Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

Updates on the RICK library

De Rubeis Emanuele, *Claudio Gheller, Giovanni Lacopo, Giuliano Taffoni, Luca Tornatore*

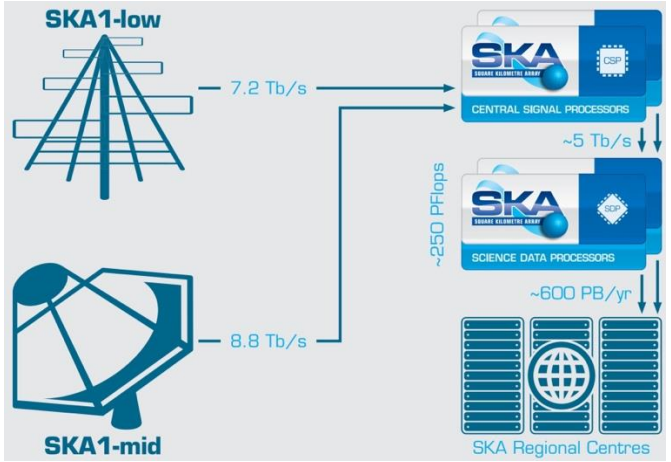
Spoke 3 III Technical Workshop, Perugia May 26 -29, 2025

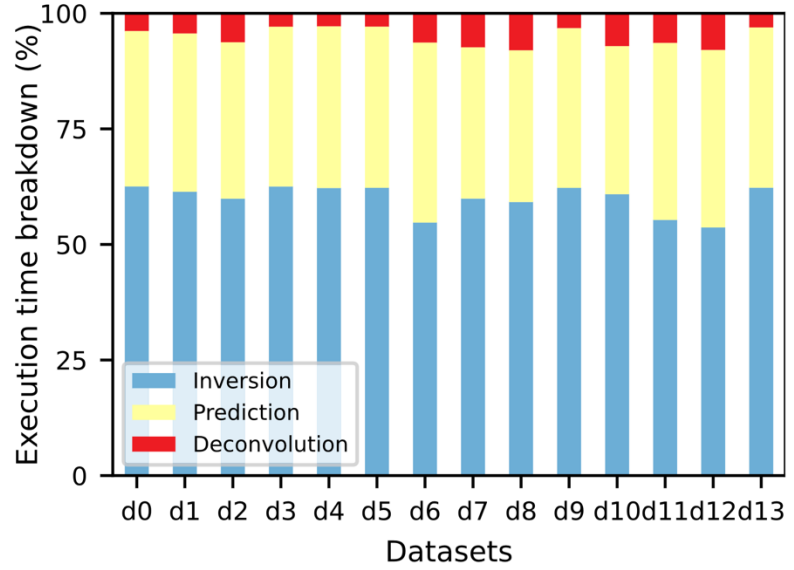
Scientific Rationale

Why HPC for radio astronomy?

Current and upcoming radio-interferometers are expected to produce **volumes of data of increasing size**. This means that current **state-of-the-art software needs to be re-designed** to handle such unprecedented **data challenge**.

Imaging in radio astronomy represents one of the most **computational demanding** steps of the processing pipeline, both in terms of memory request and in terms of computing time.



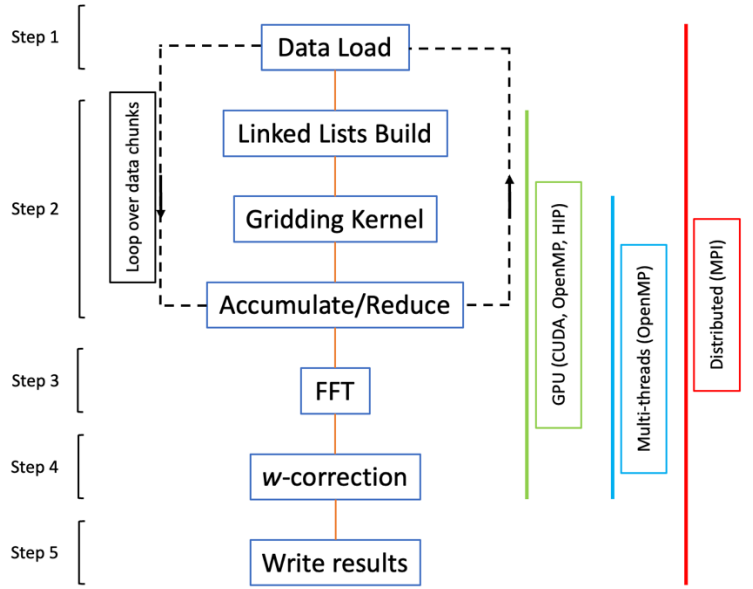


Corda et al. (2022)

Technical Objectives, Methodologies and Solutions

RICK (Radio Imaging Code Kernels) is a code that addresses the ***w*-stacking algorithm** (Offringa+14) for imaging, combining parallel and accelerated solutions.

- The code is written in **C** (with extensions to **C++**)
- **MPI** & **OpenMP** for CPU parallelization
- The code is capable of **running full on NVIDIA GPUs**, using CUDA for offloading
- HIP or OpenMP are also available for other architectures (such as AMD)



Adapted from De Rubeis et al. (2025)

Main Results

The code has been turned now into a **library** (https://github.com/ICSC-Spoke3/RICK/tree/library).

This mainly guarantees:

- **modularization**

```
void gridding(
    int rank,
    int size,
    int nmeasures,
    double *uu,
    double *vv,
    double *ww,
    double *grid,
    double *gridss,
    MPI_Comm MYMPI_COMM,
    int num_threads,
    int grid_size_x,
    int grid_size_y,
    int w_support,
    int num_w_planes,
    int polarisations,
    int freq_per_chan,
    float *visreal,
    float *visimg,
    float *weights,
    double uvmin,
    double uvmax)
```

```
void fftw_data(
    int grid_size_x,
    int grid_size_y,
    int num_w_planes,
    int num_threads,
    MPI_Comm MYMPI_COMM,
    int size,
    int rank,
    double *grid,
    double *gridss)
```

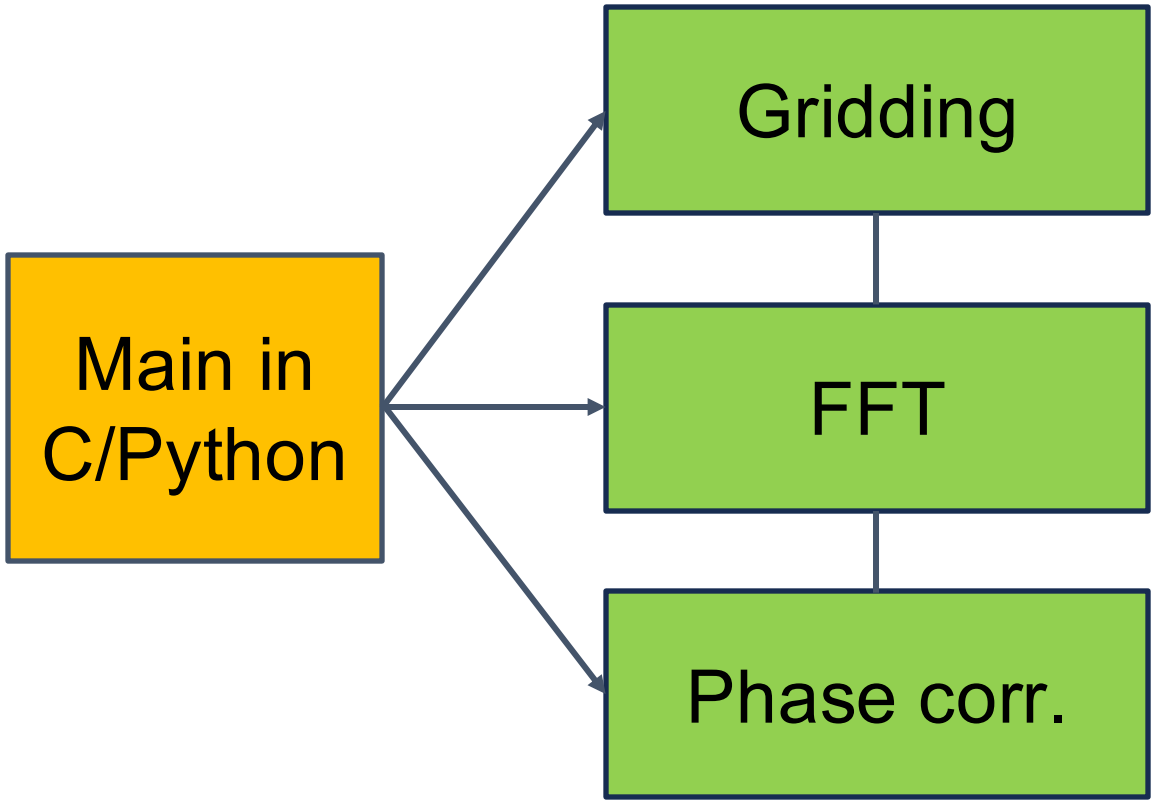
```
void phase_correction(
    double *gridss,
    double *image_real,
    double *image_imag,
    int num_w_planes,
    int xaxistot,
    int yaxistot,
    double wmin,
    double wmax,
    double uvmin,
    double uvmax,
    int num_threads,
    int size,
    int rank,
    MPI_Comm MYMPI_COMM)
```

Main Results

The code has been turned now into a **library** (https://github.com/ICSC-Spoke3/RICK/tree/library).

This mainly guarantees:

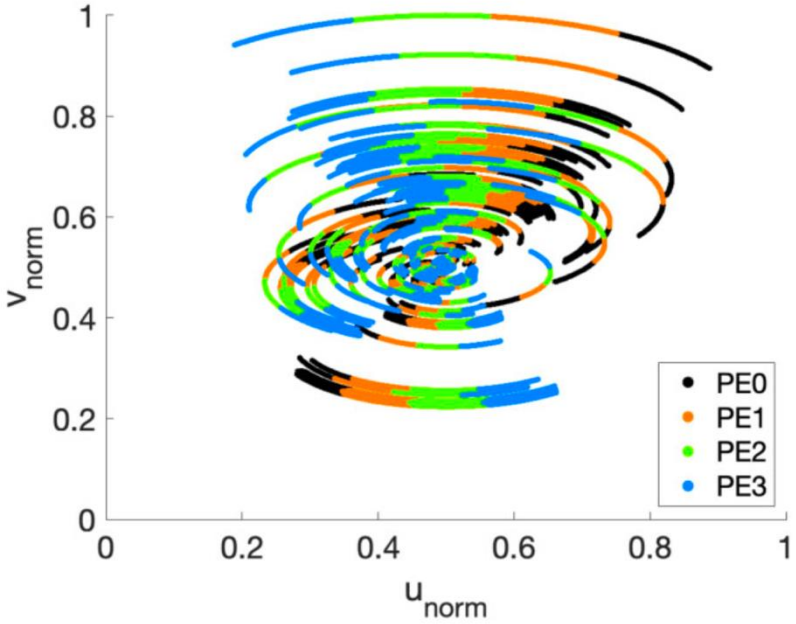
- **modularization**
- **portability**



Main Results

The distributed approach is extremely beneficial for interferometric imaging, as proved in De Rubeis et al. (2025).

- **Visibilities** are distributed among *N* processing units (e.g., MPI tasks, GPUs) in time-order



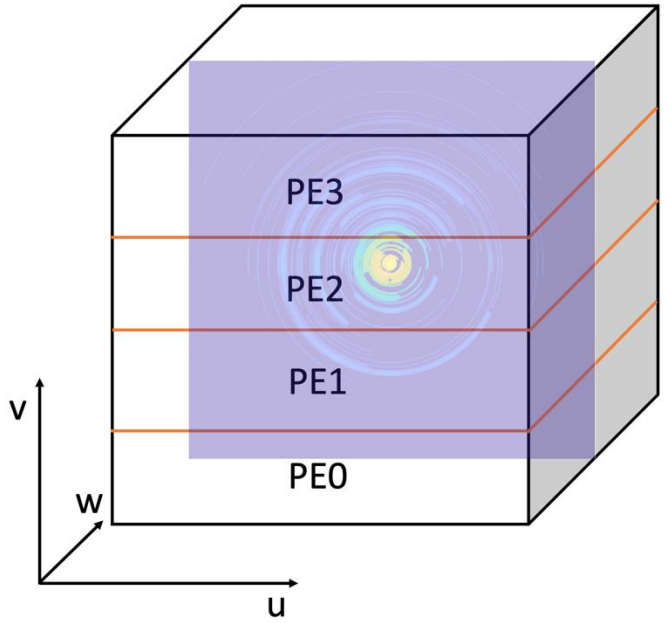
Gheller et al. (2023)

Main Results

The distributed approach is extremely beneficial for interferometric imaging, as proved in De Rubeis et al. (2025).

- **Visibilities** are distributed among *N* processing units (e.g., MPI tasks, GPUs) in time-order
- The **mesh** is split among *N* processing units

Problems of **any size** are supported, but visibilities are distributed across **memories unrelated to the mesh slabs.** This requires **communication**!



Gheller et al. (2023)

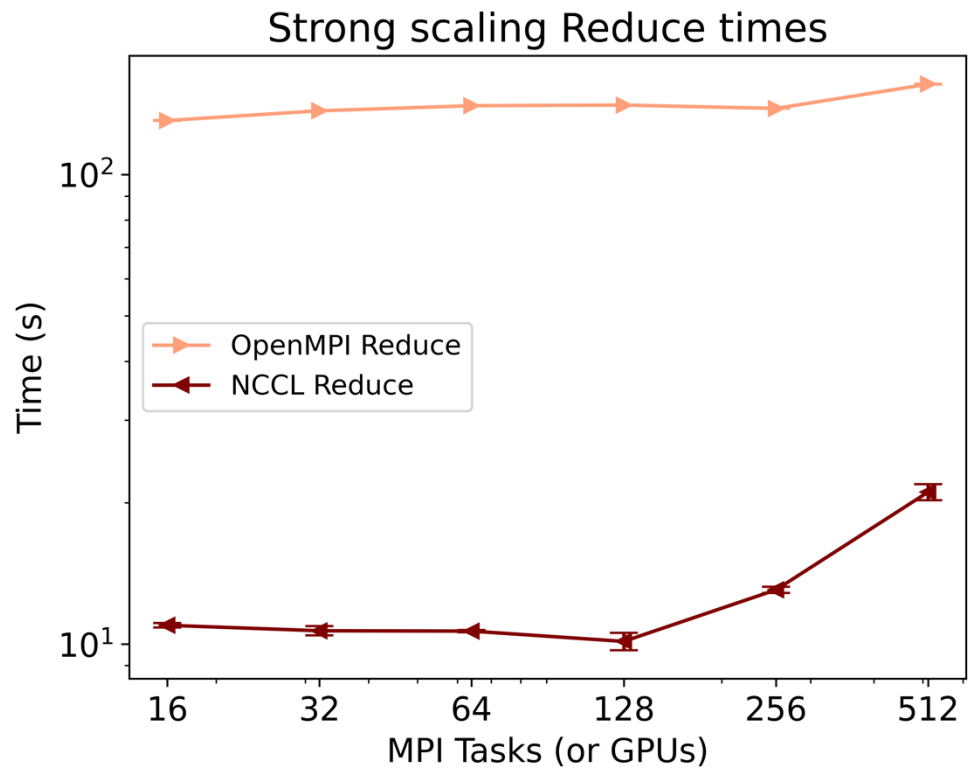
Main Results

Back in Bologna: «Watch out for the Reduce!»

RICK is now limited by the communication costs, with the Reduce operation that becomes the true bottleneck of the code.

This means that it is important to choose the right number of computational resources based on the problem size.

Increasing power does not necessarily return an increasing performance!

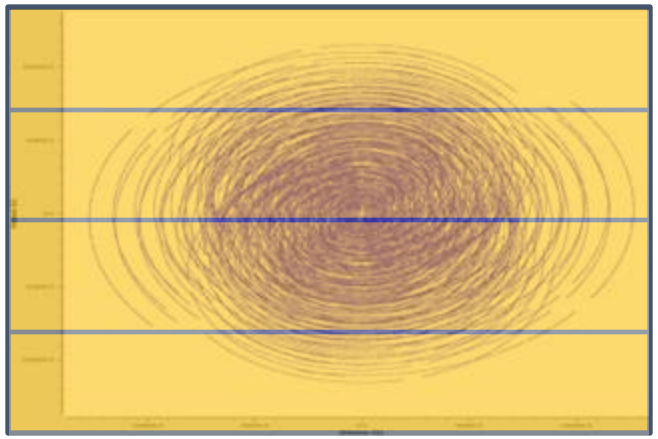


De Rubeis et al. (2025)

Main Results – Can we reduce the Reduce?

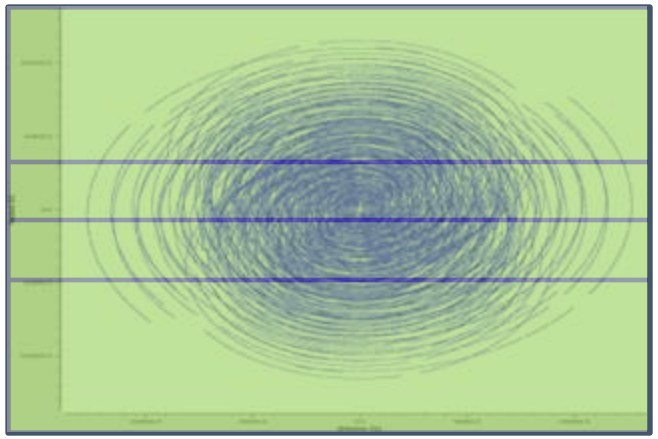
We tried to find a solution to the communication overhead of RICK implementing a different **domain decomposition**

Old approach



Grid **evenly distributed** among processing units – **unbalanced** gridding, but balanced FFT

New approach



Grid **unequally distributed** among processing units – **balanced** gridding, but unbalanced FFT

Main Results – Can we reduce the Reduce?

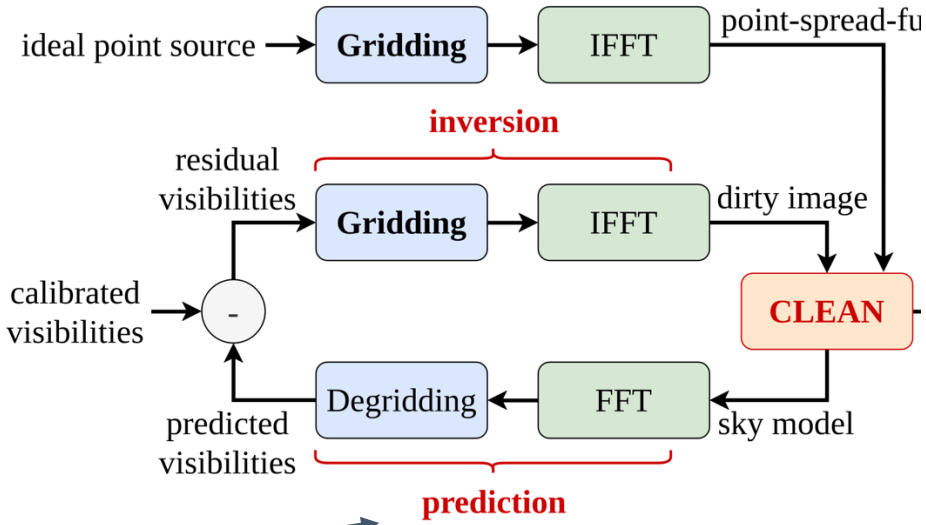
We tried to find a solution to the communication overhead of RICK implementing a different **domain decomposition**

Visibilities are bucket-sorted along the *v*-axis of the *uv* plane, and assigned to each processing unit.

✓ Each processing unit has now its resident visibilities, meaning **no additional communication**

This **removes** the Reduce problem, which represented the main bottleneck of our code.

Moreover, it will make more efficient a future implementation of the **prediction** in RICK



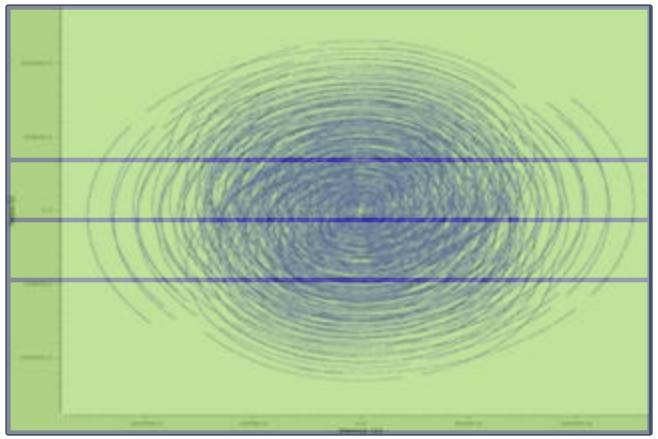
Main Results – Can we reduce the Reduce?

We tried to find a solution to the communication overhead of RICK implementing a different **domain decomposition**

Visibilities are bucket-sorted along the *v*-axis of the *uv* plane, and assigned to each processing unit.

- ✓ Each processing unit has now its resident visibilities, meaning **no additional communication**
- ❖ We need now an FFT implementation supporting **non-trivial domains**

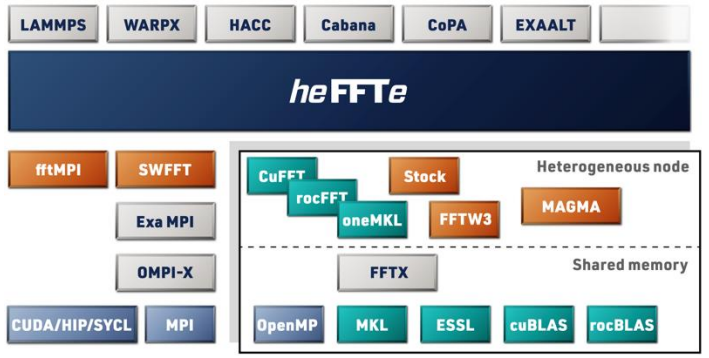
New approach



Grid **unequally distributed** among processing units – **balanced** gridding, but not FFT and phase correction

Main Results – Can we reduce the Reduce?

A possible solution is represented by the **heFFTe** library (https://icl.utk.edu/fft/)



- API supported for **C/C++/Python**
- Allows **non-trivial** and **non-evenly distributed domain decomposition**
- Fully **portable**, having backends for FFTW, cuFFT, rocFFT, oneMKL

Main Results – Implementing RICK into WSClean/DDFacet

Having turned RICK into a library simplified inserting it into WSClean, but still **tests** needed to be done to check the **validity** of the final results:

- **normalization** for the channel frequency of *u, v, w* coordinates
- **phase-correction** validation
- options to deal with **non-standard** MS (e.g., 1, 2, 4 correlations)

Next steps

- Testing the heFFTe on AMD (Setonix) and NVIDIA (Leonardo) clusters.

- **Fitting RICK library into WSClean**, to make it also scientifically productive.