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

BrahMap

An optimal map-making solution for the future CMB experiments

Avinash Anand[1] and Giuseppe Puglisi[2]

[1]University of Rome "Tor Vergata", [2]University of Catania

Spoke 3 Monthly WP1-2 Meeting | February 19, 2024

Map-making for CMB experiments

- Future CMB experiments will target the B-mode polarization of CMB
- A low SNR and a wide frequency range of foreground contamination requires the deployment of O(3)-O(5) detectors sampling at very high frequency
- Data acquisition: ~250 TB from space to ~10 PB from ground-based experiments
- First step of analysis: Reduction of time-series data to sky maps *aka* Map-making
- Map-making goals:
  - Reduction of an enormous amount of data in a reasonable amount of time
  - Mitigation of instrumental systematics
  - Removal of both un-correlated and correlated noise

Map-making for CMB experiments

- Data model for CMB signal $$d_{p,t} = I_p + Q_p \sin 2\phi_t + U_p \cos 2\phi_t + n_t$$ .... (1)
  - $d_{p,t}$ → Signal measured by the detector
  - I_p, Q_p, U_p → CMB stokes parameters
  - ϕ_t → Detector polarization angle
  - n_t → Un-correlated and correlated noise contribution

- Writing data model in matrix equation form

$$d = P \cdot s + n$$ .... (2)

  - d → signal vector, P → pointing matrix, s → sky map, n → noise vector
  - Need to solve the equation for sky map, s

$$d = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n_t-2} \\ d_{n_t-1} \\ d_{n_t} \end{bmatrix} \quad P = \begin{bmatrix} \dots & \dots & 1 & \sin 2\phi_{t_1} & \cos 2\phi_{t_1} & \dots \\ \dots & 1 & \sin 2\phi_{t_2} & \cos 2\phi_{t_2} & \dots & \dots \\ \dots & \dots & 1 & \sin 2\phi_{t_3} & \cos 2\phi_{t_3} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & 1 & \sin 2\phi_{t_{n_t-2}} & \cos 2\phi_{t_{n_t-2}} & \dots \\ \dots & 1 & \sin 2\phi_{t_{n_t-1}} & \cos 2\phi_{t_{n_t-1}} & \dots & \dots \\ \dots & 1 & \sin 2\phi_{t_{n_t}} & \cos 2\phi_{t_{n_t}} & \dots & \dots \end{bmatrix} \quad s = \begin{bmatrix} I_1 \\ Q_1 \\ U_1 \\ \vdots \\ I_{N_p} \\ Q_{N_p} \\ U_{N_p} \end{bmatrix} \quad n = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \vdots \\ n_{n_t-2} \\ n_{n_t-1} \\ n_{n_t} \end{bmatrix}$$

Map-making for CMB experiments

- Generalized least-square solution (GLS)
  - A maximum-likelihood solution

$$\hat{s} = (P^T N^{-1} P)^{-1} P^T N^{-1} d \qquad \text{.......... (3)}$$

  - $N \rightarrow$ Covariance matrix of the noise time stream

- Destriper solution
  - A map-making solution that takes explicitly into account the *1/f* noise

$$\hat{s} = \left(F^T N^{-1} Z F\right)^{-1} F^T N^{-1} Z d \qquad \text{.......... (4)}$$

  - F is the matrix that contains the baseline information, and $Z = I - P\left(P^T N^{-1} P\right)^{-1} P^T N^{-1}$
  - Assumes *1/f* noise contribution to be constant in a baseline of a given length

---

- **For the future CMB experiments, these matrices will be huge → *O(10⁹x10⁹)* and above**
- **Map-making with entire dataset will take 200,000 - 50,000,000 CPU hours**

BrahMap: A scalable map-making framework

- A modular and object-oriented framework based on COSMOMAP2[1,2]
- Python 3 interface with bindings for compute intensive parts written in C++

- Optimization to squeeze the most out of the supercomputing resources
- Scalable across multiple computing nodes
- Offloading the computations to multiple GPUs
- Utilize the capabilities of high-performance matrix-algebra libraries (e.g. MAGMA)

[1]Puglisi, G., et al. "*Iterative map-making with two-level preconditioning for polarized cosmic microwave background data sets - A worked example for ground-based experiments.*" A&A, 618 (2018) A62, https://doi.org/10.1051/0004-6361/201832710
[2]https://github.com/giuspugl/COSMOMAP2

BrahMap: Derivation from COSMOMAP2

- Conversion of codebase from Python 2 to Python 3
  - Automated conversion using Python `2to3` tool followed by manual debugging and validation
- Writing the compute extensive parts to C++
  - To control hardware specific low level optimization
  - To use generic data types to using templates (to use generic Python data types)
- Writing Python binding for C++ codes using `pybind11`
  - `pybind11` is a header-only lightweight library, with no dependencies
  - `pybind11` can be shipped with Python package
  - Supports C++11 and STL out of the box
  - Compatible with all major compiler

BrahMap: Object oriented framework

- Process the pointing information

```
proc_points = ProcessTimeSamples(pixs=pointings, npix=npix,
                                 pol=3,phi=pol_angles)
npix_new, _ = proc_points.get_new_pixel
```

- Create the pointing matrix as Sparse linear operator

```
P = SparseLO(n=npix_new, m=nsamp, pix_samples=proc_points.pixs, pol=3,
             angle_processed=proc_points)
```

- Make a noise covariance matrix

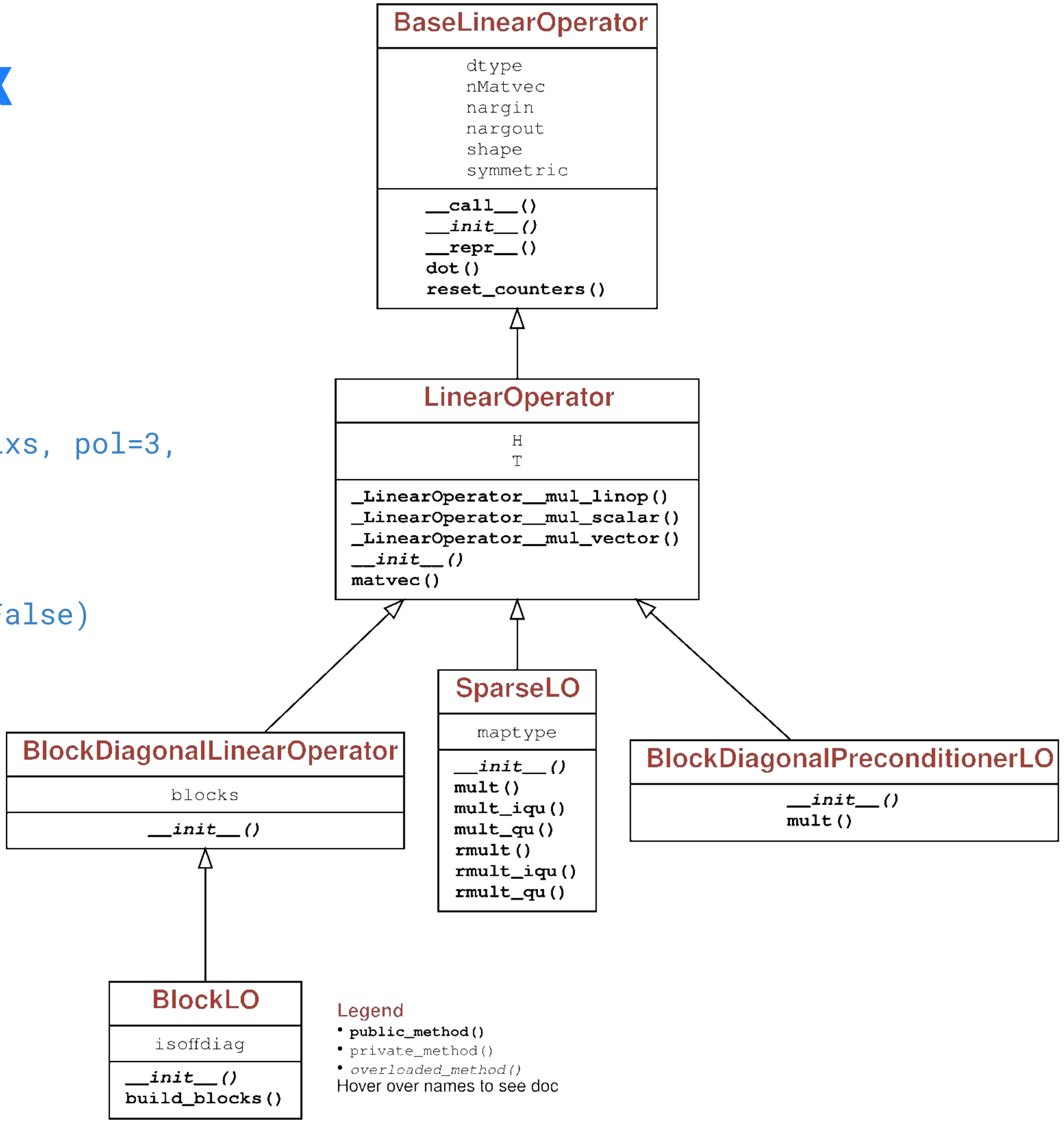
```
inv_N = BlockLO(blocksize=blocksize, t=inv_sigma2, offdiag=False)
```

- Make the Jacobi-preconditioner

```
Mbd = BlockDiagonalPreconditionerLO(CES=proc_points,
                                    n=npix_new, pol=3)
```

- Solve for the sky map using conjugate gradient method (GLS solution)

```
A = P.T * inv_N * P
b = P.T * inv_N * tod_array
map_out = scipy.sparse.linalg.cg(A, b, M=Mbd)
```

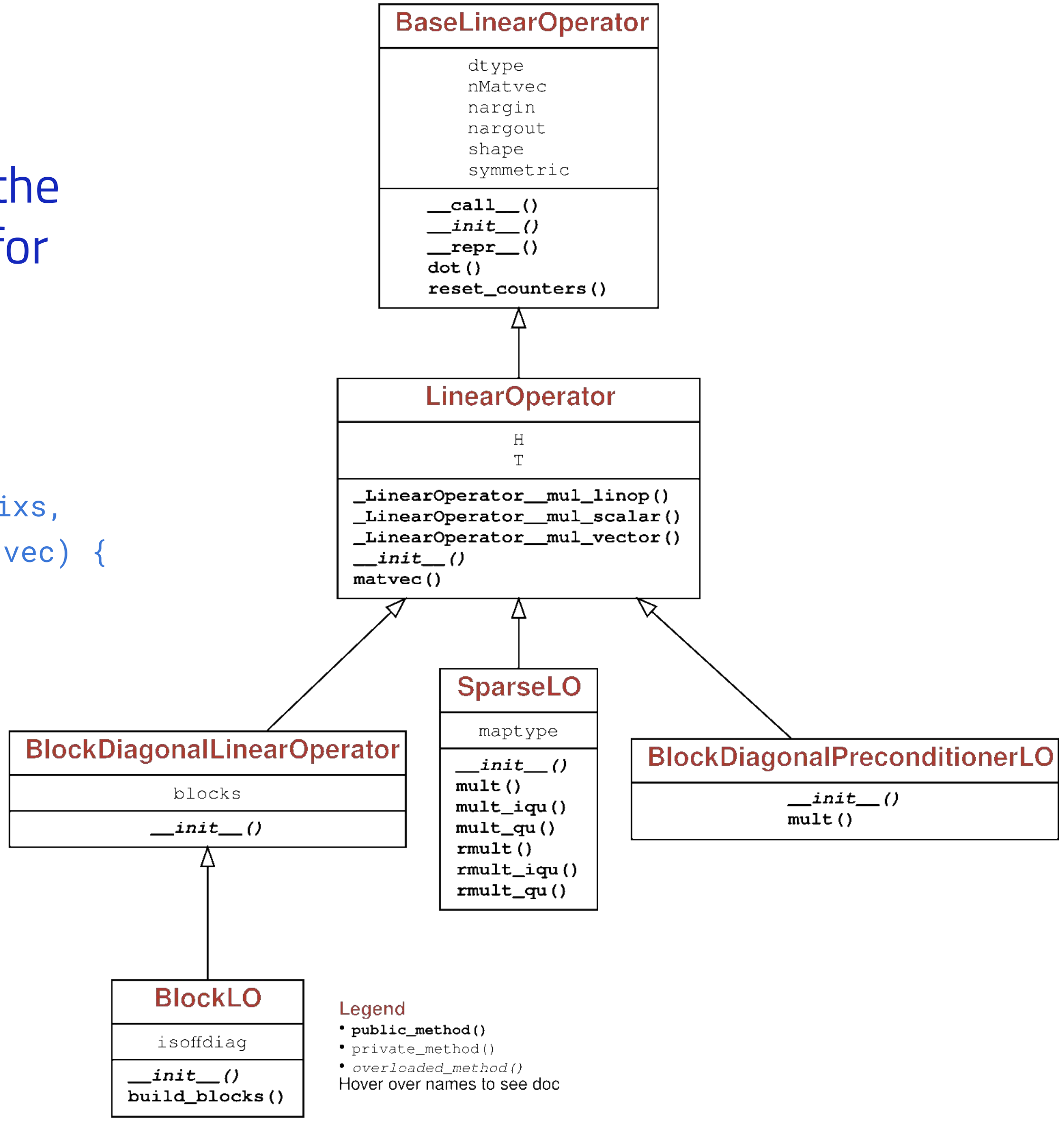


BrahMap: Modular framework

- Multiple definitions for underlying methods, opening the possibility of writing compute extensive parts in C++ for both CPUs and GPUs independently

```
template <typename dtype_int, typename dtype_float>
py::array_t<dtype_float> SparseLO_rmult(int Nrows, int Ncols,
                                        py::array_t<dtype_int> pixs,
                                       py::array_t<dtype_float> vec) {
  py::array_t<dtype_float> x_arr({Ncols});
  auto x_arr_ptr = x_arr.mutable_data();
  auto pixs_ptr = pixs.template unchecked<1>();
  auto vec_ptr = vec.template unchecked<1>();

  for (ssize_t idx = 0; idx < Nrows; ++idx) {
    if (pixs_ptr[idx] == -1) continue;
    x_arr_ptr[pixs_ptr[idx]] += vec_ptr[idx];
  } // for
  return x_arr_ptr;
} // SparseLO_rmult()
```

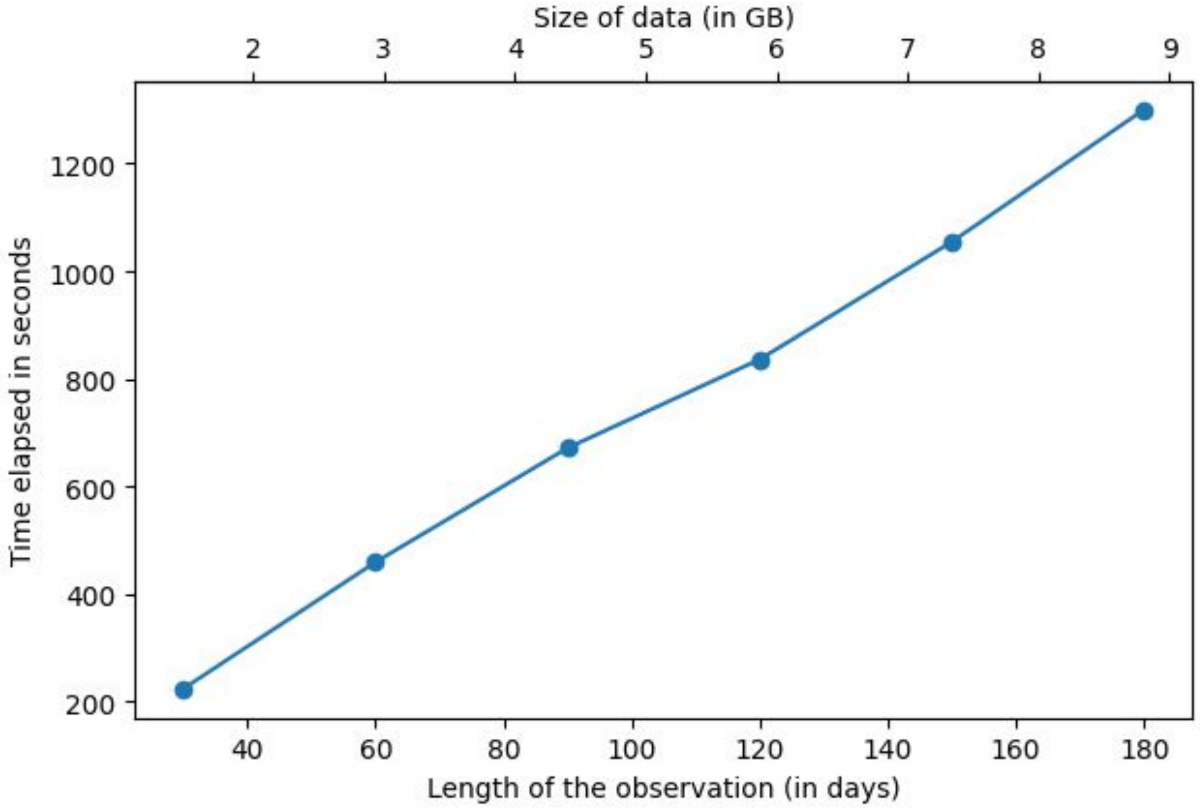


BrahMap: Code performance

- Runtime comparison

  - BrahMap: Python only version

    Scaling with the size of dataset



BrahMap: Code performance

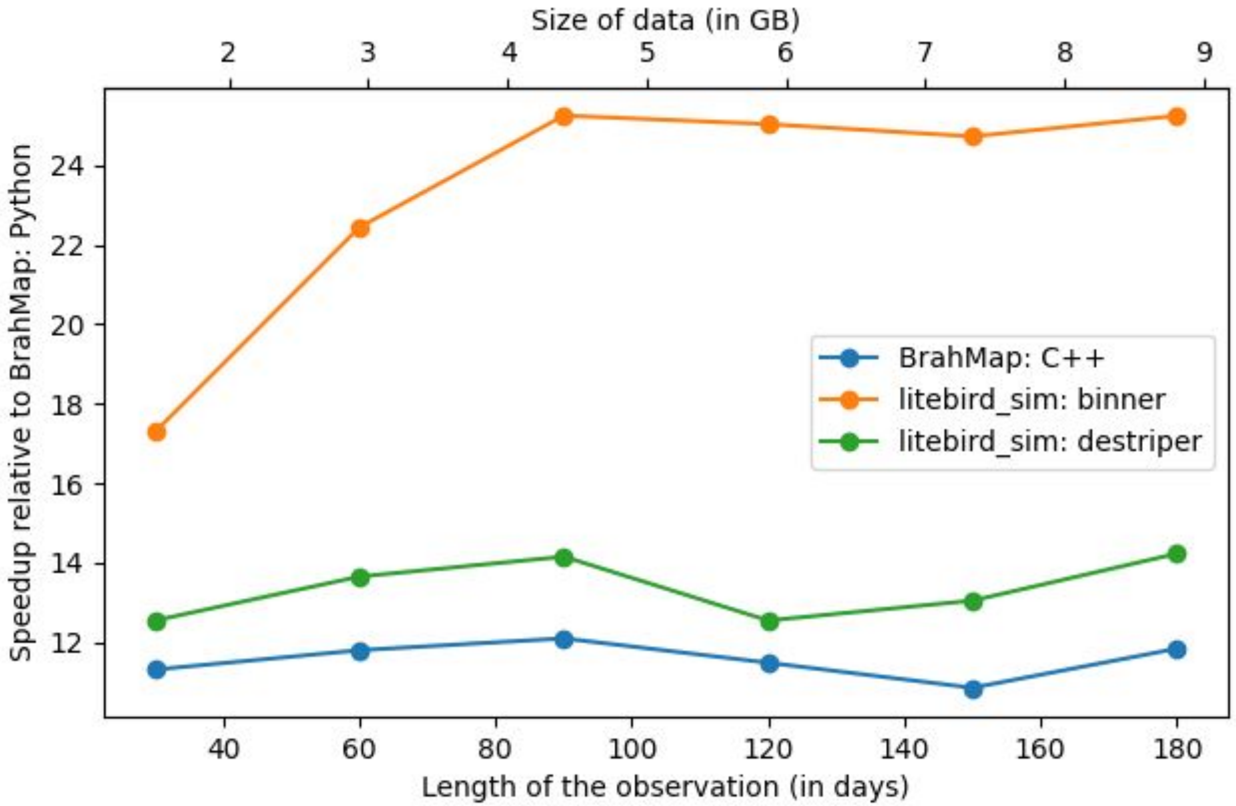
- Runtime comparison

  - BrahMap: Python binding with C++

    10 to 12 times faster
  - litebird_sim: `make_binned_map()` routine

    17 to 25 times faster
  - litebird_sim: `make_destriped_map()` routine

    12 to 14 times faster

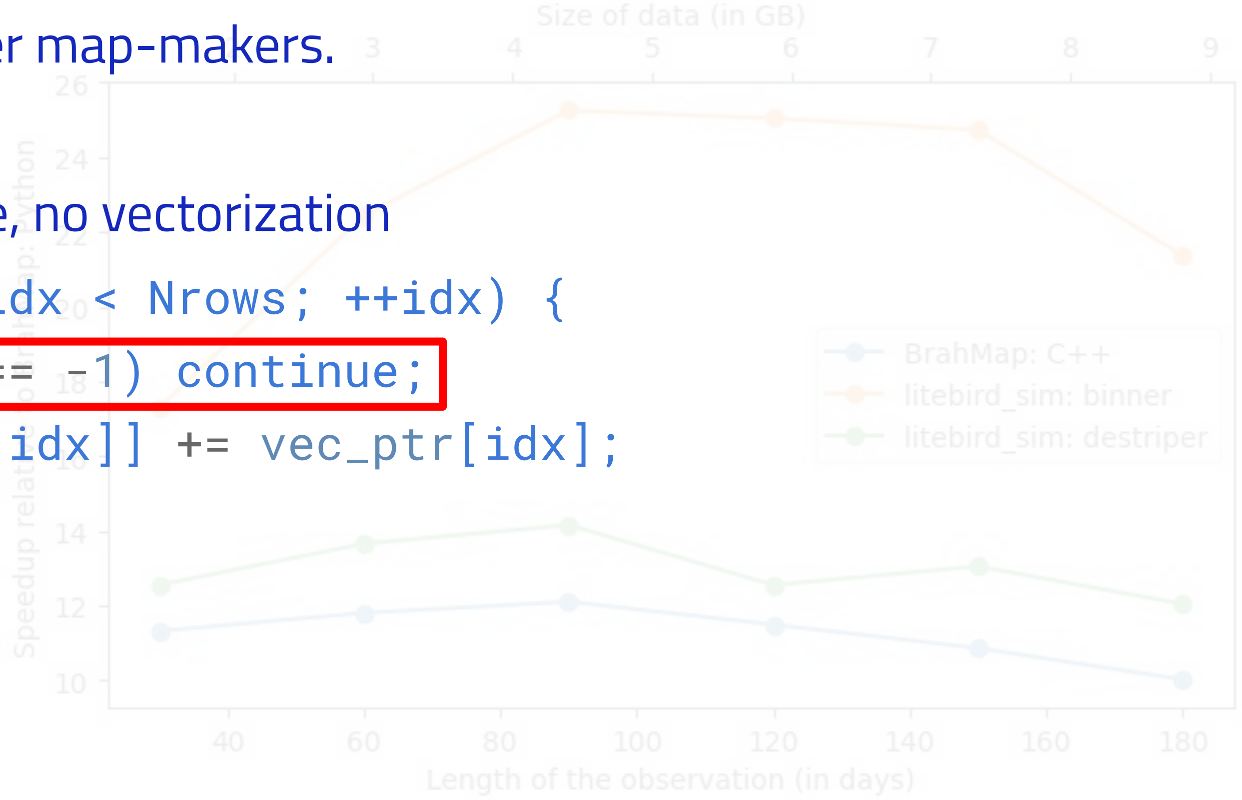


BrahMap: Code performance

BrahMap is still slower than other map-makers.

Reason: Inefficient loop structure, no vectorization

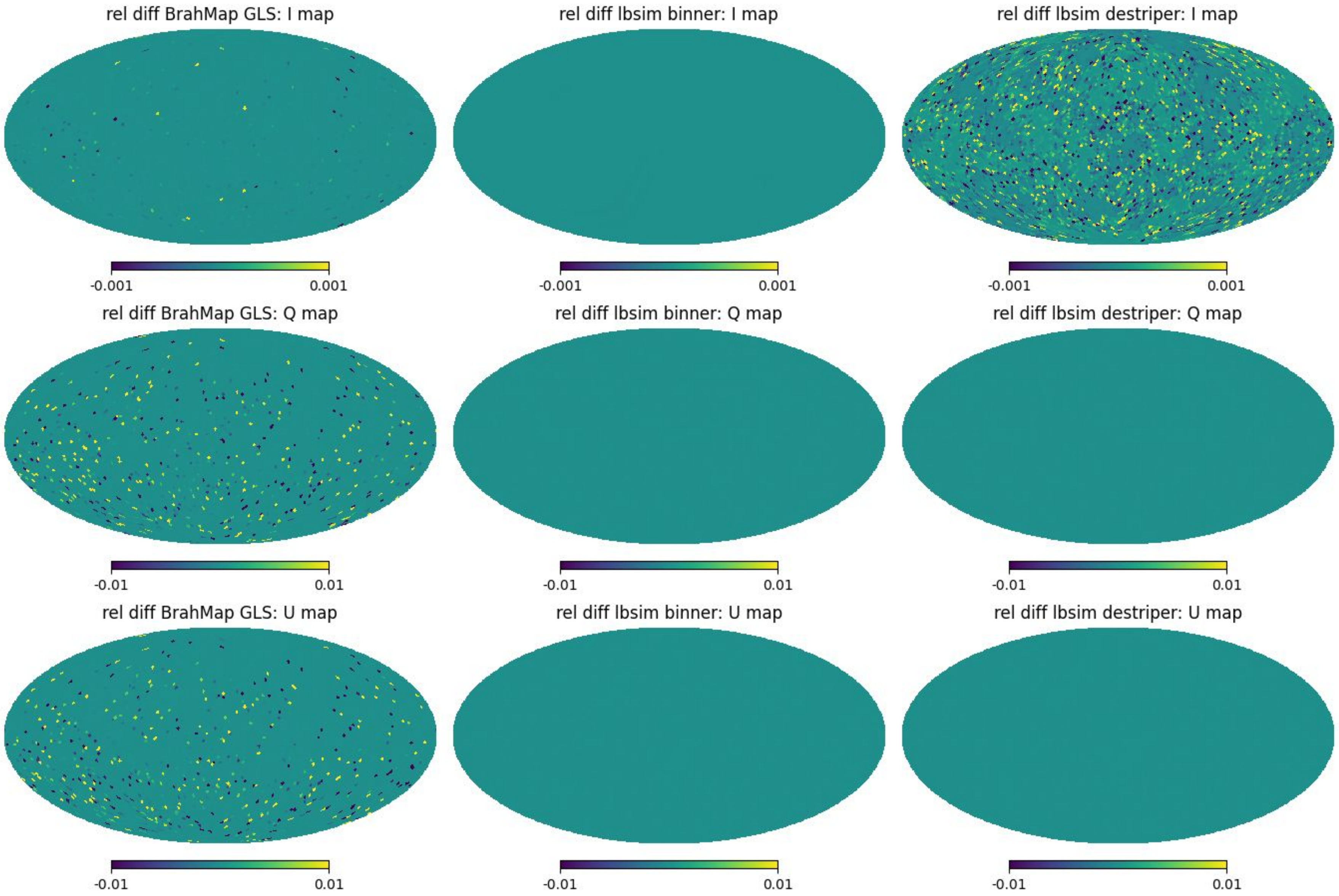
```
for (ssize_t idx = 0; idx < Nrows; ++idx) {
    if (pixs_ptr[idx] == -1) continue;
    x_arr_ptr[pixs_ptr[idx]] += vec_ptr[idx];
} // for
```



BrahMap: Validation

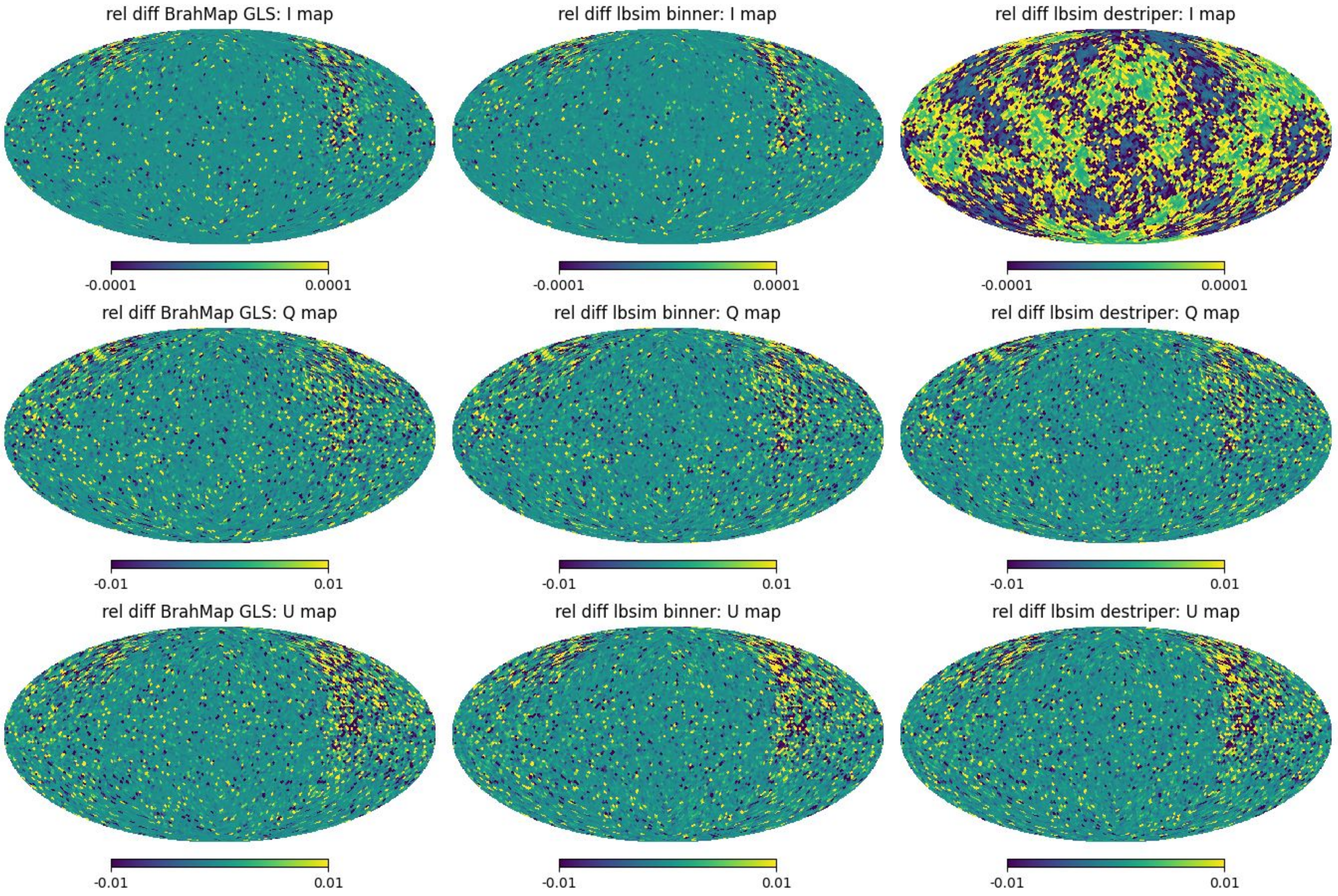
- Signal only data
  - Time-ordered data (TOD) obtained by scanning the CMB sky
- Signal + white noise
- Signal + *1/f* noise

rel. diff. = $\frac{\text{output\_map - input\_map}}{\text{input\_map}}$



BrahMap: Validation

- Signal only data
- Signal + white noise
  - TOD obtained by scanning CMB sky + 0.01 uK white noise level
- Signal + *1/f* noise

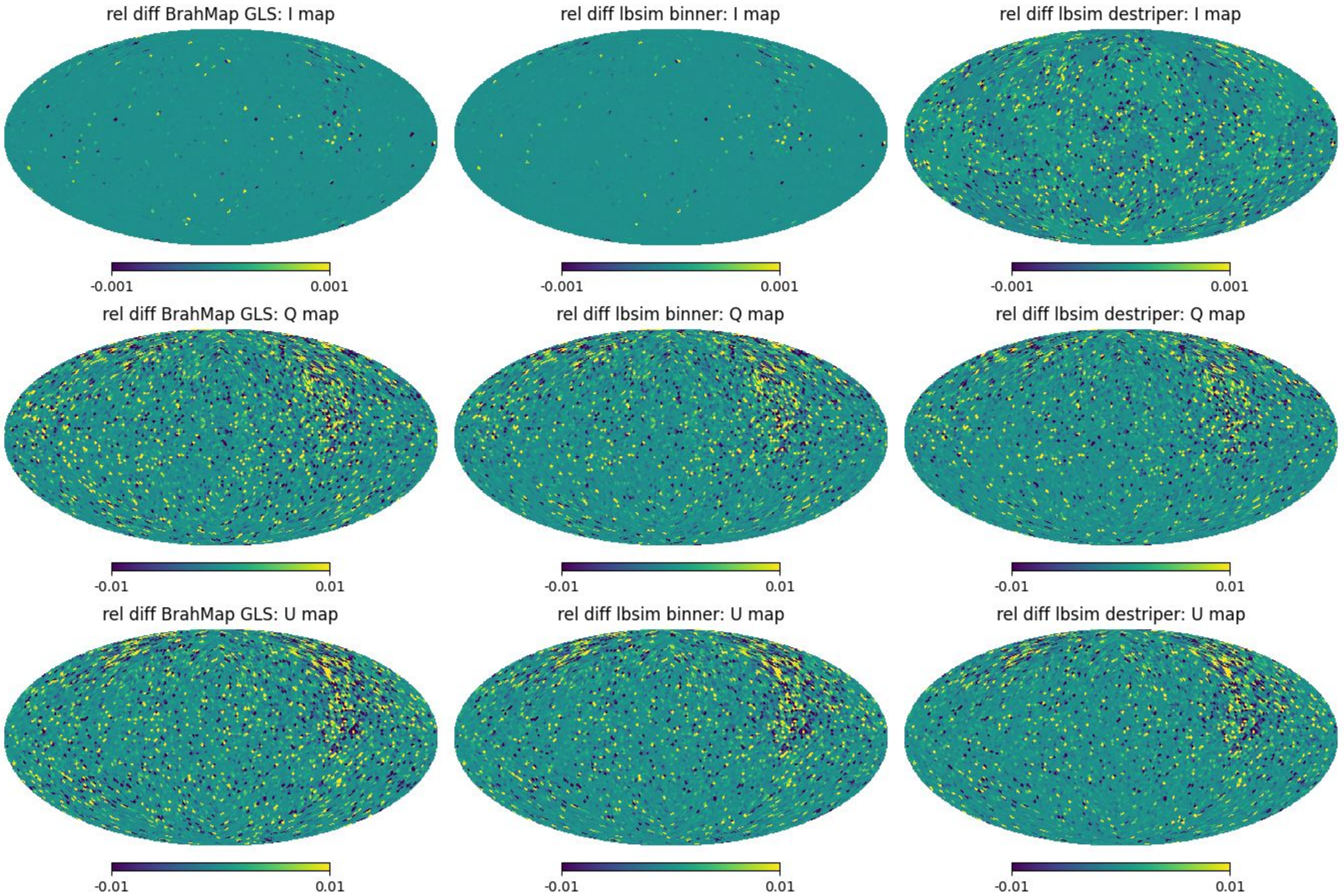


$$\text{rel. diff.} = \frac{\text{output\_map} - \text{input\_map}}{\text{input\_map}}$$

BrahMap: Validation

- Signal only data
- Signal + white noise
- Signal + *1/f* noise
  - TOD obtained by scanning CMB sky + 1/f noise at 0.01 uK level with f_{knee} = 20 mHz
  - 100 sec long baseline for destriper

$$\text{rel. diff.} = \frac{\text{output\_map} - \text{input\_map}}{\text{input\_map}}$$



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

BrahMap: Conclusion and future outlooks

- BrahMap offers object-oriented modular map-making framework
- Serial performance comparable to the existing codes, with possibility of further improvements
- Code repository: `https://github.com/anand-avinash/BrahMap`
- Documentation: `https://anand-avinash.github.io/BrahMap`

- Address the vectorization in the loop structures
- Take advantage of parallelization across multiple CPUs using MPI+OpenMP
- Offload the heavy computations to GPUs
- Implementing the cross-platform matrix equation solvers using MAGMA library