EXTREME SCALE COMPUTATIONAL IMAGING FOR RADIO ASTRONOMY AND MEDICAL IMAGING

Adrian Jackson

a.jackson@epcc.ed.ac.uk

@adrianjhpc

Prof Yves Wiaux

Heriot Watt

Imaging

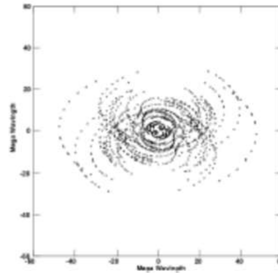
Aperture synthesis in radio interferometry images the sky at extreme resolutions and sensitivities.

- Each telescope pair probes the correlation of incoming electric fields from the source leading to Fourier measurements:

$$y(u) = \int_{S^2} G(\tau)x(\tau)e^{-2i\pi u\cdot\tau}\,d\tau \simeq \hat{x}(u)$$

- Fourier sampling is incomplete by construction

VLA



Fourier sampling

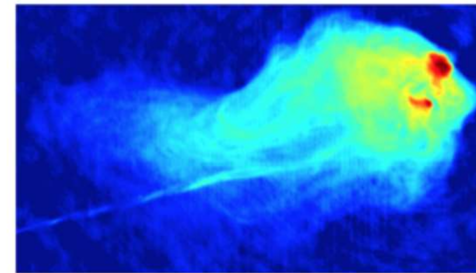


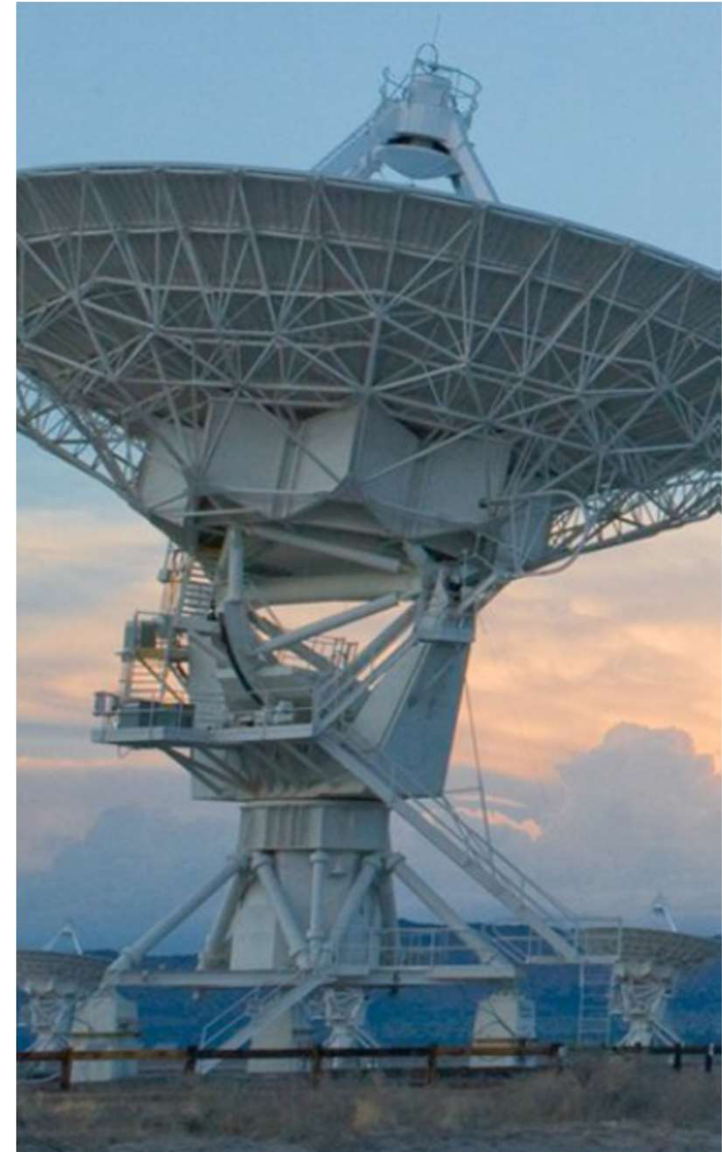
Image formed

- Image reconstruction algorithms required

epcc

High precision imaging

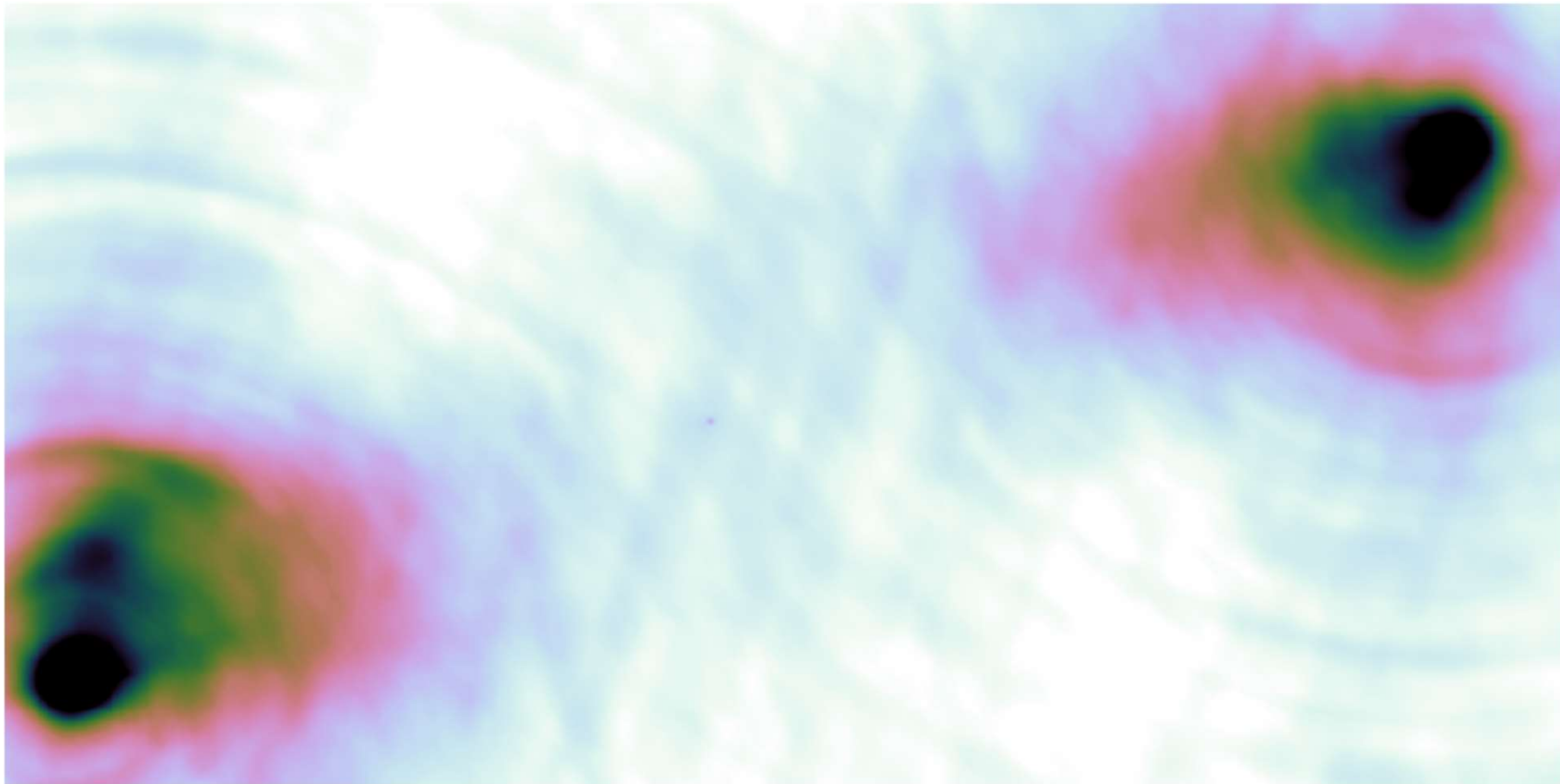
- Very Large Array Telescope
  - Cygnus A galaxy
- Data
  - 500MB size
  - 1M data points per frequency
  - 512 channels (4-8 GHz)
- Image
  - 15GB size
  - 2560 x 1536 x 512 channels
  - 0.06 arcsec spatial resolution
  - 8MHz spectal resolution



epcc

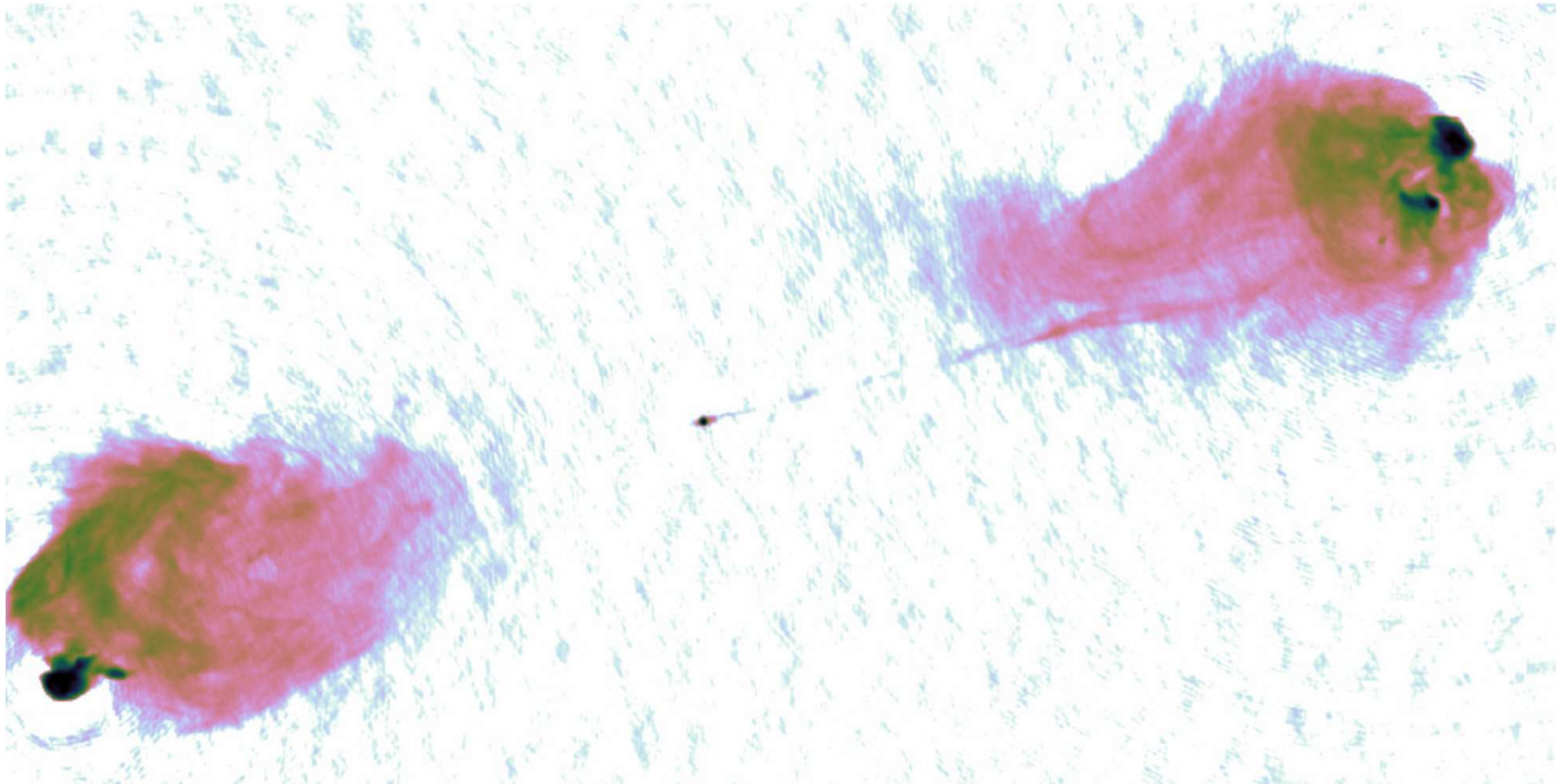
Low frequencies

- Dirty image (low resolution original)



Low frequencies

- CLEAN reconstruction

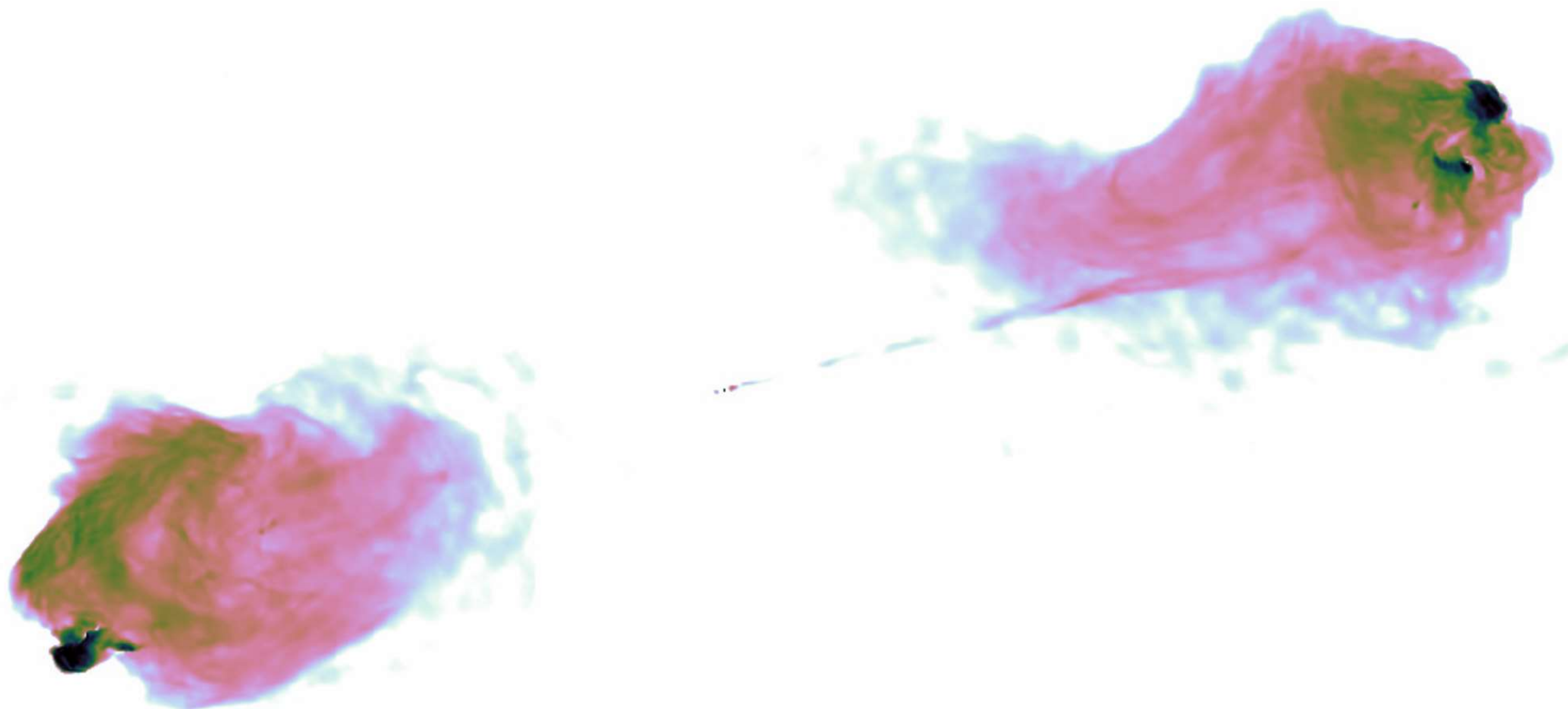


CLEAN

- Greedy non-linear deconvolution
  - Iteratively searching for atoms to be included in the solution.
  - Matching pursuit algorithm with greedy selection
  - Not easily parallel to large scale
  - Requires full data sets in single or small number of nodes
  - Suited to point sources
- SARA is Convex Optimisation algorithm
  - Sparsity-aware models for compressive sensing
  - Sparse representation in pre-defined set of dictionaries
  - Iterative reconstruction approach

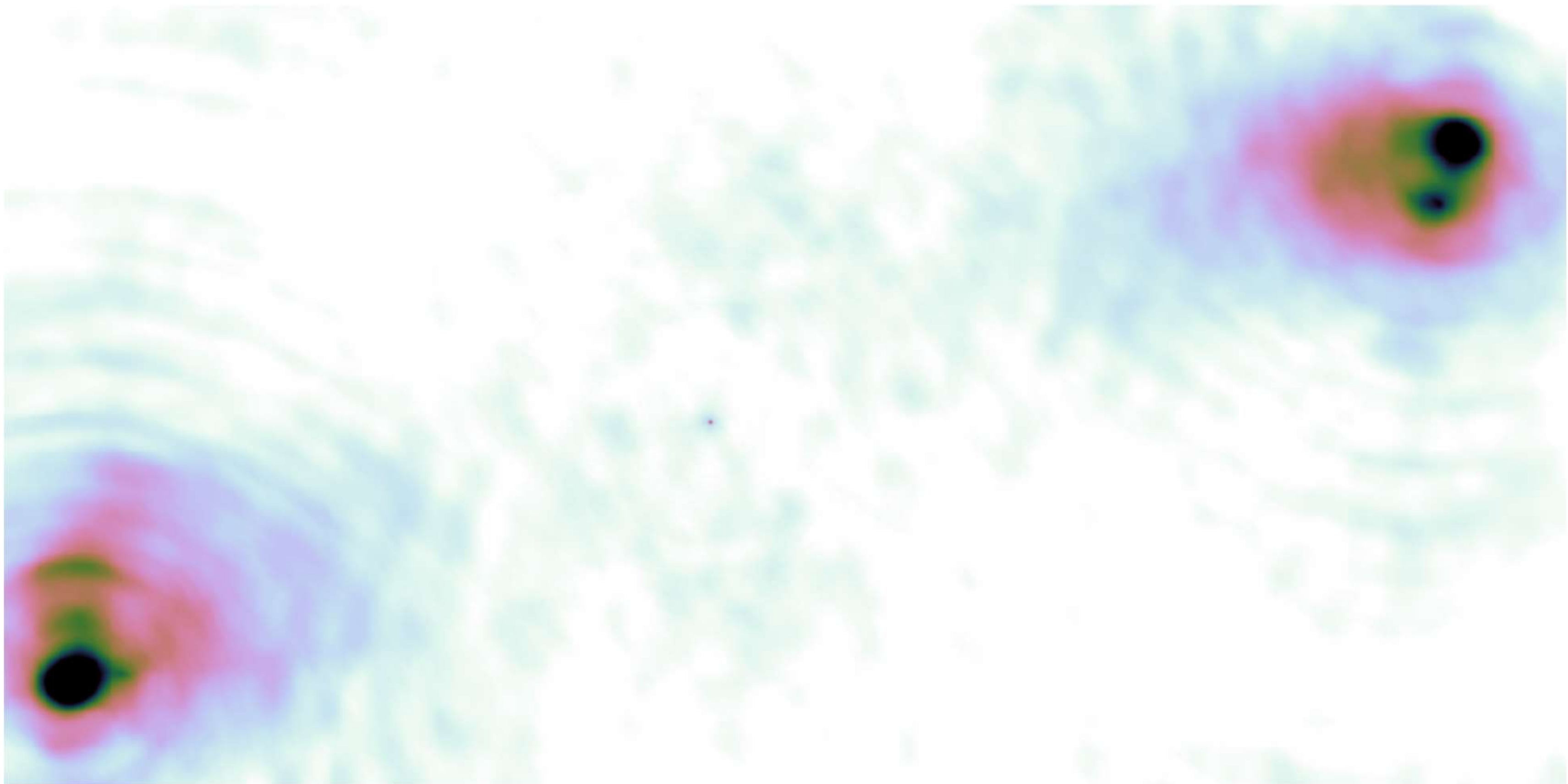
Low frequencies

- SARA reconstruction



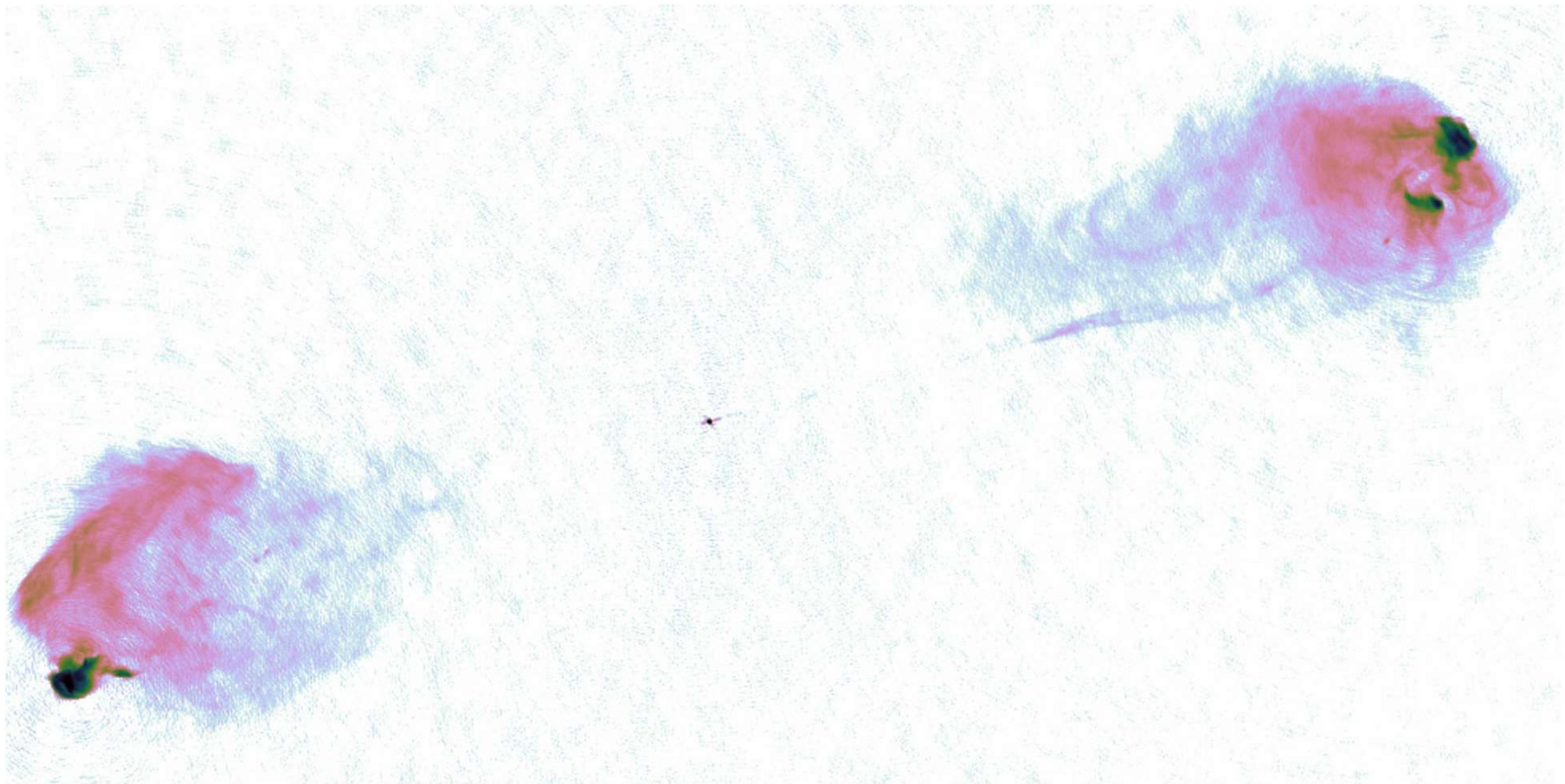
High frequencies

- Original dirty image



High frequencies

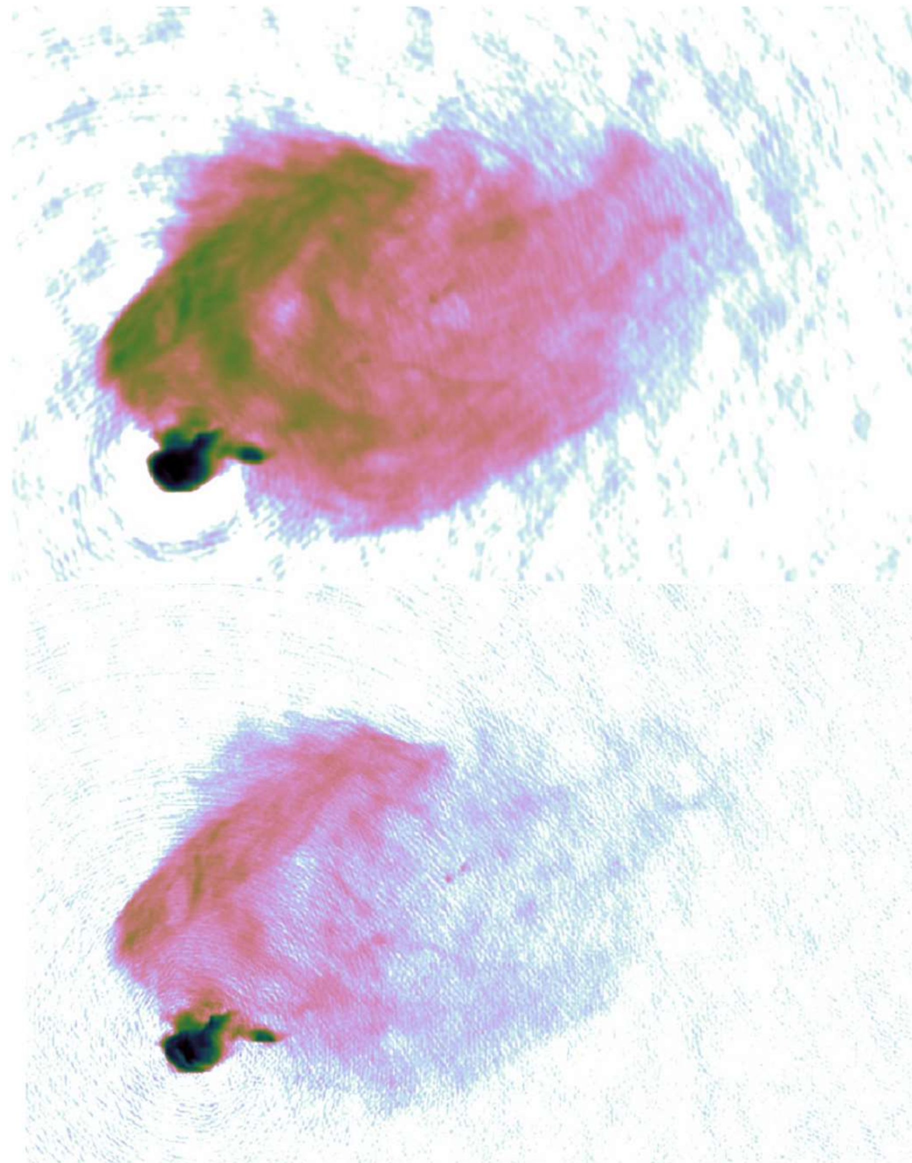
- CLEAN reconstruction

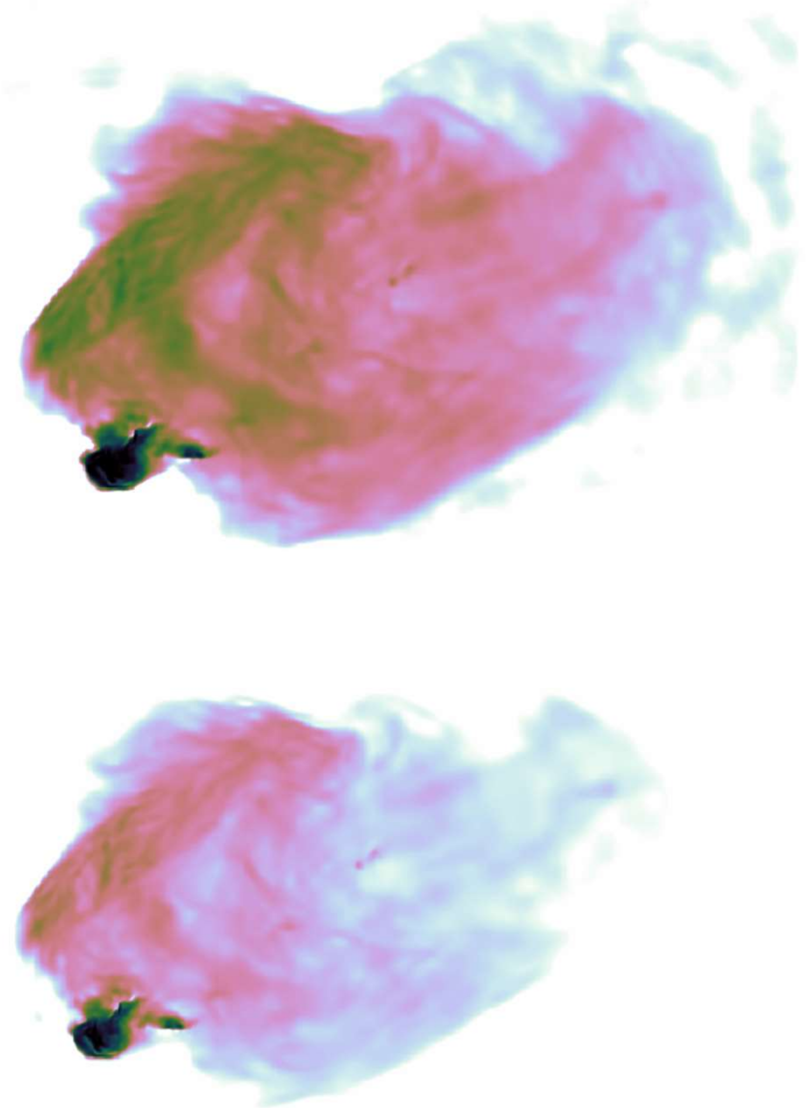


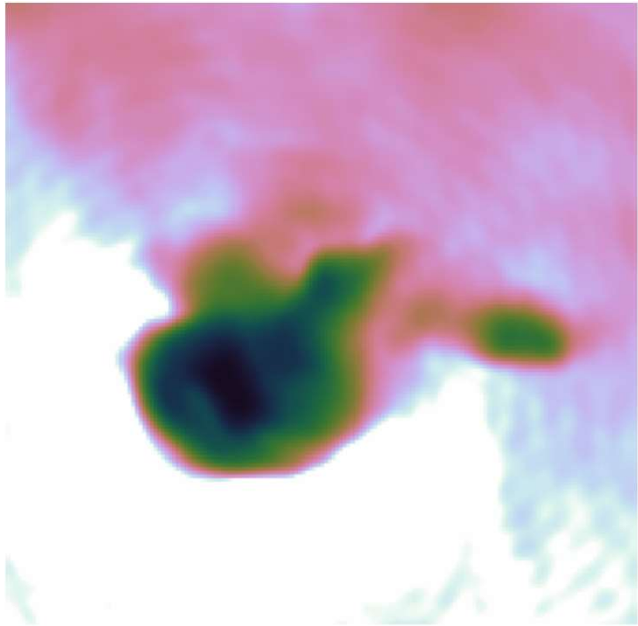
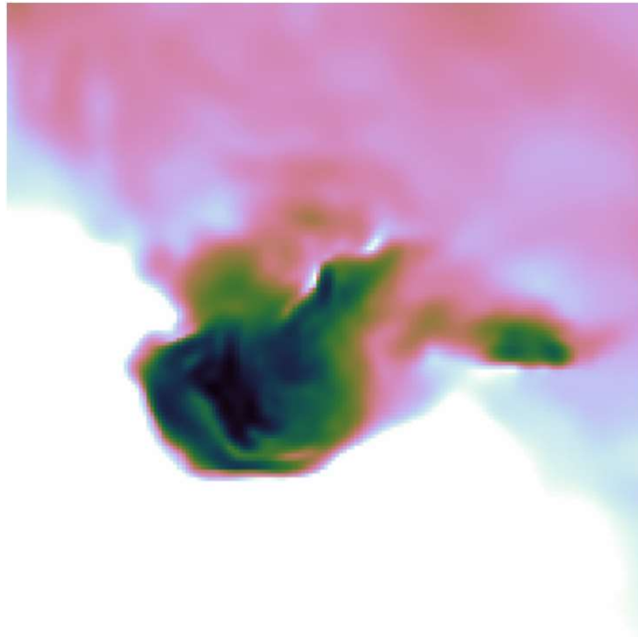
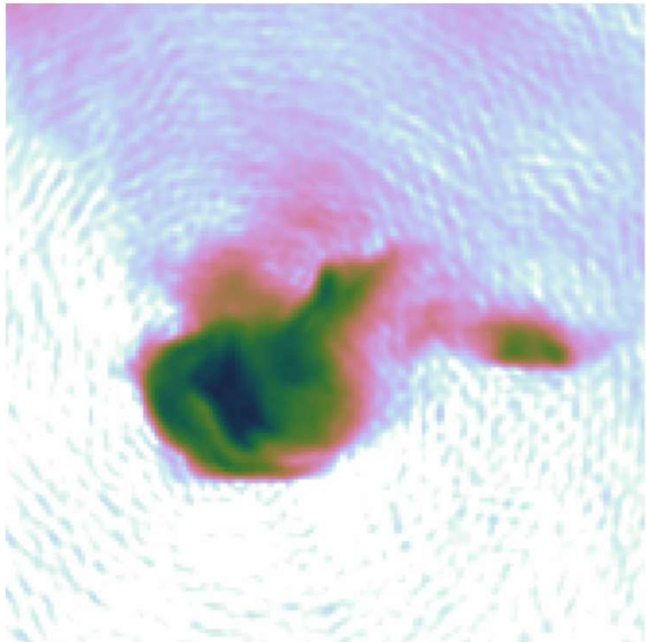
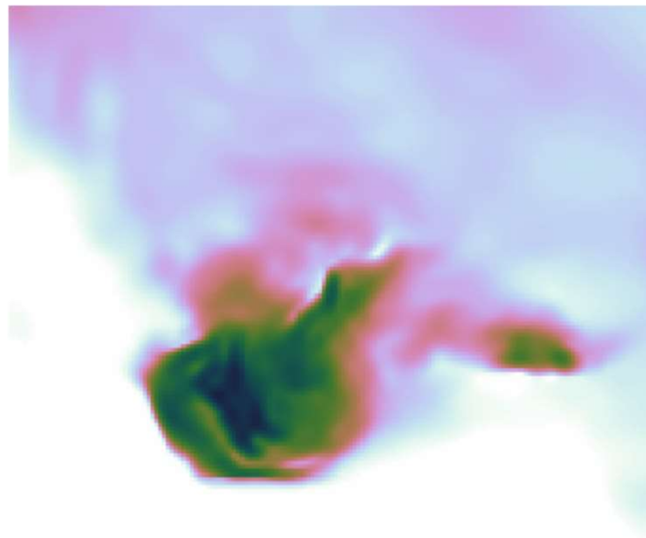
High frequencies

- SARA reconstruction

Left lobe

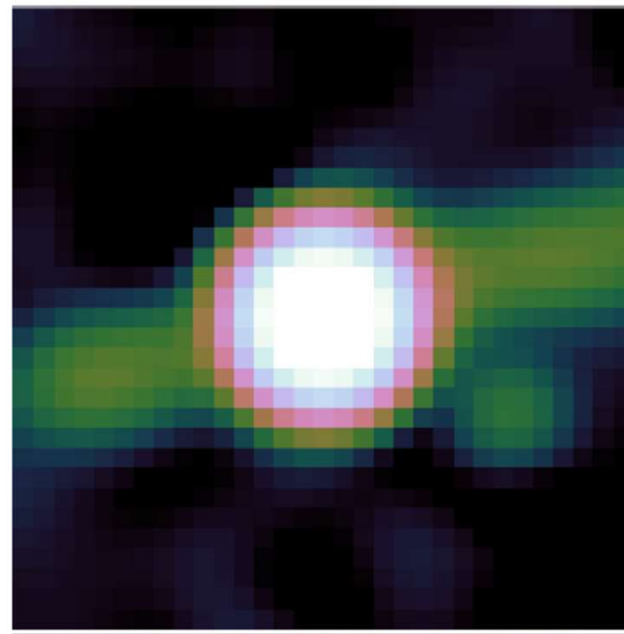


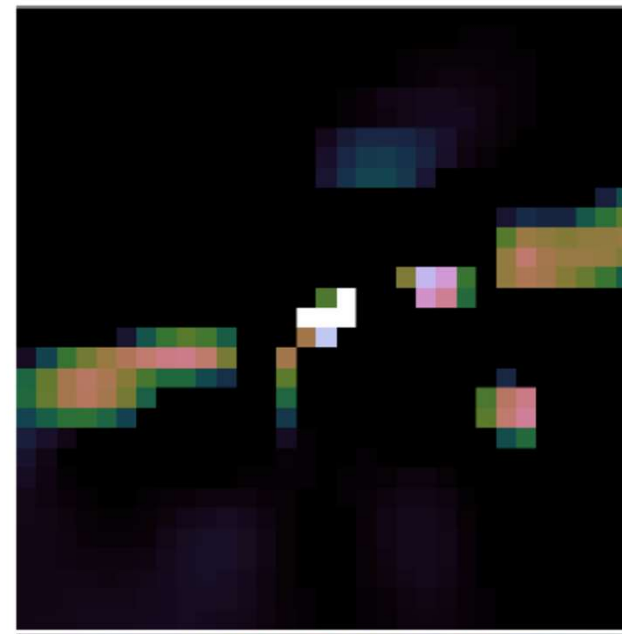


Science discoveries

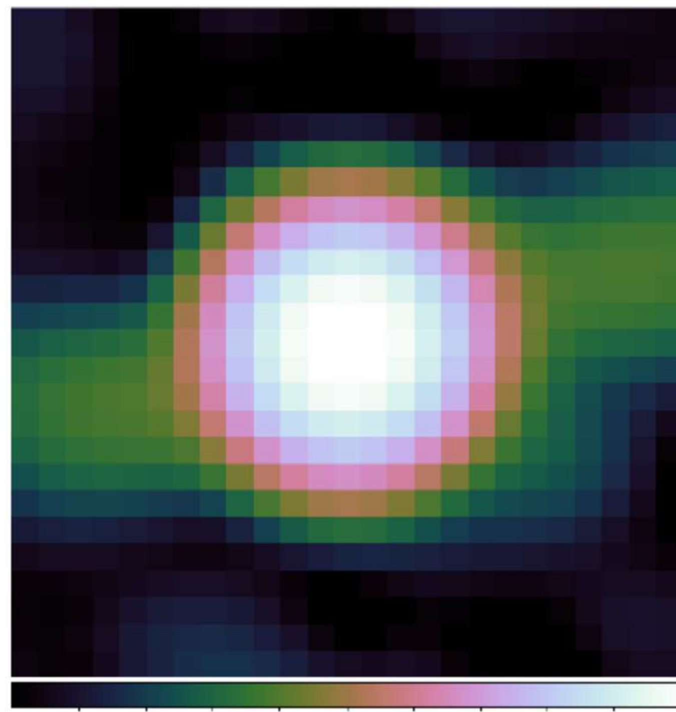
- Second black hole identified at 8.4GHz
  - (Perley et al.2017)

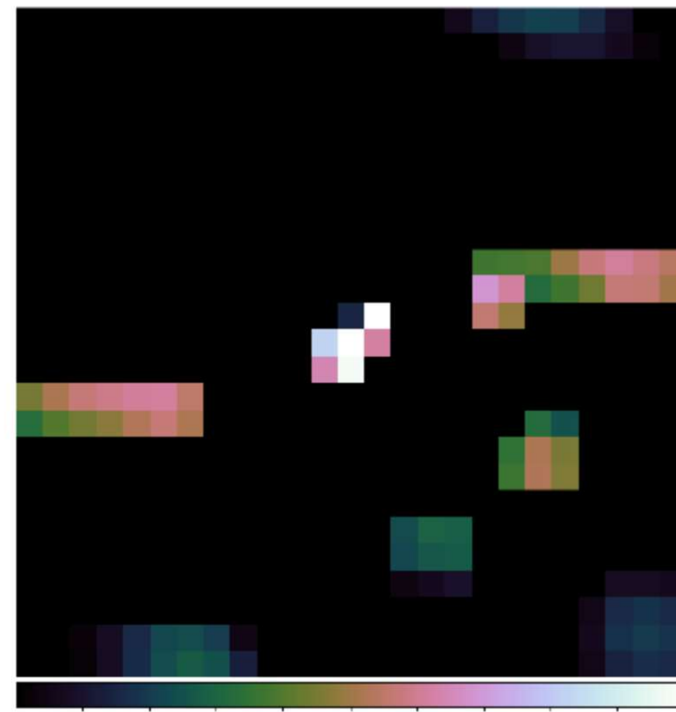




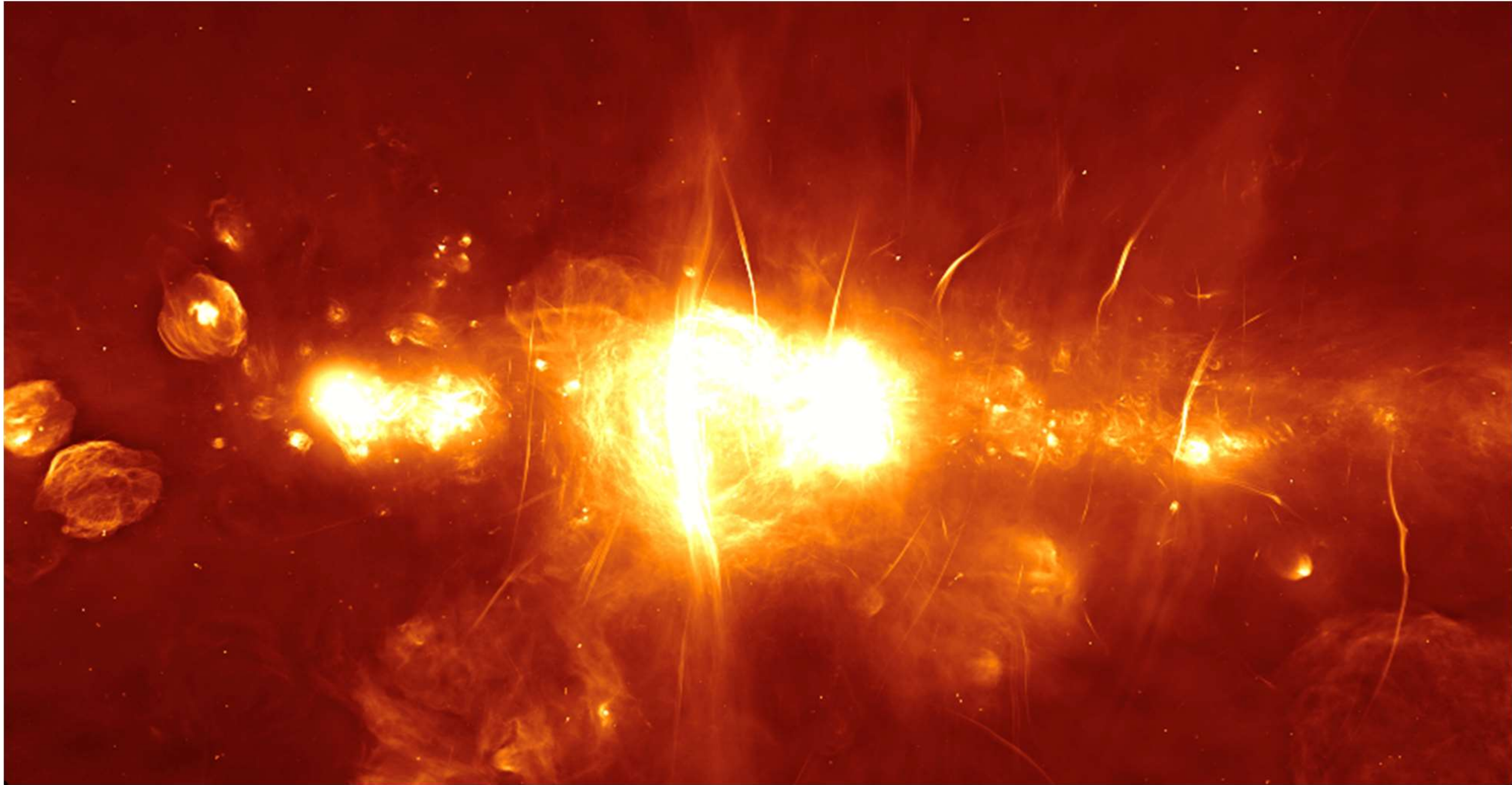
epcc

- Also visible at 6.6GHz for SARA





Meerkat – SKA precursor



SARA algorithm

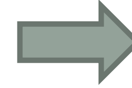
Algorithm 1 Randomised forward-backward PD.

1: **given** $\boldsymbol{x}^{(0)}, \tilde{\boldsymbol{x}}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{v}_j^{(0)}, \tilde{\boldsymbol{v}}_j^{(0)}, \kappa, \tau, \sigma, \varsigma$
2: **repeat for** $t = 1, \ldots$
3: $\quad \boldsymbol{v}^{(t)} = \boldsymbol{v}^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)} - \boldsymbol{\mathcal{P}}_{\mathcal{B}}\left(\boldsymbol{v}_i^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)}\right)$
4: $\quad \boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^{\dagger}\tilde{\boldsymbol{x}}^{(t-1)} - \boldsymbol{\mathcal{S}}_{\kappa\|\boldsymbol{\Psi}\|_{\mathrm{S}}}\left(\boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^{\dagger}\tilde{\boldsymbol{x}}^{(t-1)}\right)$
5: $\quad \boldsymbol{x}^{(t)} = \boldsymbol{\mathcal{P}}_{\mathcal{C}}\left(\boldsymbol{x}^{(t-1)} - \tau\left(\sigma\boldsymbol{\Psi}\boldsymbol{s}^{(t)} + \varsigma\boldsymbol{\Phi}^{\dagger}\boldsymbol{v}^{(t)}\right)\right)$
6: $\quad \tilde{\boldsymbol{x}}^{(t)} = 2\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}$
7: **until convergence**

Distributed memory parallelisation

Algorithm 1 Randomised forward-backward PD.

1: **given** $\boldsymbol{x}^{(0)}, \tilde{\boldsymbol{x}}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{v}_j^{(0)}, \tilde{\boldsymbol{v}}_j^{(0)}, \kappa, \tau, \sigma, \varsigma$
2: **repeat for** $t = 1, \ldots$
3: $\quad \boldsymbol{v}^{(t)} = \boldsymbol{v}^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)} - \mathcal{P}_{\mathcal{B}}\left(\boldsymbol{v}_i^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)}\right)$
4: $\quad \boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^{\dagger}\tilde{\boldsymbol{x}}^{(t-1)} - \mathcal{S}_{\kappa\|\boldsymbol{\Psi}\|_{\mathrm{S}}}\left(\boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^{\dagger}\tilde{\boldsymbol{x}}^{(t-1)}\right)$
5: $\quad \boldsymbol{x}^{(t)} = \mathcal{P}_{\mathcal{C}}\left(\boldsymbol{x}^{(t-1)} - \tau\left(\sigma\boldsymbol{\Psi}\boldsymbol{s}^{(t)} + \varsigma\boldsymbol{\Phi}^{\dagger}\boldsymbol{v}^{(t)}\right)\right)$
6: $\quad \tilde{\boldsymbol{x}}^{(t)} = 2\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}$
7: **until convergence**



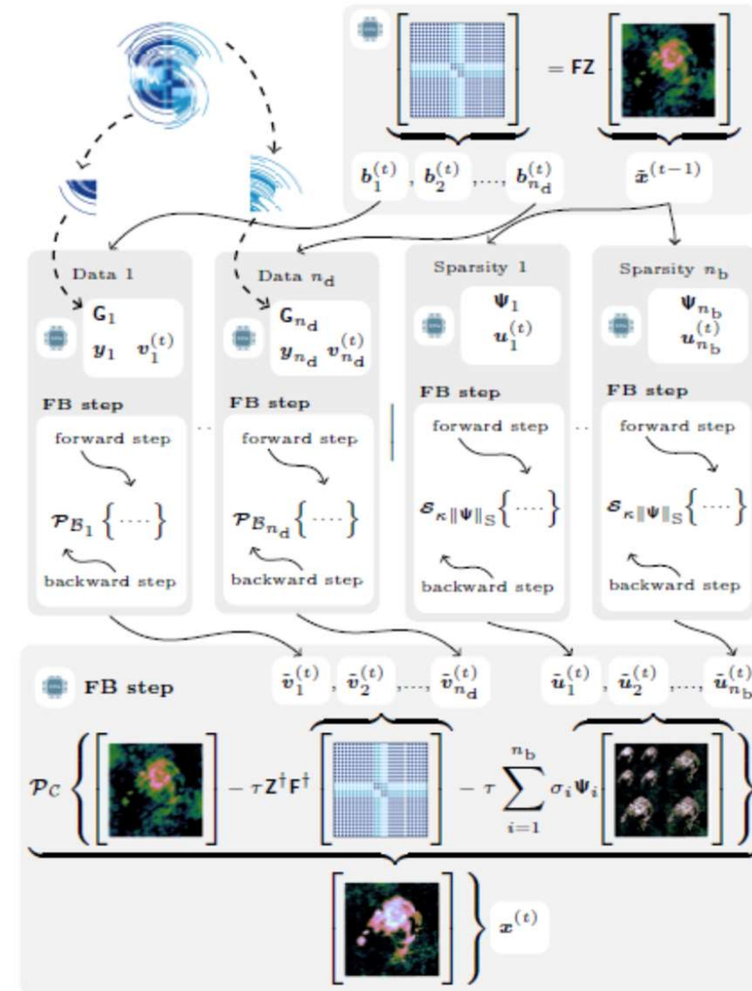
Algorithm 2 Randomised forward-backward PD.

1: **given** $\boldsymbol{x}^{(0)}, \bar{\boldsymbol{x}}^{(0)}, \boldsymbol{u}_i^{(0)}, \boldsymbol{v}_j^{(0)}, \bar{\boldsymbol{u}}_i^{(0)}, \bar{\boldsymbol{v}}_j^{(0)}, \kappa, \tau, \sigma_i, \varsigma_j, \lambda$
2: **repeat for** $t = 1, \ldots$
3: $\quad$ **generate sets** $\mathcal{P} \subset \{1, \ldots, n_{\mathrm{b}}\}$ **and** $\mathcal{D} \subset \{1, \ldots, n_{\mathrm{d}}\}$
4: $\quad \bar{\boldsymbol{b}}^{(t)} = \mathsf{FZ}\bar{\boldsymbol{x}}^{(t-1)}$
5: $\quad \forall j \in \mathcal{D}$ **set**
6: $\qquad \boldsymbol{b}_j^{(t)} = \mathsf{M}_j\bar{\boldsymbol{b}}^{(t)}$
7: $\quad$ **end**
8: $\quad$ **run simultaneously**
9: $\qquad \forall j \in \mathcal{D}$ **distribute** $\boldsymbol{b}_j^{(t)}$ **and do in parallel**
10: $\qquad\quad \bar{\boldsymbol{v}}_j^{(t)} = \left(\mathcal{I} - \mathcal{P}_{\mathcal{B}_j}\right)\left(\boldsymbol{v}_j^{(t-1)} + \mathsf{G}_j\boldsymbol{b}_j^{(t)}\right)$
11: $\qquad\quad \boldsymbol{v}_j^{(t)} = \boldsymbol{v}_j^{(t-1)} + \lambda\left(\bar{\boldsymbol{v}}_j^{(t)} - \boldsymbol{v}_j^{(t-1)}\right)$
12: $\qquad\quad \tilde{\boldsymbol{v}}_j^{(t)} = \mathsf{G}_j^{\dagger}\boldsymbol{v}_j^{(t)}$
13: $\qquad$ **end and gather** $\tilde{\boldsymbol{v}}_j^{(t)}$
14: $\qquad \forall j \in \{1, \ldots n_{\mathrm{d}}\} \setminus \mathcal{D}$ **set**
15: $\qquad\quad \boldsymbol{v}_j^{(t)} = \boldsymbol{v}_j^{(t-1)}$
16: $\qquad\quad \tilde{\boldsymbol{v}}_j^{(t)} = \tilde{\boldsymbol{v}}_j^{(t-1)}$
17: $\qquad$ **end**
18: $\qquad \forall i \in \mathcal{P}$ **do in parallel**
19: $\qquad\quad \bar{\boldsymbol{u}}_i^{(t)} = \left(\mathcal{I} - \mathcal{S}_{\kappa\|\Psi\|_{\mathrm{S}}}\right)\left(\boldsymbol{u}_i^{(t-1)} + \Psi_i^{\dagger}\bar{\boldsymbol{x}}^{(t-1)}\right)$
20: $\qquad\quad \boldsymbol{u}_i^{(t)} = \boldsymbol{u}_i^{(t-1)} + \lambda\left(\bar{\boldsymbol{u}}^{(t)} - \boldsymbol{u}_i^{(t-1)}\right)$
21: $\qquad\quad \tilde{\boldsymbol{u}}_i^{(t)} = \Psi_i\boldsymbol{u}_i^{(t)}$
22: $\qquad$ **end**
23: $\qquad \forall i \in \{1, \ldots n_{\mathrm{b}}\} \setminus \mathcal{P}$ **set**
24: $\qquad\quad \boldsymbol{u}_i^{(t)} = \boldsymbol{u}_i^{(t-1)}$
25: $\qquad\quad \tilde{\boldsymbol{u}}_i^{(t)} = \tilde{\boldsymbol{u}}_i^{(t-1)}$
26: $\qquad$ **end**
27: $\quad$ **end**
28: $\quad \bar{\boldsymbol{x}}^{(t)} = \mathcal{P}_{\mathcal{C}}\left(\boldsymbol{x}^{(t-1)} - \tau\left(\mathsf{Z}^{\dagger}\mathsf{F}^{\dagger}\sum_{j=1}^{n_{\mathrm{d}}}\varsigma_j\mathsf{M}_j^{\dagger}\tilde{\boldsymbol{v}}_j^{(t)} + \sum_{i=1}^{n_{\mathrm{b}}}\sigma_i\tilde{\boldsymbol{u}}_i^{(t)}\right)\right)$
29: $\quad \boldsymbol{x}^{(t)} = \boldsymbol{x}^{(t-1)} + \lambda\left(\bar{\boldsymbol{x}}^{(t)} - \boldsymbol{x}^{(t-1)}\right)$
30: $\quad \tilde{\boldsymbol{x}}^{(t)} = 2\bar{\boldsymbol{x}}^{(t)} - \boldsymbol{x}^{(t-1)}$
31: **until convergence**

Primal Dual

Algorithm 2 Randomised forward-backward PD.

1: **given** $x^{(0)}, \tilde{x}^{(0)}, u_i^{(0)}, v_j^{(0)}, \tilde{u}_i^{(0)}, \tilde{v}_j^{(0)}, \kappa, \tau, \sigma_i, \varsigma_j, \lambda$
2: **repeat for** $t = 1, \ldots$
3: **generate sets** $\mathcal{P} \subset \{1, \ldots, n_b\}$ **and** $\mathcal{D} \subset \{1, \ldots, n_d\}$
4: $\tilde{b}^{(t)} = \mathsf{FZ}\tilde{x}^{(t-1)}$
5: $\forall j \in \mathcal{D}$ **set**
6: $b_j^{(t)} = \mathsf{M}_j \tilde{b}^{(t)}$
7: **end**
8: **run simultaneously**
9: $\forall j \in \mathcal{D}$ **distribute** $b_j^{(t)}$ **and do in parallel**
10: $\bar{v}_j^{(t)} = \left(\mathcal{I} - \mathcal{P}_{\mathcal{B}_j}\right)\left(v_j^{(t-1)} + \mathsf{G}_j b_j^{(t)}\right)$
11: $v_j^{(t)} = v_j^{(t-1)} + \lambda\left(\bar{v}_j^{(t)} - v_j^{(t-1)}\right)$
12: $\tilde{v}_j^{(t)} = \mathsf{G}_j^{\dagger} v_j^{(t)}$
13: **end and gather** $\tilde{v}_j^{(t)}$
14: $\forall j \in \{1, \ldots n_d\} \setminus \mathcal{D}$ **set**
15: $v_j^{(t)} = v_j^{(t-1)}$
16: $\tilde{v}_j^{(t)} = \tilde{v}_j^{(t-1)}$
17: **end**
18: $\forall i \in \mathcal{P}$ **do in parallel**
19: $\bar{u}_i^{(t)} = \left(\mathcal{I} - \mathcal{S}_{\kappa\|\Psi\|_S}\right)\left(u_i^{(t-1)} + \Psi_i^{\dagger}\tilde{x}^{(t-1)}\right)$
20: $u_i^{(t)} = u_i^{(t-1)} + \lambda\left(\bar{u}^{(t)} - u_i^{(t-1)}\right)$
21: $\tilde{u}_i^{(t)} = \Psi_i u_i^{(t)}$
22: **end**
23: $\forall i \in \{1, \ldots n_b\} \setminus \mathcal{P}$ **set**
24: $u_i^{(t)} = u_i^{(t-1)}$
25: $\tilde{u}_i^{(t)} = \tilde{u}_i^{(t-1)}$
26: **end**
27: **end**
28: $\bar{x}^{(t)} = \mathcal{P}_C\left(x^{(t-1)} - \tau\left(\mathsf{Z}^{\dagger}\mathsf{F}^{\dagger}\sum_{j=1}^{n_d}\varsigma_j \mathsf{M}_j^{\dagger}\tilde{v}_j^{(t)} + \sum_{i=1}^{n_b}\sigma_i \tilde{u}_i^{(t)}\right)\right)$
29: $x^{(t)} = x^{(t-1)} + \lambda\left(\bar{x}^{(t)} - x^{(t-1)}\right)$
30: $\tilde{x}^{(t)} = 2\bar{x}^{(t)} - x^{(t-1)}$
31: **until convergence**



Parallel Primal Dual

- Five stage parallelisation possibilities
  - Shared memory for individual loops
  - Distribute wavelets (image data)
  - Distribute measurement operator (visibility data)
    - Distribute channels (frequencies)
    - Distribute blocks within a frequency (time blocks)
    - Split blocks

Parallel Primal Dual

- Data distribution

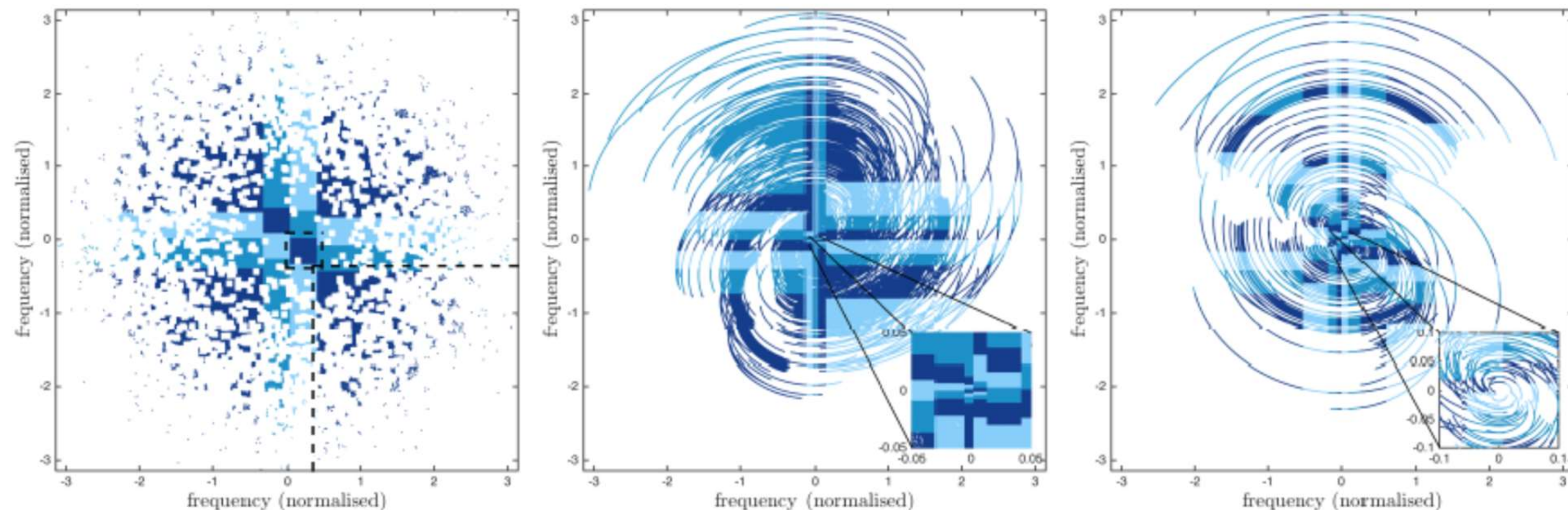
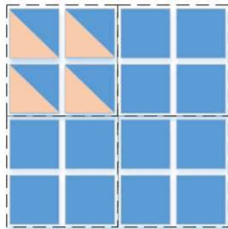
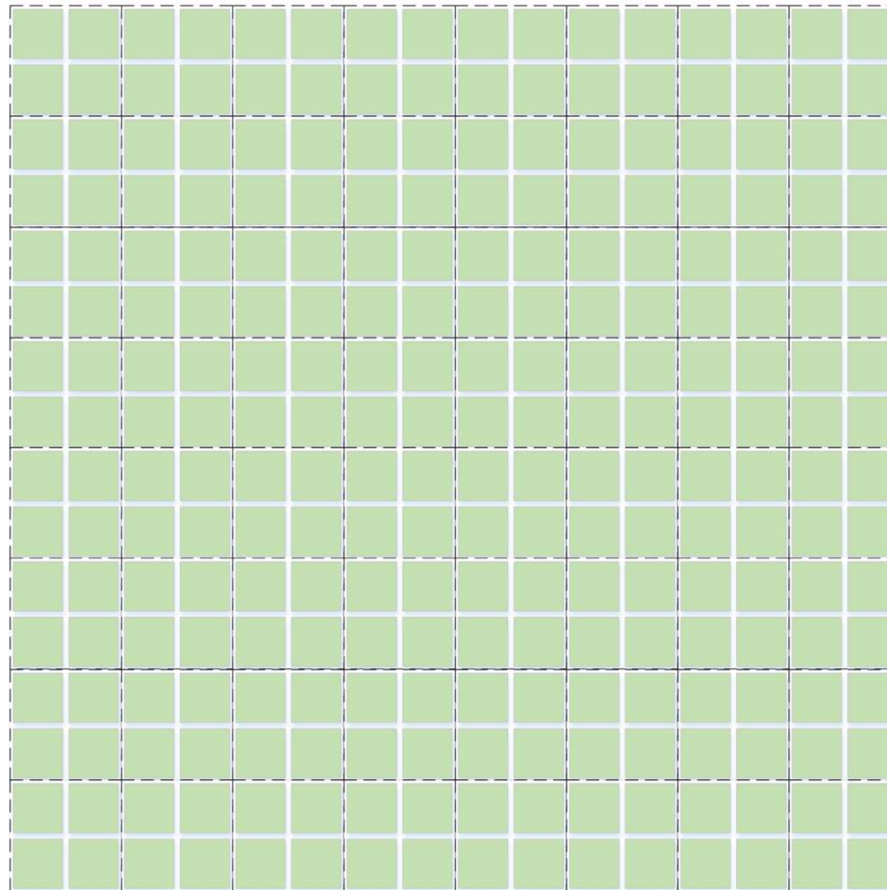
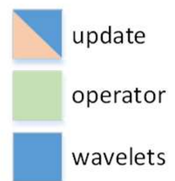


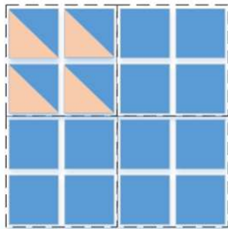
Figure 4. (left) An example of randomly generated coverage with the number of visibilities $M = 655360$. The visibilities are split into 16 equal size blocks, marked with different colours, with compact u–v grouping. The dashed lines mark the parts of the discrete Fourier space involved in the computations associated with the central-bottom-right and the bottom-right blocks, respectively. In this case, the whole discrete frequency space is considered to have 512×512 points. (centre) The SKA u–v coverage for 5 hours of observation corresponding to $M = 5791800$. (right) The VLA u–v coverage for 9 hours of observations corresponding to $M = 1788480$. The SKA and VLA data are split into 64 blocks containing an equal number of visibilities.

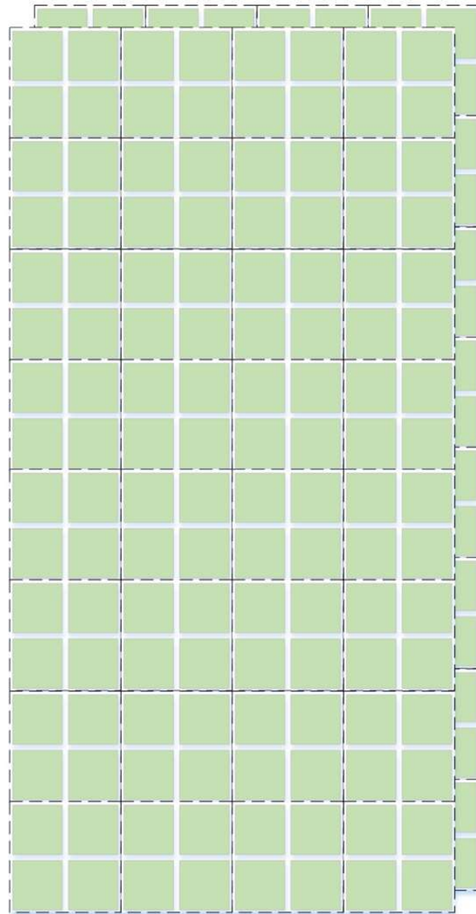
Distributions



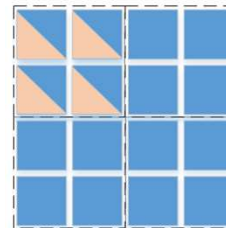


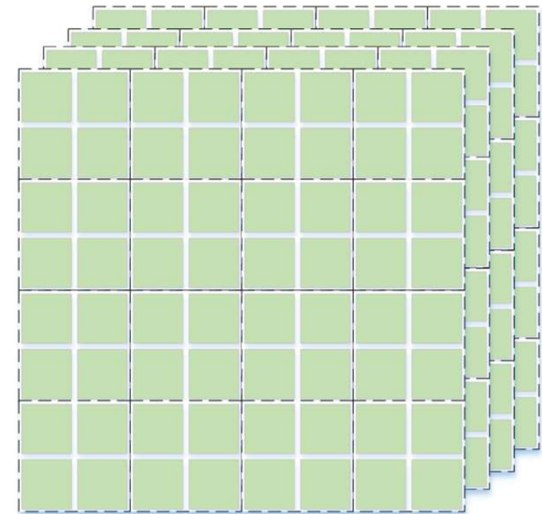






update
operator
wavelets



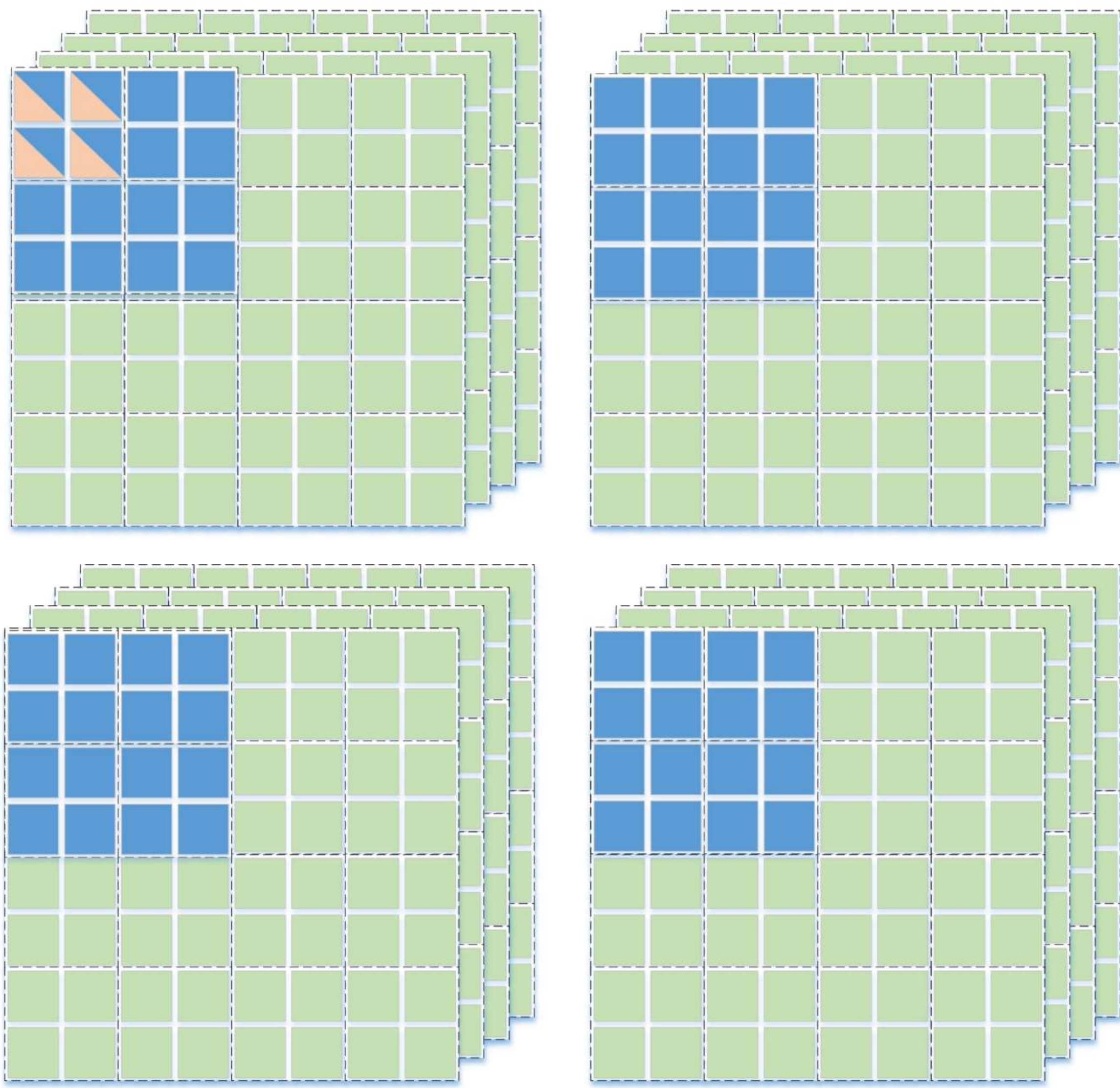


update
operator
wavelets

update

operator

wavelets



epcc

Domain decomposition

- Balanced and adaptable domain decomposition key for performance
  - Different simulations have
    - Different channel numbers
    - Different blocks per channel
    - Different visibilities per block
  - Current simulations have
    - Larger images than visibilities
    - Cygnus A -> 0.5 GB visibilities to 15 GB image
  - Future simulations will have
    - Much larger visibilities than images
    - SKA -> 1TB visibilities to 1-10GB image

Implementation

- C++ framework
  - Templated implementation
    - Enables different operators and algorithms to be used
  - Two levels of functionality
    - Core algorithmic functions: `psi`
    - Application of algorithms: `puri-psi`

Primal Dual – solver

```
template <class SCALAR> void PrimalDual<SCALAR>::iteration_step(t_Vector &out, t_Vector &residual, t_Vector &s, t_Vector &v, t_Vector &x_bar) const {
  t_Vector prev_sol = out;
  t_Vector prev_s = s;
  t_Vector prev_v = v;
  auto const l2ball_proximal = proximal::L2Ball<Scalar>(l2ball_epsilon());
  // v_t = v_t-1 + Phi*x_bar - l2ball_prox(v_t-1 + Phi*x_bar)
  t_Vector temp = v + (Phi() * x_bar);
  t_Vector v_prox = l2ball_proximal(0, temp-target()) + target();
  v = temp - v_prox;
  // s_t = s_t-1 + Psi_dagger * x_bar_t-1 - l1norm_prox(s_t-1 + Psi_dagger * x_bar_t-1)
  t_Vector temp2 = s + (Psi().adjoint() * x_bar);
  t_Vector s_prox;
  proximal::l1_norm(s_prox, kappa()/sigma1(), temp2);
  s = temp2 - s_prox;
  //x_t = positive orth projection(x_t-1 - tau * (sigma1 * Psi * s + sigma2 * Phi dagger * v))
  out = prev_sol - tau()*(Psi()*s*sigma1() + Phi().adjoint()*v*sigma2());
  if(positivity_constraint()){
    out = sopt::positive_quadrant(out);
  }
  x_bar = 2*out - prev_sol;
  residual = Phi() * out - target();
}
```

Algorithm 1 Randomised forward-backward PD.

1: **given** $\boldsymbol{x}^{(0)}, \tilde{\boldsymbol{x}}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{v}_j^{(0)}, \tilde{\boldsymbol{v}}_j^{(0)}, \kappa, \tau, \sigma, \varsigma$
2: **repeat for** $t = 1, \ldots$
3: $\quad \boldsymbol{v}^{(t)} = \boldsymbol{v}^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)} - \mathcal{P}_{\mathcal{B}}\left(\boldsymbol{v}_i^{(t-1)} + \boldsymbol{\Phi}\tilde{\boldsymbol{x}}^{(t-1)}\right)$
4: $\quad \boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^\dagger\tilde{\boldsymbol{x}}^{(t-1)} - \mathcal{S}_{\kappa\|\boldsymbol{\Psi}\|_S}\left(\boldsymbol{s}^{(t-1)} + \boldsymbol{\Psi}^\dagger\tilde{\boldsymbol{x}}^{(t-1)}\right)$
5: $\quad \boldsymbol{x}^{(t)} = \mathcal{P}_{\mathcal{C}}\left(\boldsymbol{x}^{(t-1)} - \tau\left(\sigma\boldsymbol{\Psi}\boldsymbol{s}^{(t)} + \varsigma\boldsymbol{\Phi}^\dagger\boldsymbol{v}^{(t)}\right)\right)$
6: $\quad \tilde{\boldsymbol{x}}^{(t)} = 2\boldsymbol{x}^{(t)} - \boldsymbol{x}^{(t-1)}$
7: **until convergence**

epcc

Primal Dual – solver convergence

```
std::pair<Real, Real> objectives{sopt::l1_norm(Psi().adjoint() * out, l1_weights), 0};

for(; niters < itermax(); ++niters) {

    iteration_step(out, residual, s, v, x_bar);

    objectives.second = objectives.first;

    objectives.first = sopt::l1_norm(Psi().adjoint() * out, l1_weights);

    t_real const relative_objective = std::abs(objectives.first - objectives.second) /
objectives.first;

    auto const residual_norm = sopt::l2_norm(residual, weights);

    auto const user = (not has_user_convergence) or is_converged(out);

    auto const res = residual_convergence() <= 0e0 or residual_norm < residual_convergence();

    auto const rel = relative_variation() <= 0e0 or relative_objective < relative_variation();

    converged = user and res and rel;

    if(converged) break;

}
```

Primal Dual – measurement operator

```
sopt::Vector<t_complex> rand = sopt::Vector<t_complex>::Random(measurements->imsizey() * measurements->imsizex() * nlevels);

auto const pm = sopt::algorithm::PowerMethod<sopt::t_complex>().tolerance(1e-6);

auto const nu1data = pm.AtA(Psi, rand);

auto const nu1 = nu1data.magnitude.real();

auto sigma1 = 1e0 / nu1;

rand = sopt::Vector<t_complex>::Random(measurements->imsizey() * measurements->imsizex() * (over_sample/2));

auto const nu2data = pm.AtA(measurements_transform, rand);

auto const nu2 = nu2data.magnitude.real();

auto sigma2 = 1e0 / nu2;

auto const kappa = ((measurements_transform.adjoint() * uv_data.vis).real().maxCoeff() * 1e-3) / nu2;

auto const pd = sopt::algorithm::PrimalDual<t_complex>(uv_data.vis)

                       .tau(tau)

                       .kappa(kappa)

                       .sigma1(sigma1)

                       .sigma2(sigma2)

                       .levels(nlevels)

                       .l2ball_epsilon(epsilon)

                       .nu(nu2)

                       .relative_variation(1e-5)

                       .positivity_constraint(true)

                       .residual_convergence(epsilon * 1.001)

                       .Psi(Psi)

                      .itermax(500)

                      .is_converged(convergence_function)

                      .Phi(measurements_transform);
```

epcc

CygA

Code

- MATLAB and Python algorithms available
- C++ code in progress

Puri-Psi

Introduction
Matlab
C++ / Python
Referencing
License
Authors

Introduction

Puri-Psi is an open-source code developed within the **BASP group (Biomedical and Astronomical Signal Processing)** headed by Prof. Wiaux that provides functionality to address radio and optical interferometric imaging problems using state-of-the-art optimisation.

The name **Puri-Psi** builds from former software **Purify** by Prof. Wiaux and collaborators, with the extension -Psi standing for "Parallel Proximal Scalable Imaging".

Puri-Psi is currently available as a collection of independent **Matlab toolboxes**. A **C++ library** with partial Python support is currently under development and will be provided in due course. For further details on the methods currently implemented, please see the **reference section**.

Puri-Psi is made available under the license described **below**.

Matlab

Puri-Psi is currently composed of several independent Matlab toolboxes devoted to specific radio and optical interferometric imaging imaging problems. Further details on each of these methods can be directly found on their respective Github repository, which can be separately accessed by clicking on the following links.

Radio interferometric imaging

SARA - Scalable monochromatic imaging

- https://basp-group.github.io/Puri-Psi/

epcc