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

Radio Imaging Code Kernels

De Rubeis Emanuele (INAF-IRA)

Claudio Gheller (INAF-IRA)

Giovanni Lacopo (OATs)

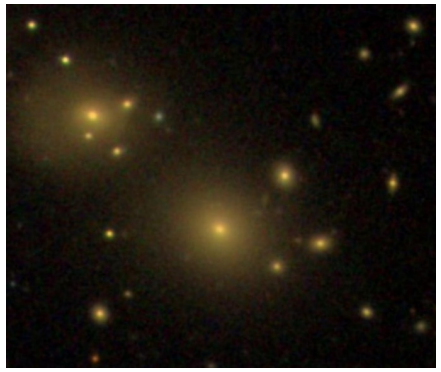
Giuliano Taffoni (OATs)

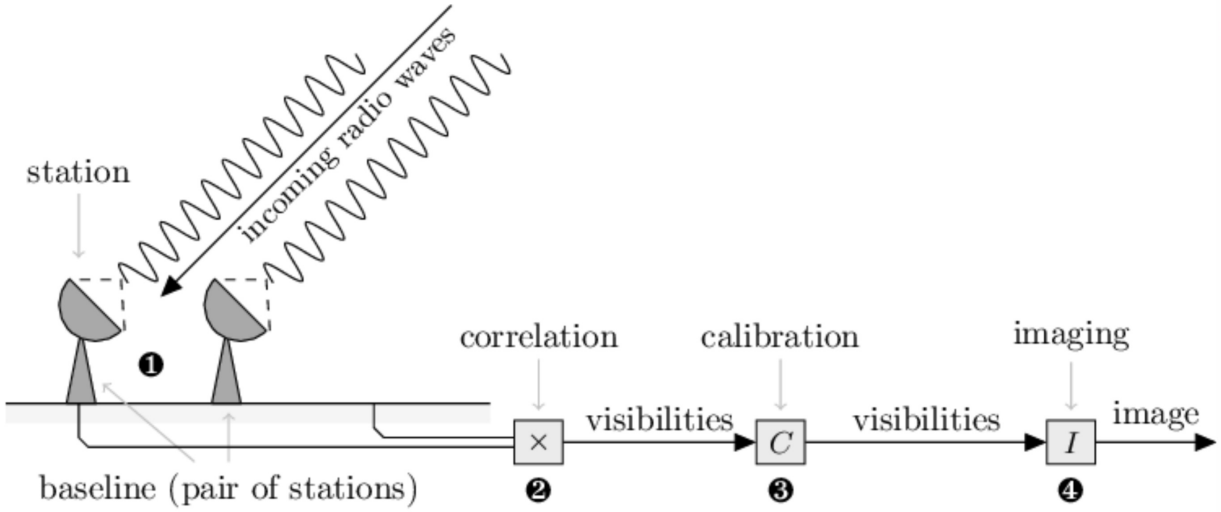
Luca Tornatore (OATs)

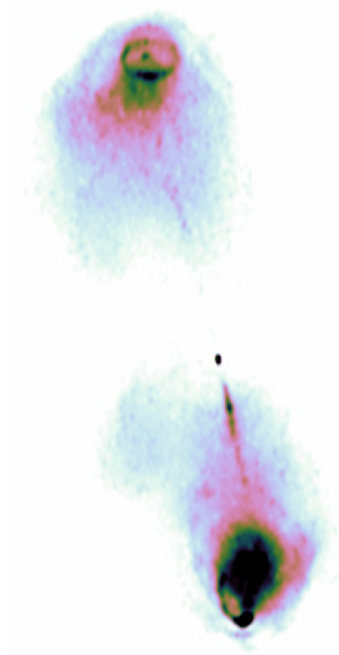
Monthly WP1-WP2 Meeting, 20/03/2024

Radio astronomy imaging

In radio astronomy, imaging is the process through which the brightness distribution $I(l,m)$ is determined by the complex visibility $V(u,v,w)$ observed by a radio telescope







Radio astronomy imaging

There is a Fourier transform relationship between the observed visibilities $V(u,v,w)$ and the brightness distribution $I(l,m)$

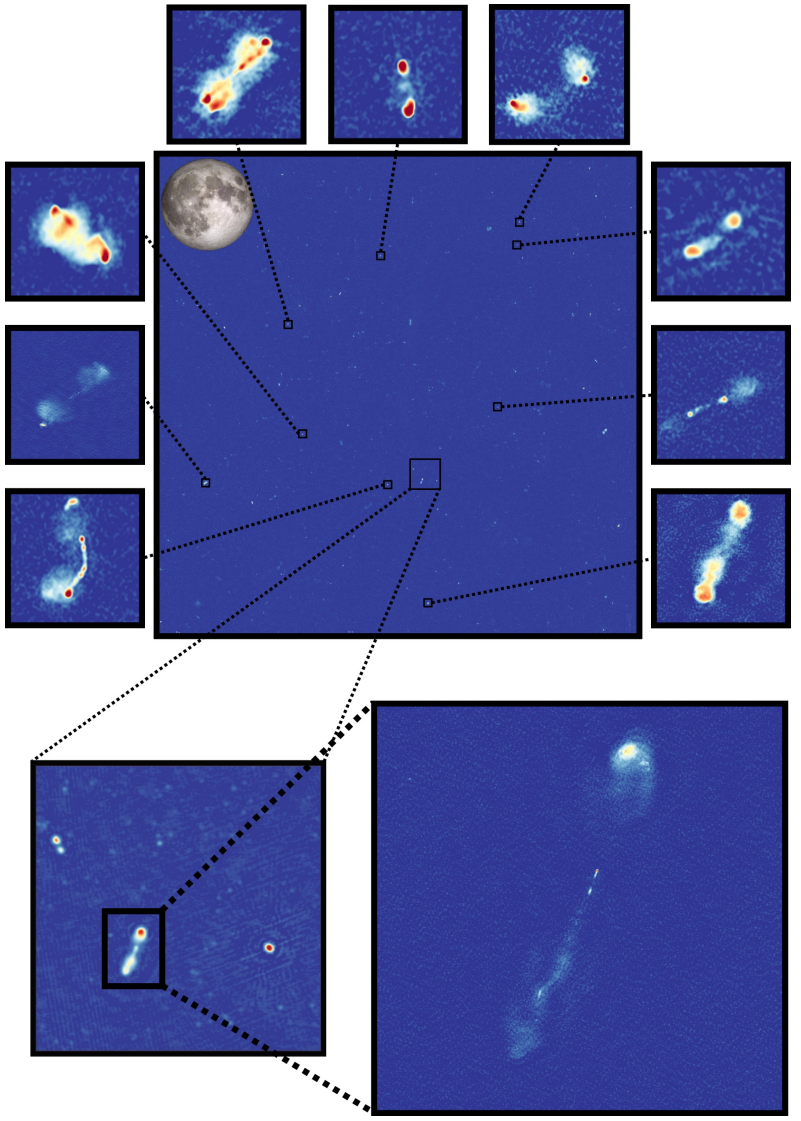
$$V(u,v,w) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(l,m) e^{-2\pi i\left[ul+vm+w\left(\sqrt{1-l^2-m^2}-1\right)\right]} \frac{dl\,dm}{\sqrt{1-l^2-m^2}}$$

Why HPC in radio astronomy

Current and upcoming radio-interferometers are expected to produce **volumes of data of increasing size** that need to be processed in order to generate the corresponding sky brightness distributions through imaging.

This represents an **outstanding computational challenge**, especially when **large fields of view** and/or **high-resolution** observations are processed.

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



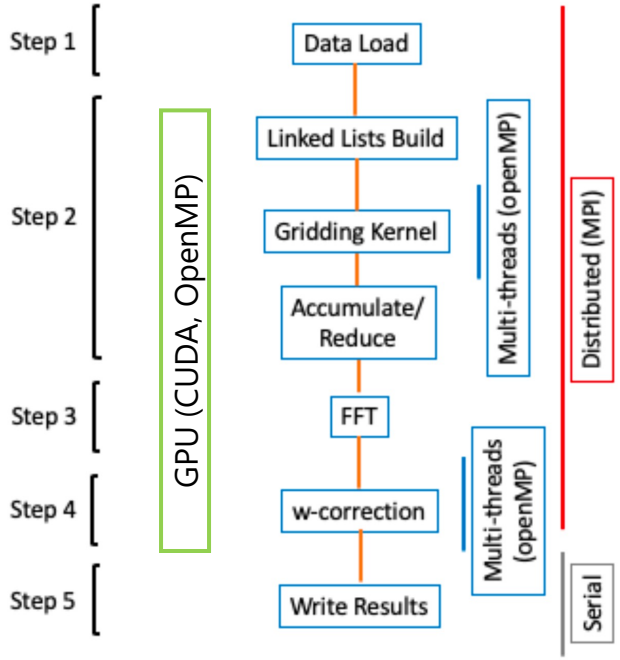
Sweijen et al. (2022)

For example, this 83,900x83,500 pixels image can take ~250,000 core-hours!

What is RICK?

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

- **C**, **C++**, **CUDA, HIP**
- **MPI** & **OpenMP** for CPU parallelization
- The code is now capable of **running full on GPU**s, using CUDA, HIP or OpenMP for offloading
- An optimized version of the **reduce** has been developed on both CPU (combining MPI+OpenMP) and GPU (using **NCCL** or **RCCL**, for Nvidia and AMD respectively); the **FFT** is done through the **cuFFTMp** library for Nvidia
- Currently under benchmarking on Leonardo (CINECA, No.4 Top500 June 23)



Adapted from Gheller et al. (2023)

What were our targets?

- ✓ Full GPU enabling, avoiding data moving back and forth between CPU and GPU
- ✓ Code presentation at ADASS XXXIII (November 2023)
- ✓ Data release of the GPU-enabled code (namely RICK v2.0)

Why do we need multiple GPUs?

Modern and future radio telescopes will produce a **huge amount of data**, that hardly fit the memory of a single GPU (not even a single node)

Why do we need multiple GPUs?

Modern and future radio telescopes will produce a **huge amount of data**, that hardly fit the memory of a single GPU (not even a single node)

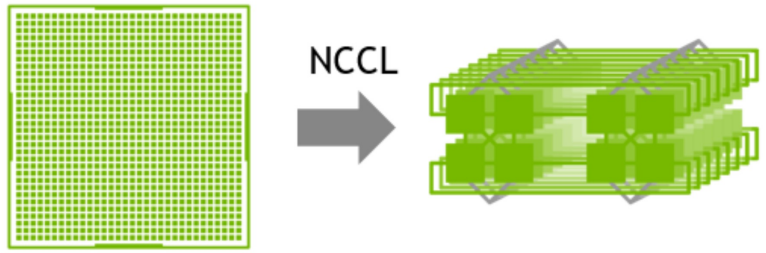
The answer is then **to distribute the problem** among multiple GPUs and multiple nodes

Easy to say, more difficult to do…

Nvidia Collective Communication Library (NCCL)

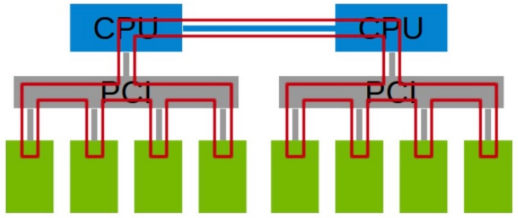
NCCL is a library of multi-GPU collective communication used to support the *Reduce* operation.

- Provides **fast collectives** over **multiple GPU both within and across nodes**.
- Supports a variety of **interconnect** technologies (e.g. NVLink, PCIe).
- NCCL closely follows the popular **collectives** API defined by **MPI**, so can be very “natural” to use.

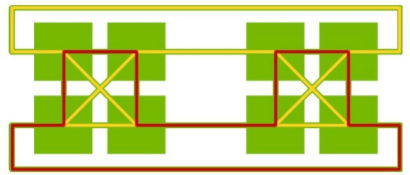


Nvidia Collective Communication Library (NCCL)

NCCL implements the *Reduce* operation as an intra-node ring, and an inter-node ring, when GPUs assigned to the main tasks communicate with RDMA with GPUs in different nodes without passing through the CPUs.



PCIe / QPI : 1 unidirectional ring



DGX-1 : 4 unidirectional rings

```
ncclUniqueId id;
ncclComm_t comm;

ncclCommInitRank(&comm, size, id, rank);

ncclReduce(send_g, recv_g, size_g, ncclDouble, ncclSum, taget_rank, comm, stream)
```

Nvidia Collective Communication Library (NCCL)

NCCL implements the *Reduce* operation as an intra-node ring, and an inter-node ring, when GPUs assigned to the main tasks communicate with RDMA with GPUs in different nodes without passing through the CPUs.

```
ncclUniqueId id;
ncclComm_t comm;

ncclCommInitRank(&comm, size, id, rank);

ncclReduce(send_g, recv_g, size_g, ncclDouble, ncclSum, taget_rank, comm, stream)
```

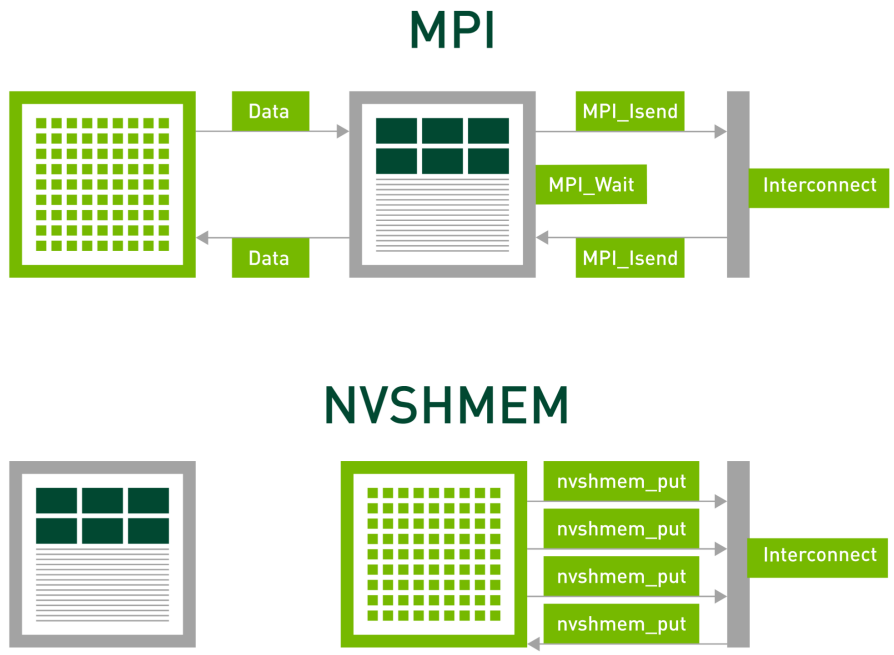
The requirement of a dedicated stream for the *Reduce* comes from the presence of asynchronous memory copies that collided with the ones within a previous function call

cuFFTMp

Fast Fourier Transform is a critical operation in radio astronomy, because it determines the relationship between the “observed” and the “desired” data (the final image).

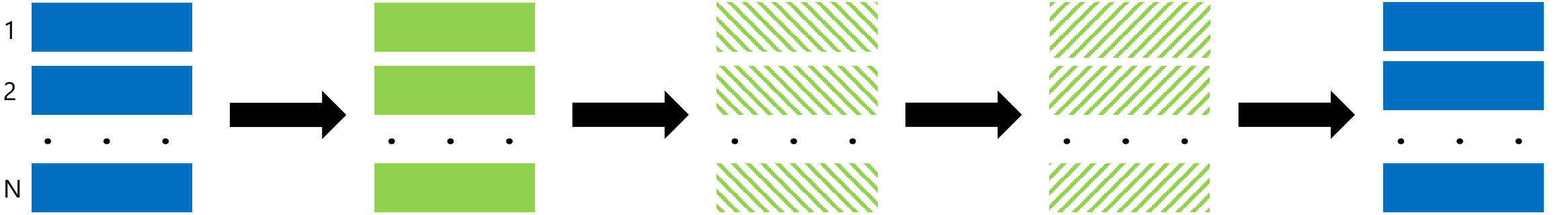
For the FFT step, RICK implements the **cuFFTMp** library, that allows to **distribute the FFT** problem using **NVSHMEM**

NVSHMEM uses asynchronous, **GPU-initiated data transfers**, eliminating synchronization overheads between the CPU and the GPU



cuFFTMp

Data are distributed among multiple GPUs and inverse-transformed.



However, **now we have data already on the GPU!**

cuFFTMp

We may need to do this FFT process even 100-1000s times, and at each time we need to create and destroy the *descriptor*, which is the data structure used by the library for the FFT.

This was critical for the performance, but we overcame this problem using CUDA kernels to write the *to-be-transformed* data each loop, and then putting them directly inside the descriptor.

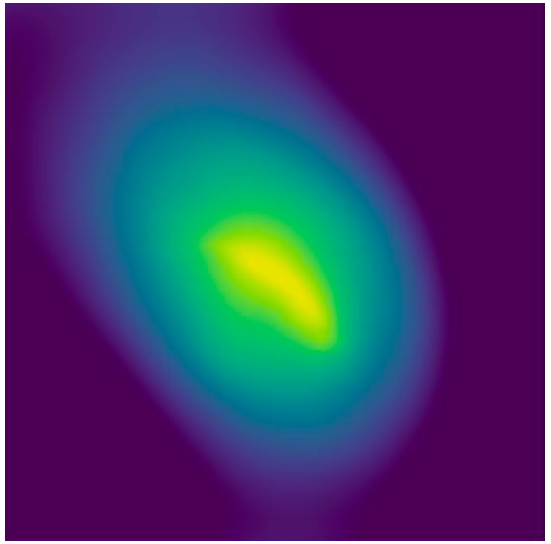
Another issue comes with the joint usage of NCCL and NVSHMEM, which can bring to severe errors during the FFT: to solve this, we need to switch off NCCL support for NVSHMEM at runtime by setting *NVSHMEM_DISABLE_NCCL=1*.

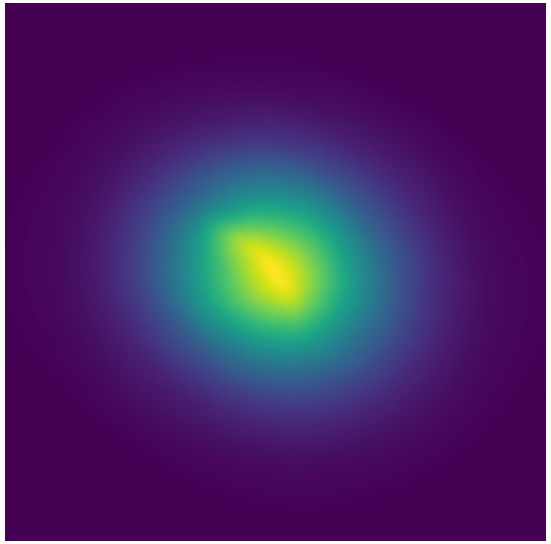
Was it worth it?

We are currently testing our code on Leonardo (CINECA) using Nvidia HPC-SDK 24.3. We are using real LOFAR-VLBI data, that is the closest facility to SKA in terms of size of end-products.

Comparing the code with GPUs, with respect to the one on CPUs:

- for a small test (~4 GB), we got a speed-up up to a factor ~ **x27** for both *Reduce* and FFT
- for a big test (~530 GB), we got a speed-up up to a factor ~ **x175** for the *Reduce* and ~ **x32** for the FFT (but here we are still doing benchmarks...)





Some work to do

- Complete the CPU benchmarks on the large dataset and test the weak scaling of the code
- Finalize the paper on the GPU enabling of the code
- Work on the AMD support of the code, but we miss required libraries such as a distributed FFT
- GPU porting of the weighting and *uv*-tapering for scientific purposes

Conclusions

- RICK is a code for enabling radio astronomy imaging on accelerators
- It is now capable of fully running on Nvidia GPUs thanks to the NCCL *Reduce* and the cuFFTMp
- A paper will soon be submitted with the presentation of the new updates and the benchmarks result on Leonardo (CINECA)
- More work is still required for the AMD GPU offloading