing up the visibilities of each single source computed independently with either `*Image` or `*Profile`. In the second approach, the displacement of each source in the field of view can be achieved (at a small computational cost) by applying a different complex phase to the individual visibilities as described in the next section. While the first approach requires executing only one Fourier transform – appearing theoretically more computationally convenient – the second approach exploits the linearity of the Fourier transform and might yield results faster if there are many identical sources to be placed in different locations.

It is worth highlighting that in all cases (single or multiple sources in the field of view), the limitations due to the assumptions (i) to (iii) apply: all the sources must be located in a region that is close to the phase centre and small compared to θ_F and the synthetic visibilities are computed in a narrow band around the observing frequency ν_0.

3.3 Image translation and rotation

The `*Profile` and `*Image` functions enable the user to apply a translation and a rotation with respect to the phase centre to the model image by specifying the optional parameters `dRA`, `dDec`, and `PA`. This functionality can be useful, e.g. to fit the centre and the Position Angle of a model image to the observations. Instead of translating and rotating the model image before taking the Fourier transform, GALARIO exploits the symmetries of the Fourier transform under these geometric operations to achieve a better performance and accuracy (Briggs et al. 1999).

To perform the rotation, we use the fact that the Fourier transform commutes with rotations. This implies that to compute the visibilities of a model $I_{\nu\,\mathrm{mod}}$ rotated by an angle PA, it is sufficient to rotate the coordinates of the uv-points by $-PA$ with

$$u'_k = u_k \cos(PA) - v_k \sin(PA), \tag{11}$$

$$v'_k = u_k \sin(PA) + v_k \cos(PA), \tag{12}$$

where u'_k and v'_k are the rotated coordinates of the k-th uv-point.

The translation of the model image is obtained by multiplying the sampled visibilities $V_\mathrm{mod}(u_k, v_k)$ by a complex phase, rather than by interpolating the image on a shifted spatial grid. This is based on the behaviour of the Fourier transform with respect to translations, according to which

$$\mathfrak{F}\, g(x - \Delta x) = \mathfrak{F}\, g(x) \times \mathrm{e}^{-2\pi \mathrm{i} u \Delta x}\,, \tag{13}$$

where $\mathfrak{F}$ denotes the Fourier transform operation. By multiplying the sampled visibilities $V_\mathrm{mod}(u_k,\ v_k)$ by a phase $\exp[\,-2\pi\mathrm{i}(u\Delta\alpha + v\Delta\delta)]$ with u, v measured in units of wavelength and $\Delta\alpha, \Delta\delta$ measured in radians, it is possible to apply the desired shift in the image plane.

3.4 Requirements on the image

To compute the Fourier transform in equation (5) GALARIO uses the fast Fourier transform algorithm (FFT) (Cooley & Tukey 1965) that requires a regularly spaced 2D image as input. In this section we describe the constraints on the image size and the pixel size that should be fulfilled for a correct computation of the complex visibilities. We note that such constraints are jointly determined by the distribution of uv-points (which sets the resolution and the maximum recoverable scale of the observations), by the diameter of the antennas (which sets the primary beam), and by the size and the location of the sources (which set the portion of the image plane of interest). For the clarity of the exposition, the considerations that follow are derived assuming a single source in a single-pointing observation. The generalization for multiple sources is at the end of this section.

Let us call N_l and N_m the number of pixels in the l and m direction, respectively, of the input matrix containing $\mathcal{A}I_\nu(l, m)$. The origin $(l, m) = (0, 0)$ is located at the image centre. If $\Delta\theta_l$ and $\Delta\theta_m$ are the angular pixel sizes in each direction, the input matrix covers a rectangular region in the image plane defined by

$$|l| \leq \frac{N_l \Delta\theta_l}{2} \quad \text{and} \quad |m| \leq \frac{N_m \Delta\theta_m}{2}\,. \tag{14}$$

In an analogous way, we can introduce the pixel size in the uv-plane Δu and Δv, in the u and v direction, respectively. The region of the uv-plane covered by the output matrix of the FFT algorithm is thus defined by

$$|u| \leq \frac{N_l \Delta u}{2} \quad \text{and} \quad |v| \leq \frac{N_m \Delta v}{2}\,. \tag{15}$$

There is a correspondence between the pixel size in the image plane and that in the uv-plane, given by

$$N_l \Delta\theta_l = \frac{1}{\Delta u} \quad \text{and} \quad N_m \Delta\theta_m = \frac{1}{\Delta v}\,. \tag{16}$$

In the remainder of this discussion, let us assume square pixels both in the image plane and in the uv-plane:

$$\Delta\theta_l = \Delta\theta_m \equiv \Delta\theta_{lm} \quad \text{and} \quad \Delta u = \Delta v \equiv \Delta uv\,. \tag{17}$$

This is a choice that is usually made and it is also assumed inside GALARIO. For the present discussion let us also assume for simplicity that the input matrix is square; i.e.

$$N_l = N_m \equiv N_{lm}\,. \tag{18}$$

The distribution of uv-points where the synthetic visibilities have to be computed imposes two fundamental constraints on the values of N_{lm}, $\Delta\theta_{lm}$, and Δuv:

(i) the region of the uv-plane that is modelled must encompass the region sampled by the uv-points, exceeding the most extended baseline by at least a factor of two in order to fulfil Nyquist sampling, that is

$$\frac{N_{lm}\Delta uv}{2} = \max_k\{(u_k^2 + v_k^2)^{1/2}\} \cdot f_\mathrm{max} \quad \text{with} \quad f_\mathrm{max} > 2\,, \tag{19}$$

where the maximum is taken over all the baselines represented by the given uv-points.

(iia) the region of the image plane that is modelled must be *at least* larger than the maximum recoverable scale θ_MRS, namely:

$$N_{lm}\Delta\theta_{lm} > \theta_\mathrm{MRS} \equiv \frac{\Gamma}{\min_k\{(u_k^2 + v_k^2)^{1/2}\}}\,, \tag{20}$$

where $\Gamma \approx 0.5$ is a constant. For reference, ALMA has $\Gamma = 0.6$ (cf. equation 3.27 in ALMA Partnership et al. 2017). Using

equation (16) we can rewrite equation (20) as a constraint on the uv cell size:

$$\Delta uv = \frac{1}{\Gamma\, f_{\rm min}} \cdot \min_k \left\{ \left(u_k^2 + v_k^2\right)^{1/2} \right\} \quad \text{with} \quad f_{\rm min} > 1\,, \tag{21}$$

and thus compute the image size:

$$N_{lm} = 2\Gamma\, f_{\rm min} \frac{\max_k\{(u_k^2 + v_k^2)^{1/2}\}}{\min_k\{(u_k^2 + v_k^2)^{1/2}\}}\,. \tag{22}$$

A conservative choice for $f_{\rm min}$ would be $f_{\rm min} = 5$ to ensure that the field of view of the input matrix encompasses at least by five times the scale of the largest sources that might be resolved in the data.

The most conservative criterion for the choice of Δuv consists of imaging the whole field of view covered by the observations:

(iib) the region of the image plane that is modelled must be as large as the primary beam, namely:

$$N_{lm}\Delta\theta_{lm} = \theta_{\rm FWHM}\,. \tag{23}$$

In this case, the image size is given by

$$N_{lm} = 2\frac{D}{K\lambda}\, \max_k\{(u_k^2 + v_k^2)^{1/2}\}\,, \tag{24}$$

which typically yields much larger N_{lm} than equation (22).

Given a distribution of uv-points (u_k, v_k), these criteria allow one to compute N_{lm} and $\Delta\theta_{lm}$ that should be used for the input image. These criteria are implemented in `get_image_size`, which can be used as

$N_{lm},\ d_{lm} =$ `get_image_size(u, v, PB=`$\theta_{\rm FWHM}$`)`

By default `get_image_size` uses criterion (iia) and equation (22). If the primary beam FWHM is specified as the optional parameter `PB`, then `get_image_size` uses equation (24) in criterion (iib).

In case the field of view contains multiple sources, criterion (iib) (instead of iia) should be used to ensure that the sources are correctly represented in the image plane. In any case, N_{lm} should always be large enough so that the sources are far from the edges of the image. Finally, we note that N_{lm} is ultimately limited by the assumptions discussed in Section 3.2.

Table 1 shows a compilation of matrix properties derived for realistic ALMA and VLA array configurations. For each configuration we report the nominal minimum and maximum baseline, N_{lm} computed using both criteria (iia) and (iib), $\Delta\theta_{lm}$ and the resolution $\theta_{\rm res}$. $\Delta\theta_{lm}$ and $\theta_{\rm res}$ depend on the observing wavelength, for which we assumed representative values of $\lambda = 1.3$ mm for ALMA and $\lambda = 7.0$ mm for the VLA. In creating the table, we computed $\theta_{\rm res} = (\text{Max Baseline}\,\lambda^{-1}\Delta uv)^{-1}$, which is an ideal estimate that assumes a natural weighting scheme and neglects inhomogeneities in the baseline distribution; real values depend on the actual distribution of the antennas and differ at most by 15 per cent (cf. ALMA Partnership et al. 2017). We notice that the typical matrix sizes requested to cover the MRS by at least a factor of five ($f_{\rm min} = 5$) range between 256^2 and 4096^2 for ALMA and between 512^2 and 2048^2 for the VLA; much larger matrix sizes (up to 16384^2) are needed to cover the full primary beam (we caveat that the image sizes N_{lm} (iib) reported for the VLA A and B configurations exceed the small field imaging assumption). In all cases the values of $\Delta\theta_{lm}$ are comfortably smaller than the synthesized beam $\theta_{\rm res}$ by 5–10 times.

For best performances in the FFT computation, it is advisable to use matrices with N_{lm} that is a power of two.

The last step in the visibilities computation requires sampling the matrix containing $V(u, v)$ at the discrete locations (u_k, v_k), as described by equation (7). This operation can be done either by convolving $V(u, v)$ with a carefully chosen kernel and then by sampling the result at the centre of each grid cell (Schwab 1984; Briggs et al. 1999), or by means of interpolation. GALARIO performs the sampling using a bilinear interpolation algorithm (Press et al. 2007); i.e. by inferring the value $V_{\rm mod}(u_k, v_k)$ from the value of $V(u, v)$ in the four closest grid points, assuming linear increments in both directions.

Table 1. Matrix and pixel sizes for ALMA and VLA configurations.

Array Config.	Baselines Min (m)	Baselines Max (m)	Matrix properties N_{lm} (iia) (px)	Matrix properties N_{lm} (iib) (px)	$\Delta\theta_{lm}$ (arcsec)	$\theta_{\rm res}$ (arcsec)
			ALMA			
C43-1	14.6	160.7	256	256	0.215	1.669
C43-2	14.6	313.7	512	512	0.108	0.855
C43-3	14.6	500.2	1024	1024	0.054	0.536
C43-4	14.6	783.5	1024	1024	0.054	0.342
C43-5	14.6	1397.9	2048	2048	0.027	0.192
C43-6	14.6	2516.9	4096	4096	0.013	0.107
C43-7	64.0	3637.8	1024	4096	0.012	0.074
C43-8	110.4	8547.7	2048	8192	0.004	0.031
C43-9	367.6	13894.2	1024	16384	0.002	0.019
C43-10	244.0	16194.0	1024	16384	0.003	0.017
			VLA			
A	680.0	36400.0	1024	16384	0.006	0.040
B	210.0	11100.0	1024	4096	0.020	0.130
C	35.0	3400.0	2048	2048	0.060	0.425
D	35.0	1030.0	512	512	0.242	1.402

Note. The baselines are taken from the ALMA Cycle 5 Technical Handbook and the VLA 2018A Call for proposal. N_{lm} (iia) and N_{lm} (iib) have been computed using criteria (ii) and (iii) in equation (22) and equation (24), respectively. We used $f_{\rm max} = 2.5$ and $f_{\rm min} = 5$. $\Delta\theta_{lm}$ and $\theta_{\rm res}$ have been computed assuming representative values $\lambda = 1.3$ mm for ALMA and $\lambda = 7.0$ mm for the VLA. We caveat that the image sizes N_{lm} (iib) for the VLA A and B configurations exceed the maximum allowed by the small-field assumption.

4 IMPLEMENTATION

The basic purpose of GALARIO is to compute synthetic visibilities at a set of points in the uv-plane as illustrated in Fig. 1. To achieve this, a number of operations have to be carried out. In Fig. 3 we show the relevant operations that are common to CPU and GPU as a flow chart in order to compute the visibilities ($V_{\rm mod}$) and the χ^2. The functions `Chi2Image` and `Chi2Profile` only differ in the first stage, where the input image is either taken as is or created from a radial profile. The next steps before the χ^2 reduction create the visibilities. If the users wishes to use these directly, perhaps in a more sophisticated analysis than a χ^2 fit, then `sampleImage` and `sampleProfile` would return at that point.

All operations shown in Fig. 3 have a multithreaded CPU and GPU implementation. We wrote the code in `C++` and parallelized custom kernels in NVIDIA CUDA (2017) on the GPU, and with the help of OpenMP (Dagum & Menon 1998) on the CPU. For custom kernels, we used common inline functions to inject the core operations into surrounding code that differs on CPU and GPU because of memory handling or available libraries. GPU kernels use grid-stride loops when applicable. Whenever possible, we prefer optimized library functions instead of custom kernels. The FFT is performed by `FFTW` (Frigo & Johnson 2005) or `cuFFT` (NVIDIA cuFFT 2017). We use `cuBLAS` (NVIDIA cuBLAS 2017) for the χ^2 reduction on the GPU.

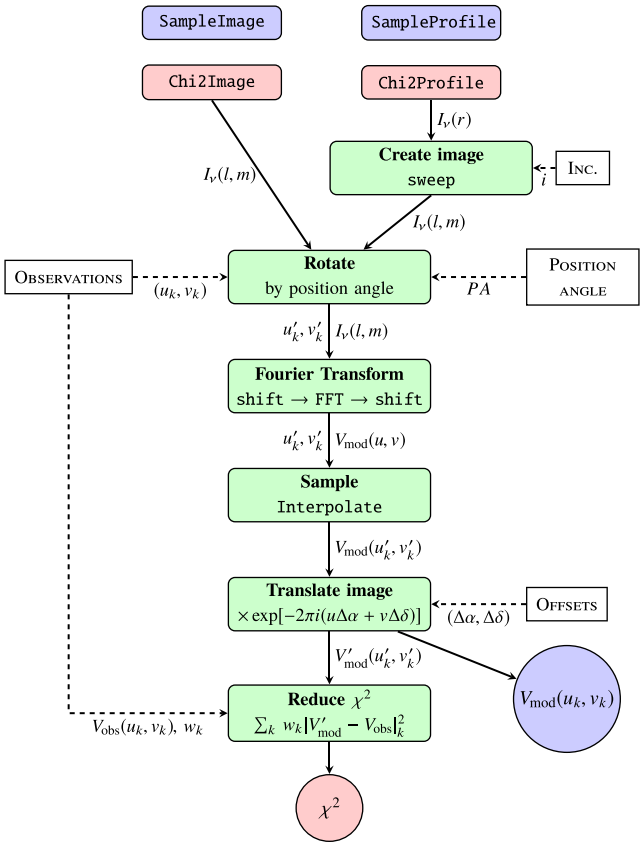

Figure 3. Flow chart of the algorithm, proceeding from top to bottom. White boxes indicate inputs, green boxes represent operations involving one or more parallel regions that can run on the GPU or the CPU. Circles indicate outputs and are colour-coded as the functions from which they are produced: red for the `chi2*` functions and the purple for the `sample*` functions. Arrows indicate data flow between kernels.

To simplify the flow chart, we omit the memory operations because they are quite different on CPU and GPU. We assume that, prior to calling GALARIO, observations and all other inputs are initially in the CPU main memory. To use the GPU, these data have to be transferred and this can take a significant fraction of the overall execution time whereas the transfer is unnecessary when computing on the CPU; see Section 6 for details. In the special case of an axisymmetric brightness profile, we can exploit the symmetry of the image to avoid unnecessary data transfer: we supply the `*Profile` functions that only copy a radial profile defined on sky coordinates and create the image directly in the GPU memory through the `sweep` function, which essentially rotates the profile to sweep the 2D image over 2π, performing bilinear interpolation as in Press et al. (2007). The purpose of the `Chi2*` function is to avoid transferring the sampled visibilities back from the GPU.

In the typical use case, an input image is such that the origin of the coordinate system is in the central pixel. But `FFTW` and `cuFFT` expect the origin in the top-left pixel. So we copy or create the input image in a buffer and perform the shift, the FFT, and the inverse shift in place. The shift algorithm is similar to the one by Abdellah (2014) but was independently devised. The input image is real which saves a factor of two in both memory and computing effort in the FFT compared to a complex-to-complex transform.

All operations of Fig. 3 are accessible separately, which greatly help with unit testing. We build up an extensive suite of tests using `pytest` that verifies the individual operations and their various combinations. To improve the speed of GALARIO, we used graphical profilers such as Nvidia `nvvp` and Intel `Amplifier` as well as custom timing methods to continuously monitor the performance in a more automated fashion.

5 ACCURACY

In this section we report the results of the tests that we conducted to check the accuracy of GALARIO against analytic results. In Appendix C we report additional accuracy checks performed against the NRAO CASA package for input images that do not necessarily have analytic visibility expressions.

To check the accuracy of GALARIO against analytic results, we use the fact that the synthetic visibilities of an axisymmetric brightness profile $I_\nu(r)$ centred at the origin of the image plane have an analytic result:

$$V(\rho) = 2\pi \int_0^\infty I_\nu(r)\, J_0(2\pi\rho r)\, r\, \mathrm{d}r\,, \tag{25}$$

where $\rho = \sqrt{u^2 + v^2}$ is the deprojected uv-baseline, r is the angular distance from the centre, and J_0 is the 0-th order Bessel function of the first kind (Pearson 1999). For example, this approach has been recently used by Zhang et al. (2016) to compare different brightness profiles to interferometric observations of protoplanetary discs.

Using equation (25) we compute analytical synthetic visibilities of four brightness-profile templates with different features and we compare them to the visibilities output by the `sample*` functions.

(a) a Gaussian disc with a Gaussian ring-like excess:

$$I_\nu(r) = \exp\left[-\left(\frac{r}{0.2\,\mathrm{arcsec}}\right)^2\right] + 0.3\,\exp\left[-\left(\frac{r - 0.4\,\mathrm{arcsec}}{0.15\,\mathrm{arcsec}}\right)^2\right], \tag{26}$$

(b) a smooth Gaussian ring:

$$I_\nu(r) = \exp\left[-\left(\frac{r - 0.5\,\mathrm{arcsec}}{0.1\,\mathrm{arcsec}}\right)^2\right], \tag{27}$$

(c) a sharp rectangular ring:

$$I_\nu(r) = \begin{cases} 1 & \text{for } 0.2\,\mathrm{arcsec} \le r \le 0.5\,\mathrm{arcsec} \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

(d) three Gaussian rings:

$$\begin{aligned} I_\nu(r) = {} & \exp\left[-\left(\frac{r - 0.2\,\mathrm{arcsec}}{0.1\,\mathrm{arcsec}}\right)^2\right] \\ & + 0.7\,\exp\left[-\left(\frac{r - 0.5\,\mathrm{arcsec}}{0.05\,\mathrm{arcsec}}\right)^2\right] \\ & + 0.2\,\exp\left[-\left(\frac{r - 0.7\,\mathrm{arcsec}}{0.03\,\mathrm{arcsec}}\right)^2\right]. \end{aligned} \tag{29}$$

The choice of these templates with smooth or sharp, small or large spatial features aims at reproducing typical brightness profiles that are used to fit real observations and also to check how well GALARIO samples the different spatial frequencies that characterize their visibility profiles.

The visibilities are computed at uv-points with baselines between 10 and 1000 kλ. For reference, ALMA 1.3 mm observations in C43-5 configuration achieve a similar uv coverage. The results presented below hold for different baseline extents and uv-point locations; in Fig. 4 we show just one of the several different configurations we tested. For each of the templates we plot the radial

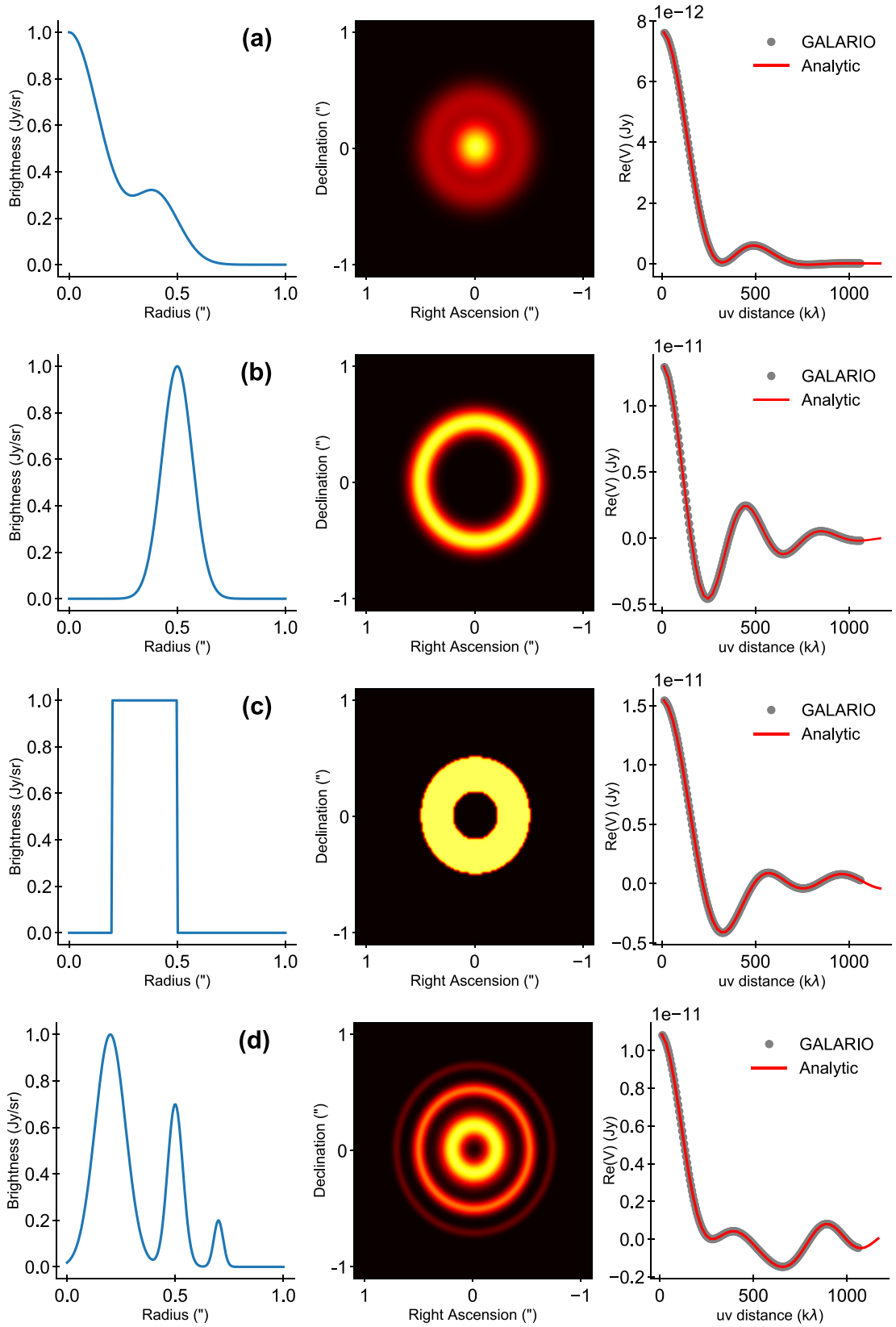

Figure 4. Results of the accuracy checks. For each of the templates we plot the radial brightness profile $I_\nu(r)$ (left-hand panel), the image of the model (central panel), and the comparison of the synthetic visibilities (right-hand panel). The synthetic visibilities computed analytically (red lines) are compared to those computed by GALARIO (grey dots). The analytic synthetic visibilities have been sampled exactly in the same *uv* locations of those computed by GALARIO but we show them as a continuous red line to aid the visual comparison.

Table 2. Execution times of `chi2Profile`.

N_{lm} (px)	Python (ms)	Serial (ms)	CPU 6 threads (ms)	CPU 12 threads (ms)	GPU (ms)
512	815	109	19	33	12
1024	1407	120	22	16	14
2048	2719	175	38	31	18
4096	5959	478	115	95	39
8192	14702	1440	358	317	126
16384	41895	6204	1536	1411	479

Note. Timings refer to the execution of the double-precision version of `chi2Profile` with $M = 10^6$ visibility points.

brightness profile $I_\nu(r)$ (left-hand panel), the image of the model (central panel), and the comparison of the synthetic visibilities (right-hand panel). The `sampleProfile` and `sampleImage` functions yield identical synthetic visibilities at machine precision level, therefore in Fig. 4 we just show the results for one of them, `sampleProfile`. Only the real part $Re(V)$ of the synthetic visibilities is shown, since the imaginary part is identically zero for axisymmetric input images.

In general we observe a very good agreement between the synthetic visibilities computed by GALARIO and those computed analytically with equation (25), as the deprojected visibility profiles in Fig. 4 clearly show. `sampleProfile` and `sampleImage` model correctly the visibility profile of the templates at all the spatial frequencies. We performed quantitative checks on the discrepancy between the results and we find that the fractional difference between the sampled $Re(V)$ values is generally smaller than 10^{-5}. Only for a few data points where $Re(V)$ is very close to zero does the fractional difference reach a level of 0.1 per cent. We conducted numerous other consistency checks during the development of GALARIO that we do not report here – e.g. comparing the output of the complex-to-complex Fourier transform with respect to the real-to-complex one, etc. – but all are available as unit tests and can be executed from GALARIO's source code.

6 PERFORMANCE

We now investigate the performance characteristics of GALARIO. All experiments shown are performed on a desktop workstation with an Intel i7-6800K CPU with six cores on one socket, hyperthreading, 3.4 GHz maximum frequency and 32 GB of RAM. The machine also has an Nvidia GTX 1060 graphics card with 6 GB of RAM and 1280 CUDA cores. We also ran identical benchmarks on high-performance systems with 32 CPU cores and more powerful Nvidia P100 GPUs. While the exact timings differed, we verified that our qualitative conclusions presented below also hold on these much more expensive systems.

All results are for double precision only. We observed significant loss of precision in the single-precision FFT for reasonably sized images beyond 512^2 pixels that could affect scientific results whereas the double-precision FFT was much more robust. Therefore we recommend double precision as the default mode in GALARIO.

6.1 Scaling with image size

To justify the effort of creating this package, we consider an alternative implementation of `chi2Profile` in standard PYTHON without any explicit loops, using instead the widespread NUMPY and SCIPY packages (van der Walt, Colbert & Varoquaux 2011) to do all the 'heavy lifting'. This represents a baseline solution that could be assembled in a short time without requiring deep thought. This PYTHON version is shipped with GALARIO's unit tests and is reported for completeness in Appendix D.

In Fig. 5 we show the scaling behaviour by calling GALARIO's `chi2Profile` double-precision implementation for different sizes of the input image varying from 512^2 to 16384^2 pixels. This is repeated on the CPU with 1, 6, and 12 OpenMP threads and on the GPU. The absolute timings are reported in Table 2, while Fig. 5 presents the timings in terms of the speed-up relative to the PYTHON-only baseline. Contrary to common belief, even the serial CPU implementation of GALARIO is significantly faster than the baseline, thus implying that there is a price to pay when relying on NUMPY and SCIPY even though the relevant parts also execute compiled C code just like GALARIO.

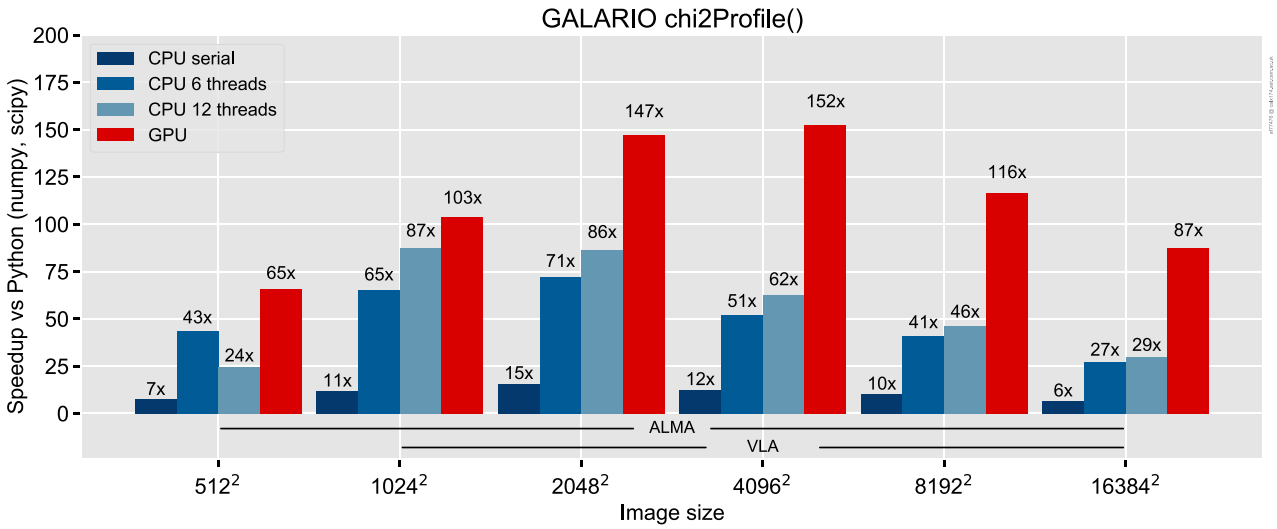

Figure 5. Scaling with image size: speed-up of the CPU and GPU version of `chi2Profile`. The CPU version is executed with 1, 6, and 12 threads. The speed-up is computed with respect to a PYTHON version that relies on NUMPY and SCIPY and make no use of explicit loops. The absolute execution times are reported in Table 2. The horizontal brackets highlight typical matrix sizes for realistic ALMA and VLA array configurations (cf. Table 1).

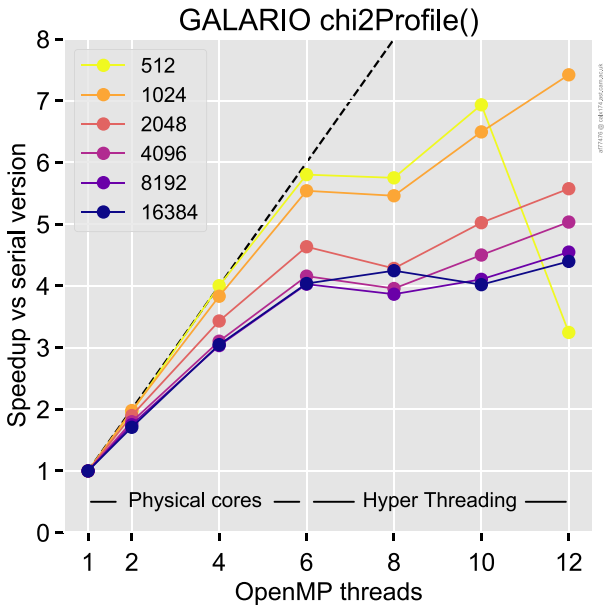

Figure 6. Strong scaling of the CPU version for different image sizes: speed-up (with respect to the serial version) for increasing computing power.

It turns out that because of the large amount of memory accesses inherent to our algorithm, hyperthreading can help significantly. The biggest gain is 30 per cent for a 1024^2 image comparing 6–12 threads; i.e. one to two threads per physical core. In contrast, for the smallest image size 512^2, the overhead of threads leads to a performance penalty of about 50 per cent. It appears the optimum number of threads has to be determined by trial and error.

Comparing the fastest CPU timing to the GPU timing, we observe that executing on the GPU is about 30 per cent to three times faster. A speed-up of two to five is often observed when comparing optimized parallel implementations on CPU and GPU, and bigger speed-ups occur when the baseline is serial or unoptimized CPU code (Lee et al. 2010). The advantage of the GPU is the enormous number of threads that can operate simultaneously, so it performs best if there are many arithmetic operations per data unit. The disadvantage is that data transfer from the CPU to the GPU and memory allocation on the GPU are much slower compared to the CPU. The ideal application for the GPU then is for a large image that is created on the GPU and need not be transferred from host memory.

6.2 Strong scaling

Taking the serial CPU code for one OpenMP thread as the baseline, Fig. 6 shows the strong-scaling behaviour; i.e. by how much the execution improves with more threads for a fixed image size. We compute `chi2Profile` 300 times with identical input parameters for each number of threads, and display the shortest of the 300 times recorded.

For small images up to 1024^2 pixels, the speed-up is nearly equal to the number of threads until it reaches six, the number of physical cores on our test machine. For larger images, the cache size becomes a factor as threads compete for it and we have many memory-heavy operations. The improvement up to six threads is still monotonous. Using more threads than cores slightly hides the cache misses, so for all image sizes except 512^2, the highest performance is attained for 12 threads, i.e. two threads per core, the maximum supported by native hyperthreading.

6.3 Profiling suboperations

While GALARIO aims to be user friendly and accepts an input image supplied by the user with `chi2Image`, it is generally advantageous to create the image on the GPU. For the particular case of an image created from a radial profile, `chi2Profile` only transfers a radial profile equivalent to one row of pixels to the GPU, creates the image on the GPU, then performs the same operations on the image as `chi2Image`. In Fig. 7, we show a detailed break-

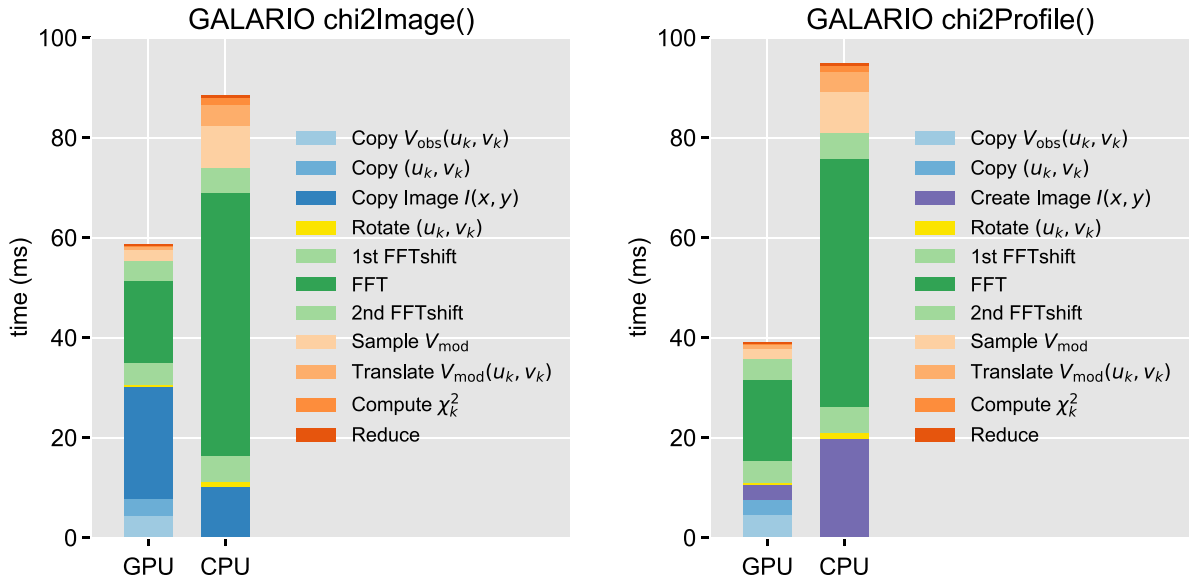

Figure 7. Execution times of the suboperations in `chi2Image` (left-hand panel) and `chi2Profile` (right-hand panel). Each vertical bar represents the smallest time across 20 calls of `chi2*` for a 4096^2 image. The CPU version is run with the optimal configuration of threads for this image size. CPU to GPU (Host to Device) copy operations are coloured in tones of blue. On the GPU, `chi2Profile` dramatically reduces the time spent in copying with respect to `chi2Image` and, even accounting for the extra-time needed to create the image, it manages to be 1.5 times faster than `chi2Image`.

down of the time that each of the suboperations requires for both `chi2*` functions for a 4096^2 image. We compare only the optimal number of threads on the CPU (12 in this case) to the GPU implementation. We repeated each function call 20 times and display the minimum time for each suboperation during the 20 runs. Since no run features the minimum time for all suboperations, the minimum time for `chi2*` across the 20 runs is slightly larger than the sum of timings shown in Fig. 7.

The important point of Fig. 7 is that all compute-intensive operations are faster on the GPU compared to the CPU, in particular the Fourier transform and the creation of the image. But the data transfers partly reduce this advantage, either because they are not needed at all on the CPU (for example the copy of read-only data like observations) or because the PCI express bus has a much reduced bandwidth and higher latency compared to accesses to the main CPU memory.

For a large image, copying to the device actually takes longer than the FFT, that is why `chi2Image` is nearly 20 ms, or 50 per cent, slower than `chi2Profile` on the GPU, and that does not even account for the time needed to create the image on the CPU, which in this example takes another 20 ms. On the CPU, `chi2Profile` is only about 15 per cent faster than `chi2Image`.

7 CONCLUSIONS

In this paper we have presented GALARIO, a GPU accelerated library for analysing radio interferometer observations. Distributed under the open source GNU LGPLv3, GALARIO is actively developed at http://github.com/mtazzari/galario and can be easily installed on machines with different configurations.

Unlike single dishes, which directly measure the brightness of a source over a continuous region of the sky, radio interferometers measure its Fourier transform, sampling it at discrete locations and producing a collection of complex visibilities. The computational effort required to compare a model prediction to an observational data set of complex visibilities has increased dramatically in the last years due to the improved angular resolution and data rate of modern radio interferometers, which require larger matrix sizes and involve hundreds of thousands of visibility points.

The process of computing synthetic visibilities from a model brightness involves several time-consuming matrix operations such as Fourier transforms, quadrant swaps, and interpolations. These operations have to be performed once for every likelihood evaluation, and the likelihood is called thousands or even a million times in the normal workflow required to fit a model to the data.

In this context, GALARIO leverages the computing power of modern GPUs to accelerate the computation of the synthetic visibilities, thus reducing the overall execution time of the likelihood. For ease of use, GALARIO offers dedicated functions that produce directly the weighted χ^2 of the model for the given observations. Such functions can be easily included in any analysis scheme, be it a Markov Chain Monte Carlo sampler or a classical χ^2 optimizer. Moreover, thanks to its modularity, GALARIO can be used to fit simultaneously several observations at different wavelengths, thus speeding up even the most demanding multiwavelength analyses.

GALARIO is easy to use – computing the synthetic visibilities from a model image can be done in one line of code – and easy to adopt in existing code – PYTHON wrappers to the underlying C and CUDA code are available for all the functions. The design of GALARIO, with symmetric CPU and GPU versions of all the functions, allows the user to develop highly reusable code that can be executed both on CPUs and on GPUs with minimal changes, ensuring considerable speed-ups also on machines without a GPU.

In terms of performances, GALARIO is faster than a standard PYTHON implementation of the same functionalities by up to 150 times on the GPU and up to 90 times on the CPU. We note that these speed-ups are achievable not only on top-tier GPUs, but also on affordable desktop-class ones.

In the future releases of GALARIO we plan to generalize the implementation of the synthesis imaging equations by including the primary beam correction and a proper treatment of non-coplanar baselines (relevant for wide field imaging). Moreover, several new features will be added, including the multiwavelength synthesis of brightness models with spectral dependence.

ACKNOWLEDGEMENTS

The authors thank the anonymous referee for a constructive report that helped improving the clarity of the paper in many points. MT is grateful to Padelis P. Papadopoulos, Attila Juhasz, Luca Matrà, and Sebastian Marino for the numerous insightful discussions on radio interferometry. MT has been supported by the DISCSIM project, grant agreement 341137 funded by the European Research Council under ERC-2013-ADG. FB and LT acknowledge support by the DFG cluster of excellence Origin and Structure of the Universe (www.universe-cluster.de). The initial development of GALARIO was boosted by the GPU Hackathon at the TU Dresden in 2016 February thanks to two supportive mentors, Thorsten Hater and Andreas Herten. The GPU Hackathon is a collaboration between and used resources of both TU Dresden and the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory. Oak Ridge National Laboratory is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Some development and testing of the GPU implementation has taken place on GPU machines by the Leibniz Supercomputing Center's data lab. Testing has also been carried out on the Hydra computing cluster by the Max Planck Gesellschaft, with support by Paola Caselli. GALARIO 1.0 has the following Zenodo reference https://doi.org/10.5281/zenodo.889991.

REFERENCES

Abdellah M., 2014, Proc. High Performance Comput. Symp. HPC'14. Society for Computer Simulation International, San Diego, CA, p. 5:1
ALMA Partnership, 2017, ALMA Cycle 5 Technical Handbook. Vol. 5.3 ver. 1.0
Briggs D. S., Schwab F. R., Sramek R. A., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 127
Clark B. G., 1980, A&A, 89, 377
Clark B. G., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 1
Cooley J. W., Tukey J. W., 1965, 19, 297
Cornwell T. J., Evans K. F., 1985, A&A, 143, 77
Cornwell T., Braun R., Briggs D. S., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 151
Cornwell T. J., Golap K., Bhatnagar S., 2008, IEEE J. Sel. Top. Signal Process., 2, 647
Dagum L., Menon R., 1998, IEEE Comput. Sci. Eng., 5, 46
Fedele D. et al., 2018, A&A, 610, A24
Frigo M., Johnson S. G., 2005, Proc. IEEE, 93, 216
Högbom J. A., 1974, A&AS, 15, 417

Lee V. W. et al., 2010, ACM SIGARCH Computer Architecture News, Vol. 38, p. 451
Martí-Vidal I., Vlemmings W. H. T., Muller S., Casey S., 2014, A&A, 563, A136
Nickolls J., Buck I., Garland M., Skadron K., 2008, Queue, 6, 40
NVIDIA cuBLAS 2017, CUDA basic linear algebra subprograms. NVIDIA Corporation. Available at: http://docs.nvidia.com/cuda/cublas/
NVIDIA CUDA 2017, Compute Unified Device Architecture Programming Guide. NVIDIA Corporation. Available at: http://docs.nvidia.com/cuda
NVIDIA cuFFT 2017, CUDA Fast Fourier Transform. NVIDIA Corporation. Available at: http://docs.nvidia.com/cuda/cufft/
Pearson T. J., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 335
Perkins S. J., Marais P. C., Zwart J. T. L., Natarajan I., Tasse C., Smirnov O., 2015, Astron. Comput., 12, 73
Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P., 2007, Numerical Recipes: The Art of Scientific Computing. Cambridge Univ. Press, Cambridge, p. 1
Sault R. J., Teuben P. J., Wright M. C. H., 1995, in Shaw R. A., Payne H. E., Hayes J. J. E., eds, ASP Conf. Ser. Vol. 77, Astronomical Data Analysis Software and Systems IV. Astron. Soc. Pac., San Francisco, p. 433
Schwab F. R., 1984, in Roberts J. A., ed., Indirect Imaging. Measurement and Processing for Indirect Imaging. p. 333
Tazzari M. et al., 2016, A&A, 588, A53
Tazzari M. et al., 2017, A&A, 606, A88
Testi L., Natta A., Scholz A., Tazzari M., Ricci L., de Gregorio Monsalvo I., 2016, A&A, 593, A111
Thompson A. R., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 11
van der Walt S., Colbert S. C., Varoquaux G., 2011, Comput. Sci. Eng., 13, 22
Wilson T. L., Rohlfs K., Hüttemeister S., 2013, Tools of Radio Astronomy. Springer, Berlin, Heidelberg
Wrobel J. M., Walker R. C., 1999, in Taylor G. B., Carilli C. L., Perley R. A., eds, ASP Conf. Ser. Vol. 180, Synthesis Imaging in Radio Astronomy II. Astron. Soc. Pac., San Francisco, p. 171
Zhang K., Bergin E. A., Blake G. A., Cleeves L. I., Hogerheijde M., Salinas V., Schwarz K. R., 2016, ApJ, 818, L16

APPENDIX A: INSTALLATION

GALARIO is actively developed online at http://github.com/mtazzari/galario. Contributions are welcome and we invite users that encounter problems in using GALARIO to report them at https://github.com/mtazzari/galario/issues.

The easiest way to install GALARIO is via CONDA, the package manager of the Anaconda PYTHON distribution,[7] which ensures all the dependencies are installed automatically. With CONDA, the user gets access to GALARIO C/C++ and PYTHON bindings, both with support for multithreading. The installation command is as easy as

```
conda install -c conda-forge galario
```

Due to technical limitations, the CONDA package does not support GPUs at the time of writing. In order to use the GPU version, GALARIO must be compiled by hand as follows. First, download the latest stable version from the repository with

```
git clone <t>https://github.com/mtazzari/
galario.git
```

GALARIO works with both PYTHON 2 and 3, and to simplify the build we suggest to work in a PYTHON virtual environment. Instructions on how to create an environment are reported in the online documentation.

Once downloaded, GALARIO can be installed with:

```
cd galario
mkdir build && cd build
cmake.. && make
```

which will compile the CPU version of GALARIO and, if a GPU is present on your machine, also the GPU version. The `cmake` command takes care of adapting the compilation instructions to the compilers and the libraries available on your machine.

Once compiled, GALARIO can be installed with

```
sudo make install
```

or, in the case the user has no root privileges, an installation path can be specified with

```
cmake -DCMAKE_INSTALL_PREFIX=/path/to/
galario/..make install
```

This installs the C libraries of GALARIO in `path/to/galario/lib` and the PYTHON libraries in the currently active Python environment.

A full list of system requirements and detailed instructions to compile GALARIO on different systems are available in the online documentation at https://mtazzari.github.io/galario/.

APPENDIX B: PERFORMANCE (CONTINUED)

In Section 6 we presented the performance of `sampleProfile`. For completeness, in this appendix we report analogous performance measurements conducted for `sampleImage`. Fig. B1 shows the scaling of the CPU and GPU version as a function of matrix size, while Fig. B2 shows the scaling of the CPU version for increasing computing power. The absolute time measurements are reported in Table B1.

[7] https://www.anaconda.com/

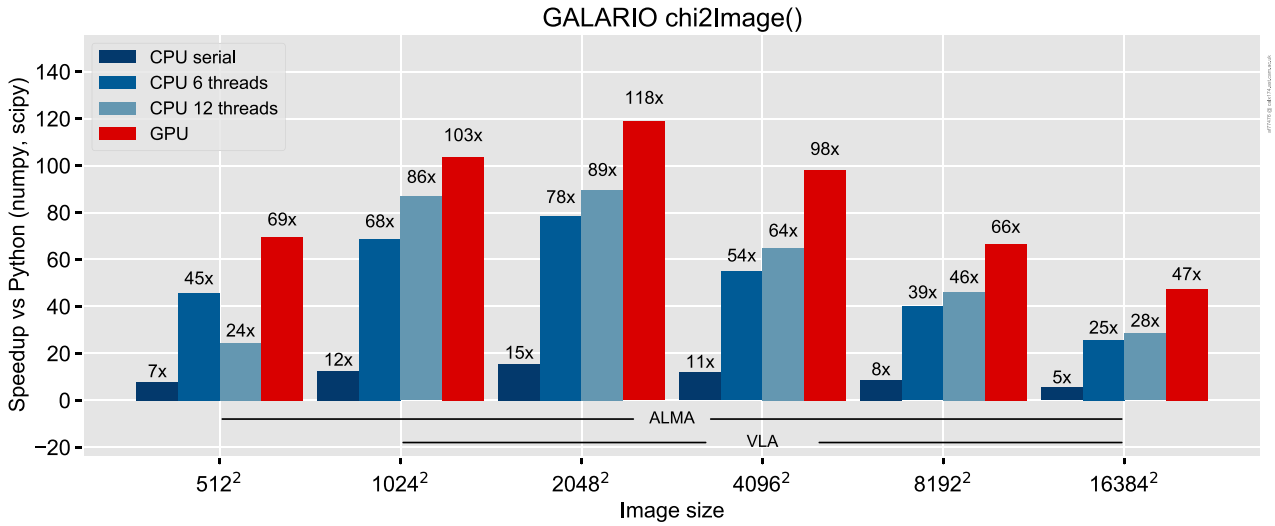

Figure B1. Scaling with image size: speed-up of the CPU and GPU version of `chi2Image`. The CPU version is executed with 1, 6, and 12 threads. The speed-up is computed with respect to a PYTHON version that relies on NUMPY and SCIPY and make no use of explicit loops. The absolute execution times are reported in Table B1. The horizontal brackets highlight typical matrix sizes for realistic ALMA and VLA array configurations (cf. Table 1).

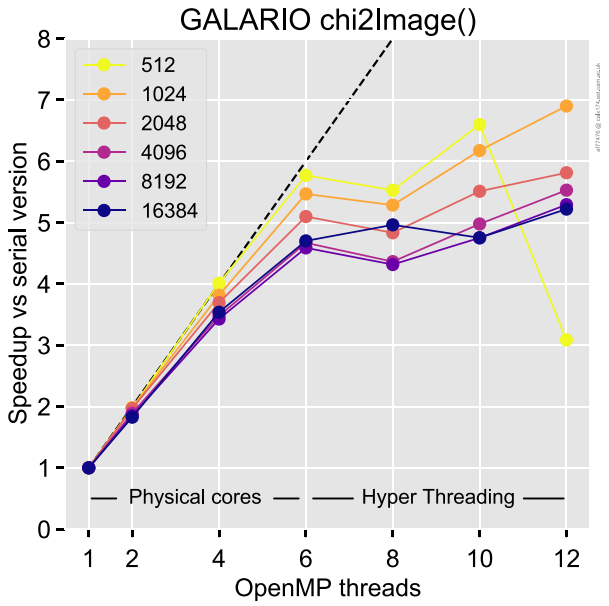

Figure B2. Strong scaling of the CPU version for different image sizes: speed-up (with respect to the serial version) for increasing computing power.

Table B1. Execution times of `chi2Image`.

N_{lm} (px)	Python (ms)	Serial (ms)	CPU 6 threads (ms)	CPU 12 threads (ms)	GPU (ms)
512	799	101	18	33	11
1024	1385	110	20	16	13
2048	2651	172	34	30	22
4096	5747	489	105	88	59
8192	13829	1589	346	300	208
16384	38311	7038	1497	1348	810

Note. Timings refer to the execution of the double-precision version of `chi2Image` with $M = 10^6$ visibility points.

APPENDIX C: ACCURACY (CONTINUED)

In this section we present some of the results of the accuracy checks that we carried out against the NRAO CASA package. We ran a large suite of tests, here we show only some representative cases for different model images.

In all these tests we used the results of the `sampleImage` function of GALARIO and of the `ft` command of CASA, which are designed to perform the same operation: compute the sampled visibilities $V(u_k, v_k)$ for a given model image. For each image of the source brightness $I_\nu(l, m)$, we computed the visibilities in two ways: (i) we applied GALARIO's `sampleImage` to the 2d matrix containing the image; (ii) we exported the image to a FITS file, imported it in CASA with the `importfits` task and then Fourier-sampled it with the `ft` task (invoked with `usescratch` option set to `True`).

Fig. C1 shows the comparison for three different input models, reporting a central cut of the image $I_\nu(l, m)$ (left column), a comparison of the amplitude $[\mathrm{Re}(V)^2 + \mathrm{Im}(V)^2]^{1/2}$ (central column) and a comparison of the phase $\arctan[\mathrm{Im}(V)/\mathrm{Re}(V)]$ (right column). In the amplitude and phase plots, the bottom panels represent, respectively, the relative and absolute difference between the GALARIO and the CASA results (except in the first row, where the results are each compared to the analytic solution). The uv-points used for the comparison represent a realistic uv coverage for an ALMA observation, with baselines in the range 11–1370 kλ, corresponding to a maximum recoverable scale $\theta_{\mathrm{MRS}} \sim 10$ arcsec and an angular resolution $\theta_{\mathrm{res}} \sim 0.15$ arcsec.

In the first row the image is an axisymmetric model centred at the phase centre. By definition its visibility function is real-valued and has an analytic solution given by the Hankel transform in equation (25). We therefore compare the results given by GALARIO and by CASA against the analytic solution. Both GALARIO and CASA reproduce very well the analytic amplitude within 0.1 per cent up to approximately 750 kλ. GALARIO is slightly more accurate than CASA at longer baselines, up to 875 kλ. The frequent spikes in the relative difference are due to the very sharp shape of the lobes, and occur only at the amplitude minima. Since the model is axisymmetric, the phase should be identically zero at every baseline:

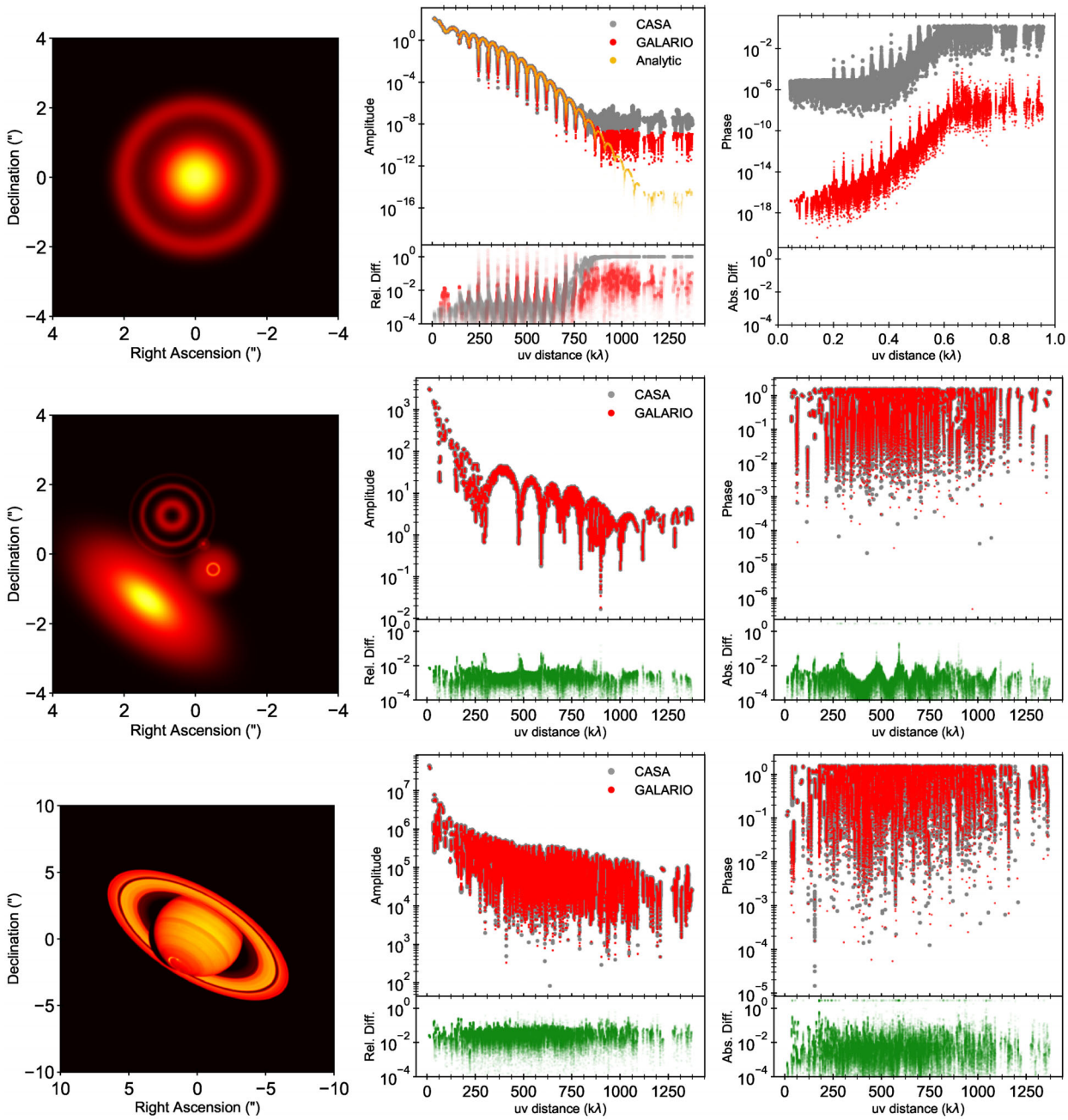

Figure C1. Results of the accuracy checks that we carried out against the NRAO CASA package. *Left column:* images of the simulated sources. *Central and right columns:* the comparison of the amplitude and phase of the synthetic visibilities. Each dot represents a *uv*-point. The lower panels of the amplitude and phase plots represent, respectively, the relative and the absolute difference between the computed quantities. *First row:* the results from CASA (grey dots) and GALARIO (red dots) are compared to the analytic solution (yellow dots). *Second and third row:* the results of CASA are compared to those of GALARIO.

the phase plot clearly shows that GALARIO is almost 10 orders of magnitude more accurate than CASA in reproducing the null phase.

In the second row the image is made by multiple sources displaced across the field of view, and in the third row by a mock observation of Saturn (not to scale). These images have been chosen because they exhibit structures spanning a wide range of spatial scales, and therefore are useful to probe the accuracy of the codes across a wide range of spatial frequencies. In both cases, the amplitude and phase comparison shows a good agreement between the results obtained by GALARIO and CASA. In the second row the vast majority of *uv*-points agree better than 0.5–1 per cent, while in the third row the agreement is slightly worse, within 1–4 per cent. The discrepancies are significant only for the visibility phase in a handful of *uv*-points (10–100) out of a total of approximately 10^5 visibilities. Additionally, we compared the synthesized images (not reported here) produced from the visibilities computed by CASA and GALARIO and we find that they also are in very good agreement. We note that the slightly larger discrepancies found in the third row might be due to the fact that the CASA `ft` task is likely applying a correction for wide field effects that is not included in the first release of GALARIO used here: this might be marginally relevant for the source brightness used in the third row, which is more extended

```python
import numpy as np
from scipy.interpolate import RectBivariateSpline, interp1d

def py_chi2Image(image, dxy, u, v, vis_obs_re, vis_obs_im, weights, dRA=0., dDec=0., PA=0.):
    """ Python implementation of galario `chi2Image` function. """
    nxy = reference_image.shape[0]
    dRA *= 2.*np.pi
    dDec *= 2.*np.pi
    du = 1. / (nxy*dxy)

    # Real to Complex transform
    fft_r2c_shifted = np.fft.fftshift(np.fft.rfft2(np.fft.fftshift(image)), axes=0)

    # rotate (u, v) point coordinates
    cos_PA = np.cos(PA)
    sin_PA = np.sin(PA)
    urot = u * cos_PA - v * sin_PA
    vrot = u * sin_PA + v * cos_PA
    dRArot = dRA * cos_PA - dDec * sin_PA
    dDecrot = dRA * sin_PA + dDec * cos_PA

    # compute interpolation indices
    urot_idx = np.abs(urot)/du
    vrot_idx = nxy/2. + vrot/du
    uneg = urot < 0.
    vroti[uneg] = nxy/2 - vrot[uneg]/du

    # coordinates of FT matrix
    u_axis = np.linspace(0., nxy // 2, nxy // 2 + 1)
    v_axis = np.linspace(0., nxy - 1, nxy)

    # sample the Fourier Transform in the (u, v) points
    f_re = RectBivariateSpline(v_axis, u_axis, fft_r2c_shifted.real, kx=1, ky=1, s=0)
    ReInt = f_re.ev(vrot_idx, urot_idx)
    f_im = RectBivariateSpline(v_axis, u_axis, fft_r2c_shifted.imag, kx=1, ky=1, s=0)
    ImInt = f_im.ev(vrot_idx, urot_idx)
    ImInt[uneg] *= -1.  # correct for Real to Complex transform frequency mapping

    # apply the phase change to translate image by (dRA, dDec)
    theta = urot*dRArot + vrot*dDecrot
    vis = (ReInt + 1j*ImInt) * (np.cos(theta) + 1j*np.sin(theta))

    chi2 = np.sum(((vis.real - vis_obs_re)**2. + (vis.imag - vis_obs_im)**2.)*weights)

    return chi2


def py_chi2Profile(intensity, Rmin, dR, nxy, dxy, u, v, vis_obs_re, vis_obs_im, weights, dRA=0., dDec=0., PA=0, inc=0.):
    """ Python implementation of galario `chi2Profile` function. """
    inc_cos = np.cos(inc)
    nrad = len(intensity)
    gridrad = np.linspace(Rmin, Rmin + dR * (nrad - 1), nrad)
    ncol, nrow = nxy, nxy

    # create the mesh grid
    x = (np.linspace(0.5, -0.5 + 1./float(ncol), ncol)) * dxy * ncol
    y = (np.linspace(0.5, -0.5 + 1./float(nrow), nrow)) * dxy * nrow
    x_axis, y_axis = np.meshgrid(x / inc_cos, y)
    x_meshgrid = np.sqrt(x_axis ** 2. + y_axis ** 2.)

    # bilinear interpolation on the 2d grid to create the image
    intensity *= dxy**2.  # convert to Jansky
    f = interp1d(gridrad, intensity, kind='linear', fill_value=0.,
                 bounds_error=False, assume_sorted=True)
    intensmap = f(x_meshgrid)
    f_center = interp1d(gridrad, intensity, kind='linear', fill_value='extrapolate',
                        bounds_error=False, assume_sorted=True)
    intensmap[int(nrow/2), int(ncol/2)] = f_center(0.)

        # use py_chi2Image to compute the chi square from the image
    chi2 = py_chi2Image(intensmap, dxy, u, v, vis_obs_re, vis_obs_im, weights, dRA=dRA, dDec=dDec, PA=PA)

    return chi2
```

Figure D1. Implementation of `py_chi2Image` and `py_chi2Profile`, the PYTHON version of GALARIO's `chi2Image` and `chi2Profile` functions. These PYTHON functions are used as reference for the speed-up factors computed in Section 6.

than those used in the first two rows. However, due to the lack of documentation in the CASA package, it was unclear how to disable such correction for wide field effects when using the `ft` task.

APPENDIX D: PYTHON REFERENCE FUNCTIONS

For completeness, in Fig. D1 we report the implementation of `py_chi2Image` and `py_chi2Profile`, the PYTHON version of GALARIO's `chi2Image` and `chi2Profile` functions. These PYTHON functions are used as the reference for the speed-up factors computed in Section 6. For all the computate-heavy operations they employ only optimized NUMPY and SCIPY functions, as provided in the Anaconda Python distribution. The `py_chi2Image` and `py_chi2Profile` functions are also provided in the unit tests in the online repository.

This paper has been typeset from a TEX/LATEX file prepared by the author.