Matrix Completion in Parallel Architectures: A Julia Implementation

Oscar López, Keegan Lensing, Rajiv Kumar, Henryk Modzelewski




SLIM
University of British Columbia

Motivation

- Industry-scale seismic data interpolation

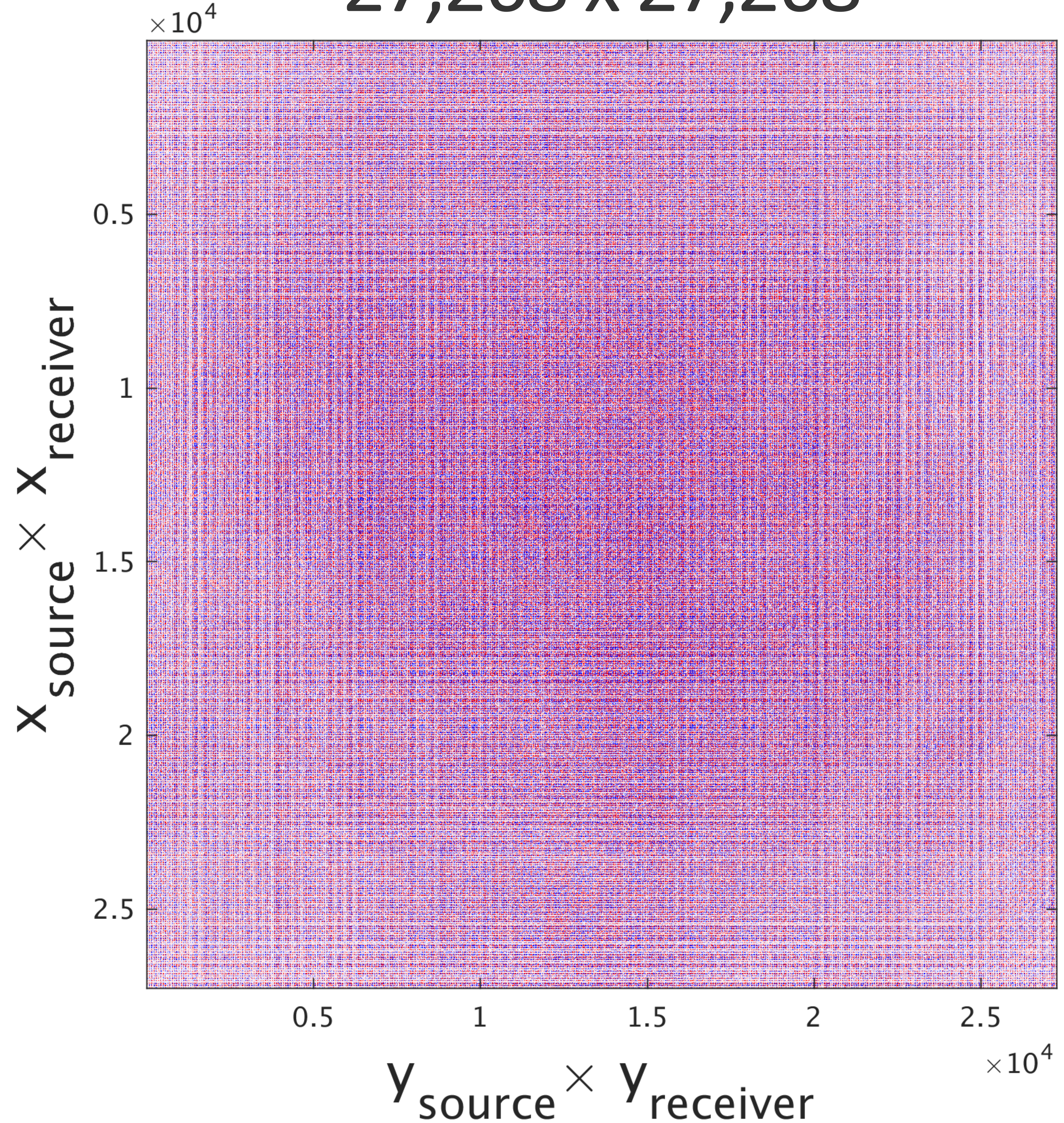
- Exploit *low-rank* structure of seismic data
  - matrix completion techniques

- Need to improve time complexity
  - design for parallel architectures

Motivation

Matricized 3D Frequency slice

27,268 x 27,268



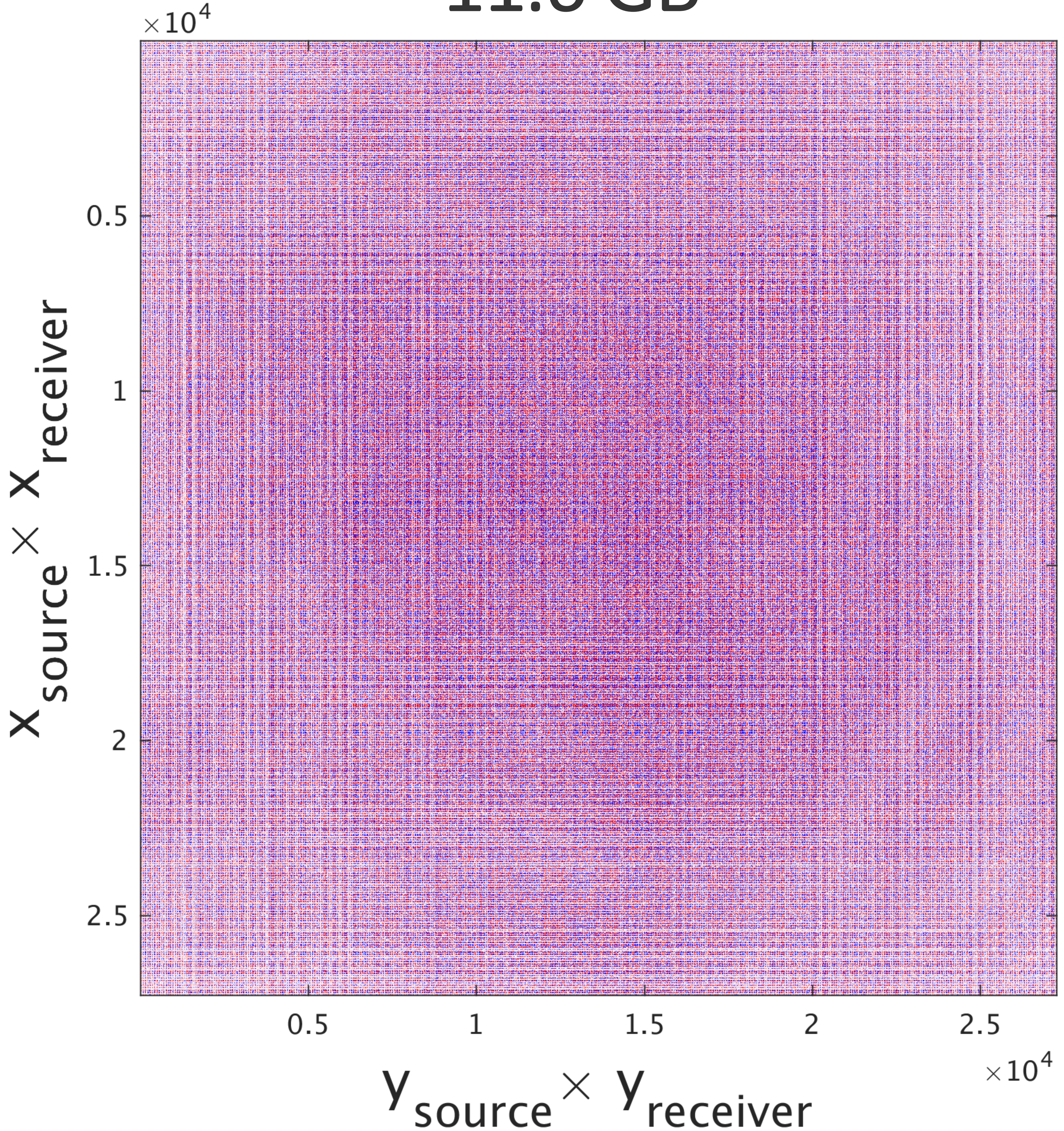
Seismic data: huge matrices

Interpolation quality deteriorates when working on smaller windows

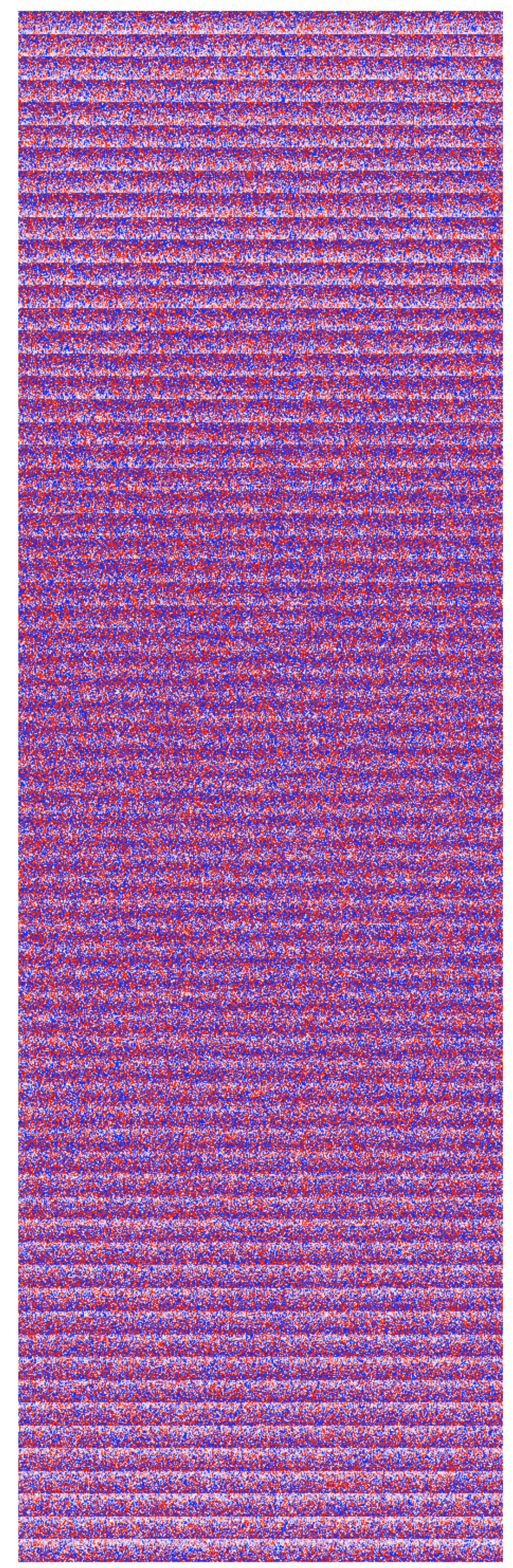
Want to work w/ full matrix

Motivation

11.6 GB

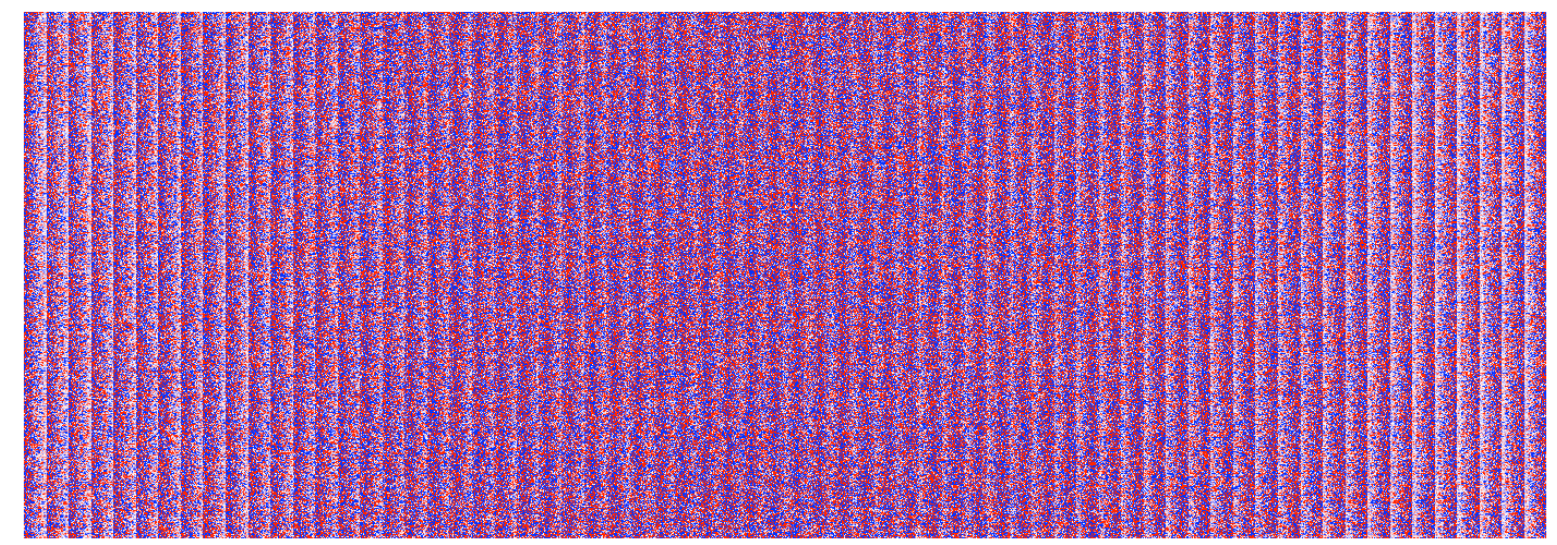


$\approx$



454 MB

$\times$



No need to store full matrix (96% compression)

Can directly generate gathers

SLIM

Contributions

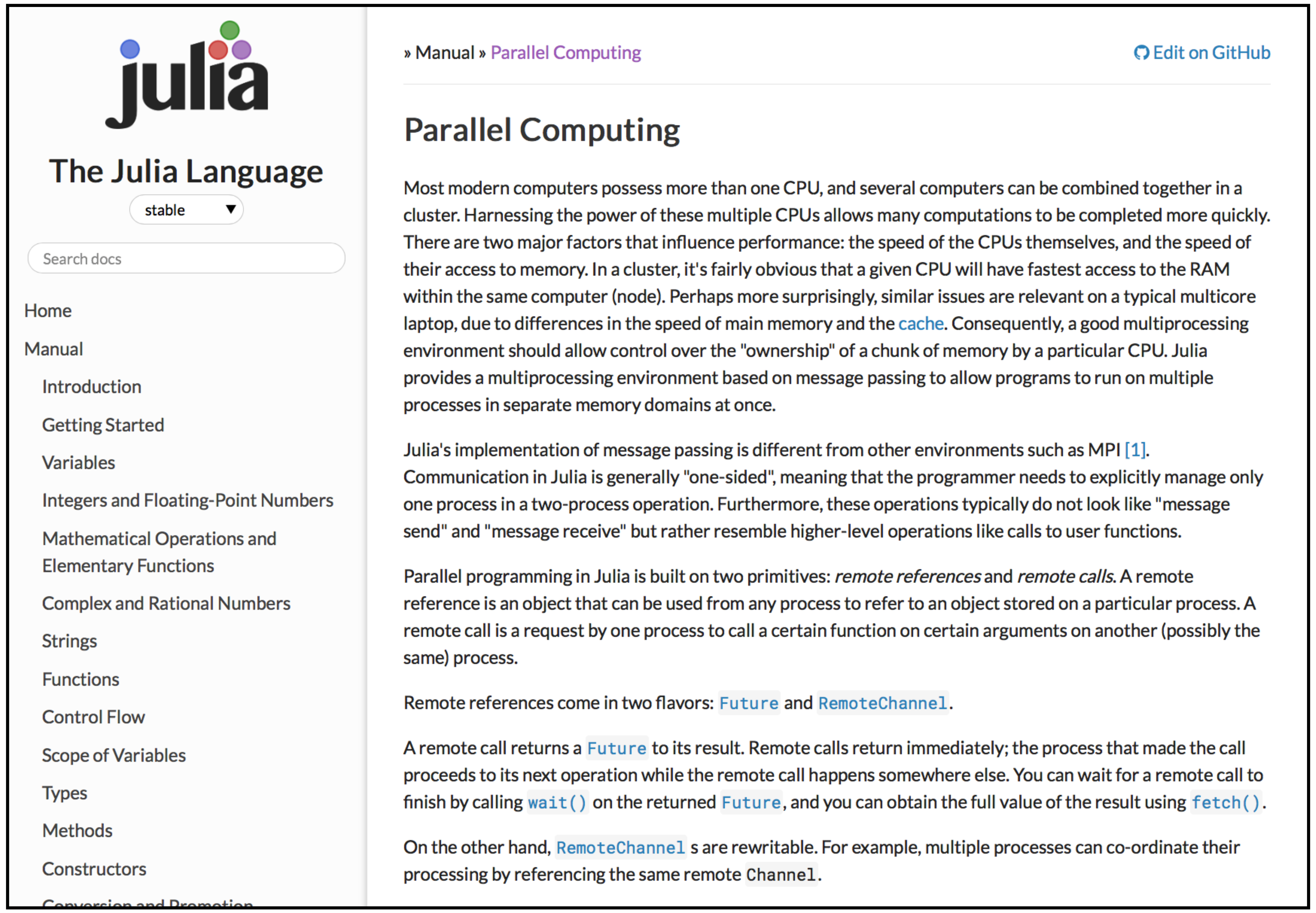
Matrix completion: Decoupling method

- decompose into independent subproblems
- solve in parallel architectures

Julia implementation:

- efficient multiprocessing environment
- optimize communication time between workers

Contributions


julia

The Julia Language

stable

Search docs

Home

Manual

Introduction

Getting Started

Variables

Integers and Floating-Point Numbers

Mathematical Operations and Elementary Functions

Complex and Rational Numbers

Strings

Functions

Control Flow

Scope of Variables

Types

Methods

Constructors

Conversion and Promotion

» Manual » Parallel Computing

Edit on GitHub

Parallel Computing

Most modern computers possess more than one CPU, and several computers can be combined together in a cluster. Harnessing the power of these multiple CPUs allows many computations to be completed more quickly. There are two major factors that influence performance: the speed of the CPUs themselves, and the speed of their access to memory. In a cluster, it's fairly obvious that a given CPU will have fastest access to the RAM within the same computer (node). Perhaps more surprisingly, similar issues are relevant on a typical multicore laptop, due to differences in the speed of main memory and the cache. Consequently, a good multiprocessing environment should allow control over the "ownership" of a chunk of memory by a particular CPU. Julia provides a multiprocessing environment based on message passing to allow programs to run on multiple processes in separate memory domains at once.

Julia's implementation of message passing is different from other environments such as MPI [1]. Communication in Julia is generally "one-sided", meaning that the programmer needs to explicitly manage only one process in a two-process operation. Furthermore, these operations typically do not look like "message send" and "message receive" but rather resemble higher-level operations like calls to user functions.

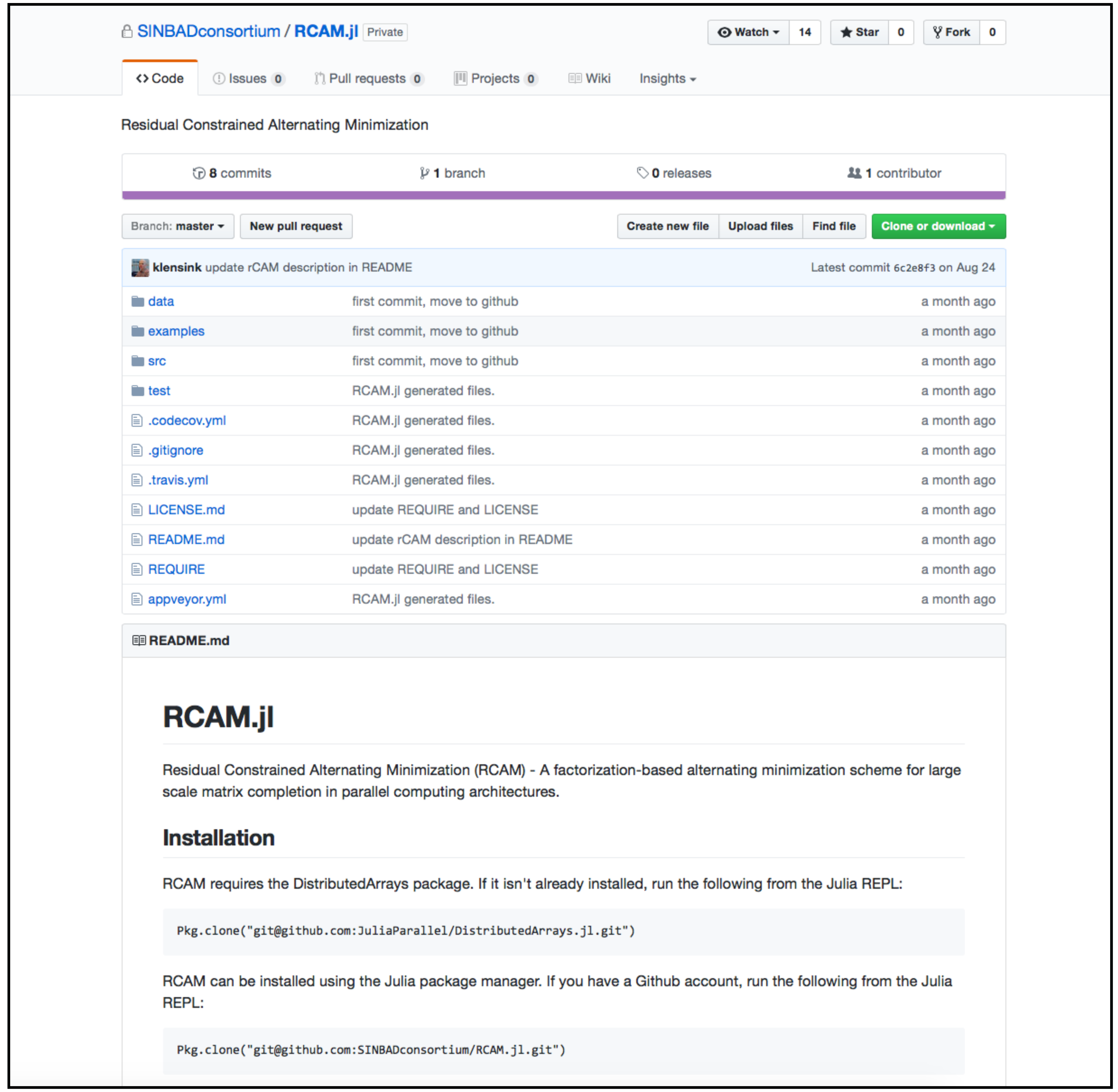
Parallel programming in Julia is built on two primitives: *remote references* and *remote calls*. A remote reference is an object that can be used from any process to refer to an object stored on a particular process. A remote call is a request by one process to call a certain function on certain arguments on another (possibly the same) process.

Remote references come in two flavors: `Future` and `RemoteChannel`.

A remote call returns a `Future` to its result. Remote calls return immediately; the process that made the call proceeds to its next operation while the remote call happens somewhere else. You can wait for a remote call to finish by calling `wait()` on the returned `Future`, and you can obtain the full value of the result using `fetch()`.

On the other hand, `RemoteChannel` s are rewritable. For example, multiple processes can co-ordinate their processing by referencing the same remote `Channel`.

Software Release Available


SINBADconsortium / RCAM.jl Private

Watch 14 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Residual Constrained Alternating Minimization

8 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

klensink update rCAM description in README Latest commit 6c2e8f3 on Aug 24

data	first commit, move to github	a month ago
examples	first commit, move to github	a month ago
src	first commit, move to github	a month ago
test	RCAM.jl generated files.	a month ago
.codecov.yml	RCAM.jl generated files.	a month ago
.gitignore	RCAM.jl generated files.	a month ago
.travis.yml	RCAM.jl generated files.	a month ago
LICENSE.md	update REQUIRE and LICENSE	a month ago
README.md	update rCAM description in README	a month ago
REQUIRE	update REQUIRE and LICENSE	a month ago
appveyor.yml	RCAM.jl generated files.	a month ago

README.md

RCAM.jl

Residual Constrained Alternating Minimization (RCAM) - A factorization-based alternating minimization scheme for large scale matrix completion in parallel computing architectures.

Installation

RCAM requires the DistributedArrays package. If it isn't already installed, run the following from the Julia REPL:

```
Pkg.clone("git@github.com:JuliaParallel/DistributedArrays.jl.git")
```

RCAM can be installed using the Julia package manager. If you have a Github account, run the following from the Julia REPL:

```
Pkg.clone("git@github.com:SINBADconsortium/RCAM.jl.git")
```

Outline

- Matrix completion
  - alternating least squares
  - decoupling method
- Parallel implementation in Julia
- Numerical experiments

Matrix completion

Goal is to approximate $\mathbf{M} \in \mathbb{C}^{n\times m}$, given

observed entries $\Omega \subset \{1,2,...,n\} \times \{1,2,...,m\}$

via

$$\mathbf{b}_{k,\ell} = P_{\Omega}(\mathbf{M})_{k,\ell} = \begin{cases} \mathbf{M}_{k,\ell} & \text{if } (k,\ell) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

SLIM

Methodology: Least Squares

If $\mathbf{M}$ is approximately rank-r, we solve

$$(\mathbf{L}^\sharp, \mathbf{R}^\sharp) = \underset{\mathbf{L}\in\mathbb{C}^{n\times r}, \mathbf{R}\in\mathbb{C}^{m\times r}}{\arg\min} \|P_\Omega(\mathbf{L}\mathbf{R}^*) - \mathbf{b}\|_F$$

and approximate

$$\mathbf{L}^\sharp(\mathbf{R}^\sharp)^* \approx \mathbf{M}$$

Prateek Jain, Praneeth Netrapalli, Sujay Sanghavi. "Low-Rank Matrix Completion Using Alternating Minimization".

SLIM

Methodology: Alternating Least Squares

We alternate the optimization over each factor

$$\mathbf{L}^t = \underset{\mathbf{L}\in\mathbb{C}^{n\times r}}{\arg\min} \|P_\Omega(\mathbf{L}(\mathbf{R}^t)^*) - \mathbf{b}\|_F$$

$$\mathbf{R}^{t+1} = \underset{\mathbf{R}\in\mathbb{C}^{m\times r}}{\arg\min} \|P_\Omega(\mathbf{L}^t\mathbf{R}^*) - \mathbf{b}\|_F$$

starting at initial factor ($\mathbf{R}^0$) and iteratively obtain

$$\mathbf{L}^T(\mathbf{R}^T)^* \approx \mathbf{M}$$

Prateek Jain, Praneeth Netrapalli, Sujay Sanghavi. "Low-Rank Matrix Completion Using Alternating Minimization".

SLIM

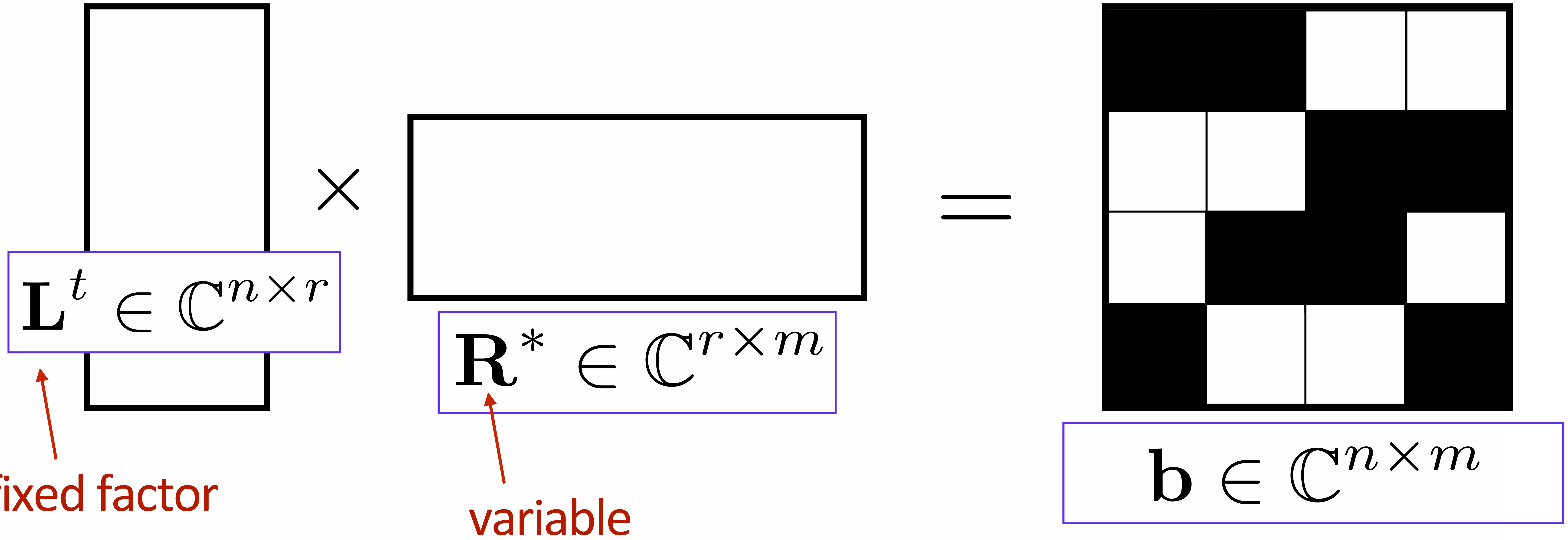
Methodology: Decoupling method

Each subproblem, e.g.,

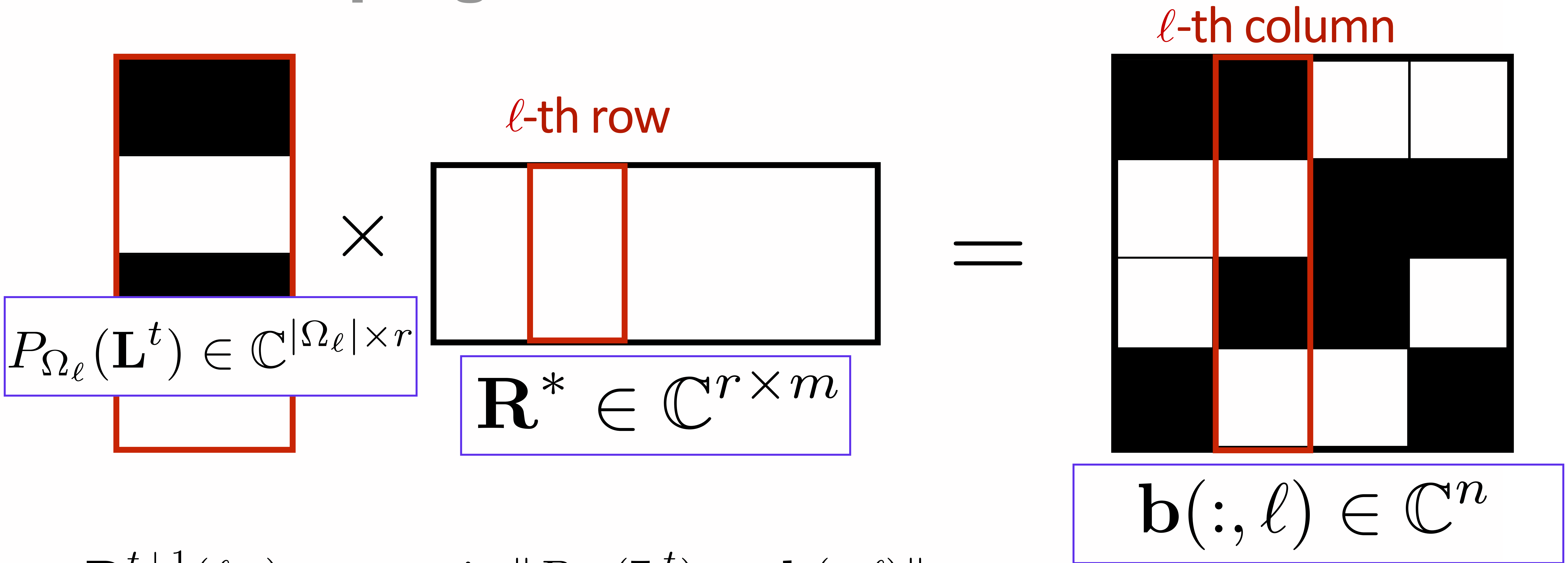
$$\mathbf{R}^{t+1} = \operatorname*{arg\,min}_{\mathbf{R}\in\mathbb{C}^{m\times r}} \|P_{\Omega}(\mathbf{L}^t\mathbf{R}^*) - \mathbf{b}\|_F$$

can be decoupled to solve each row of $\mathbf{R}^{t+1}$ independently.

Decoupling method: Visualization



Decoupling method: Visualization



$$\mathbf{R}^{t+1}(\ell, :) = \underset{v \in \mathbb{C}^r}{\arg\min} \|P_{\Omega_\ell}(\mathbf{L}^t)v - \mathbf{b}(:, \ell)\|_2$$

SLIM

Methodology: Decoupling method

So each row can be solved independently as

$$\mathbf{R}^{t+1}(\ell,:) = \underset{v\in\mathbb{C}^r}{\arg\min}\, \|P_{\Omega_\ell}(\mathbf{L}^t)v - \mathbf{b}(:,\ell)\|_2$$

$P_{\Omega_\ell}(\mathbf{L}^t)$ is $\mathbf{L}^t$ restricted to the entries observed in the ℓ-th column

Methodology: Decoupling method

$$\mathbf{R}^{t+1}(\ell,:) = \underset{v\in\mathbb{C}^r}{\arg\min}\, \|P_{\Omega_\ell}(\mathbf{L}^t)v - \mathbf{b}(:,\ell)\|_2$$

closed form solution is

$$\mathbf{R}^{t+1}(\ell,:) = \left(P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t)\right)^{-1} P_{\Omega_\ell}(\mathbf{L}^t)^* \mathbf{b}(:,\ell)$$

Methodology: Decoupling method

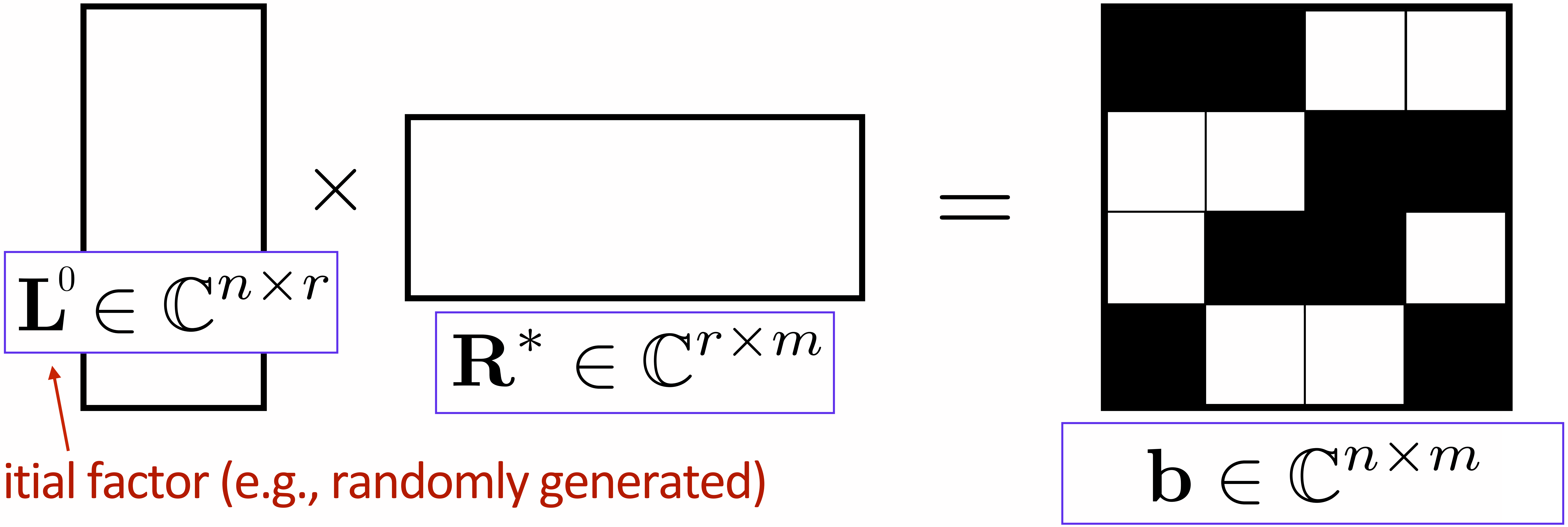
$$\mathbf{R}^{t+1}(\ell,:) = \left(P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t)\right)^{-1} P_{\Omega_\ell}(\mathbf{L}^t)^* \mathbf{b}(:,\ell)$$

cheap if $r \ll \min\{n,m\}$

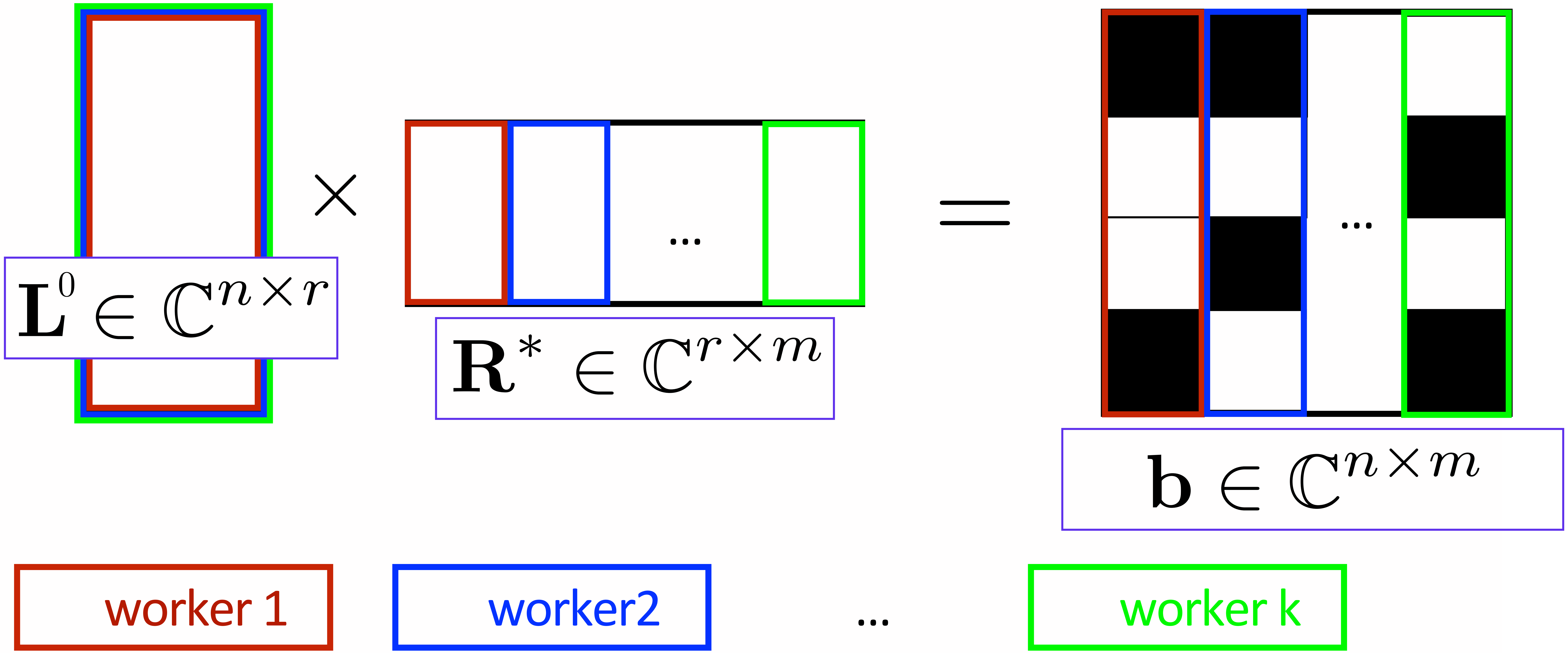
since $P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t) \in \mathbb{C}^{r\times r}$

(e.g., via Cholesky factorization)

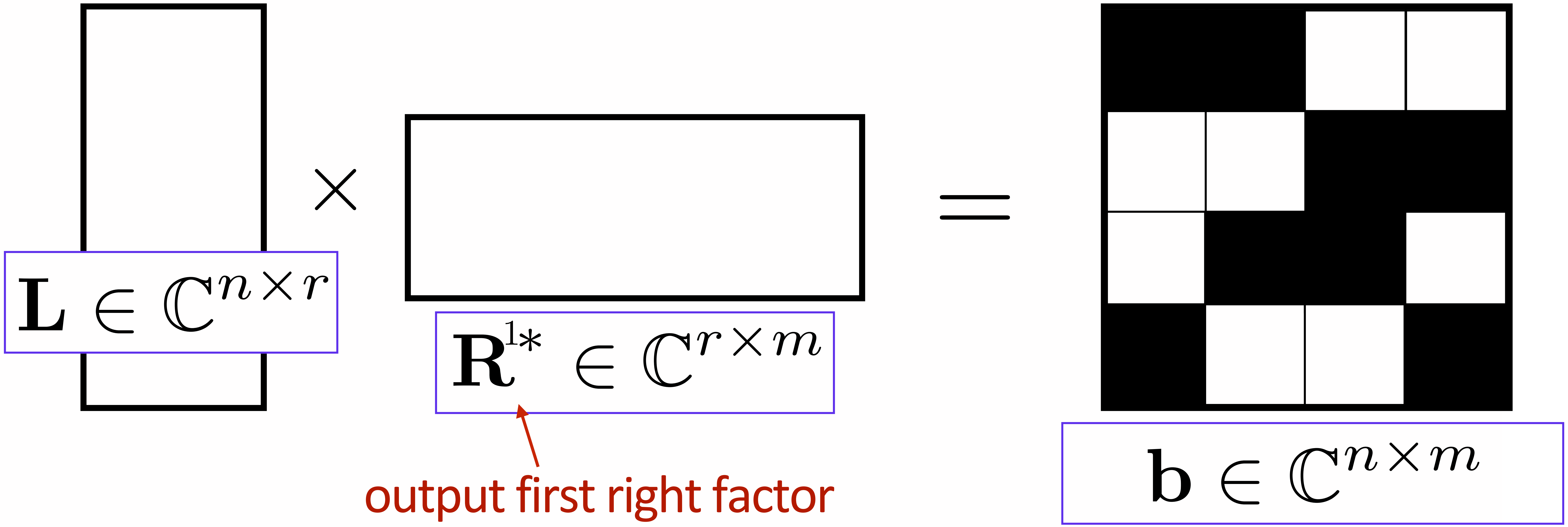
Decoupling method in action



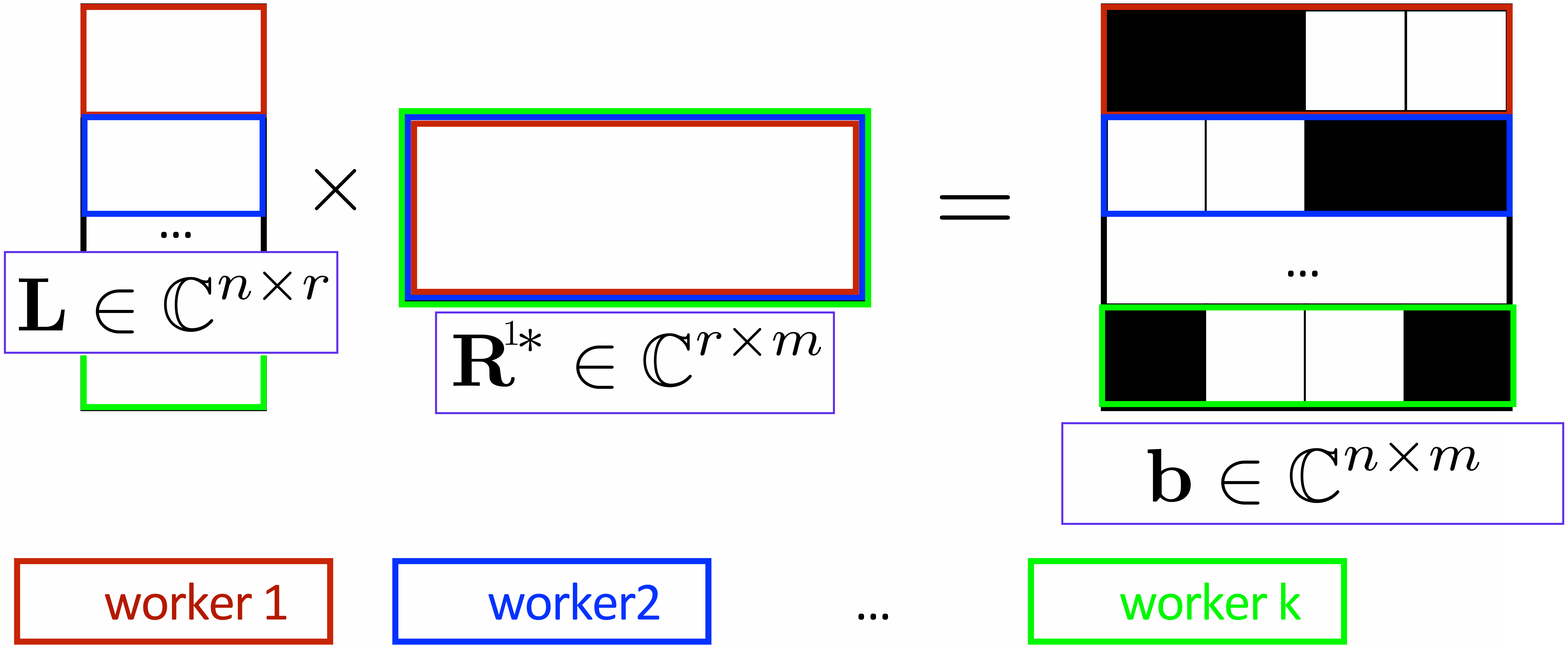
Decoupling method in action



Decoupling method in action

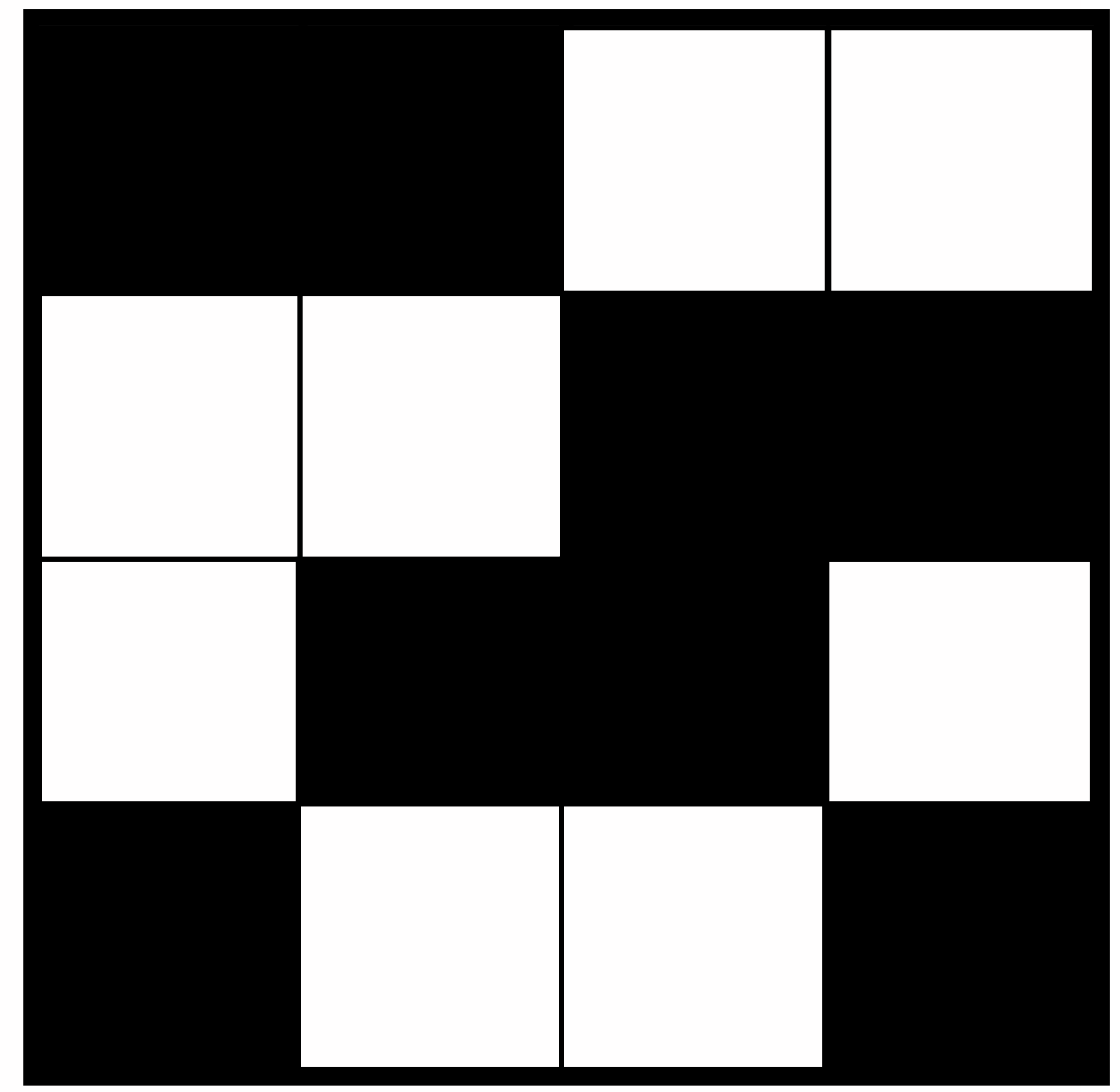


Decoupling method in action



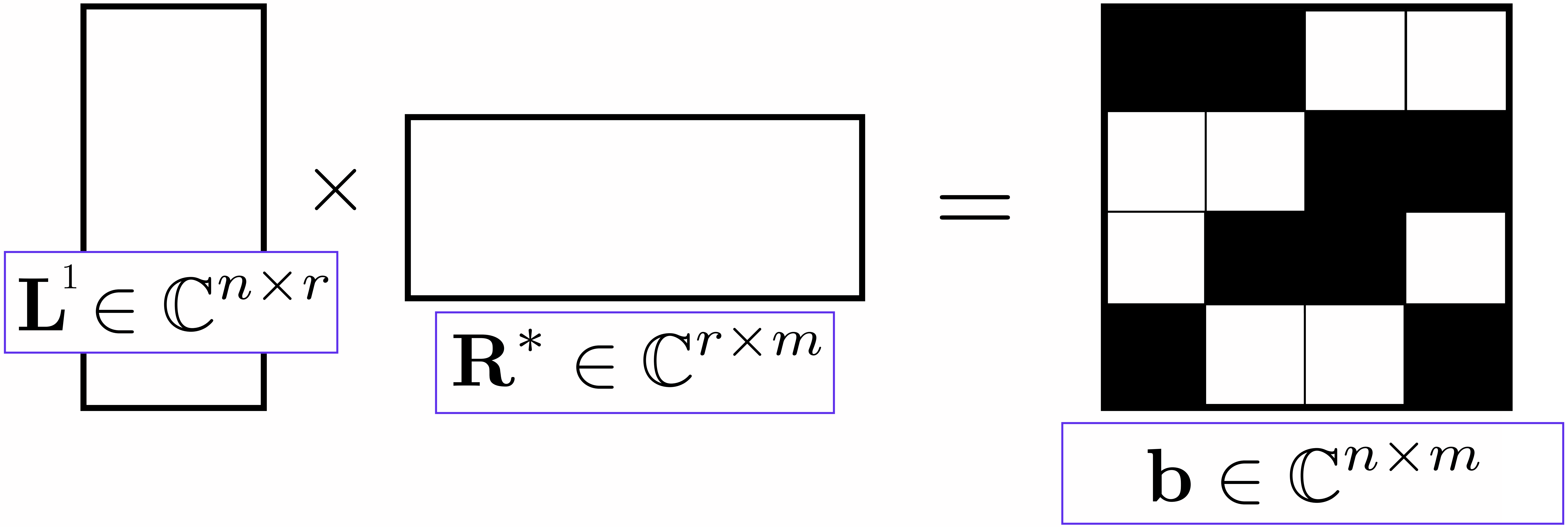
Decoupling method in action

$$\mathbf{L}^1 \in \mathbb{C}^{n\times r} \quad \times \quad \mathbf{R}^* \in \mathbb{C}^{r\times m} \quad = \quad \mathbf{b} \in \mathbb{C}^{n\times m}$$



output first left factor

Decoupling method in action



...and so on solve for $(\mathbf{L}^2, \mathbf{R}^2),\ (\mathbf{L}^3, \mathbf{R}^3),\ \ldots\ ,\ (\mathbf{L}^T, \mathbf{R}^T)$

SLIM

Outline

- Matrix completion
  - alternating least squares
  - decoupling method
- Parallel implementation in Julia
- Numerical experiments

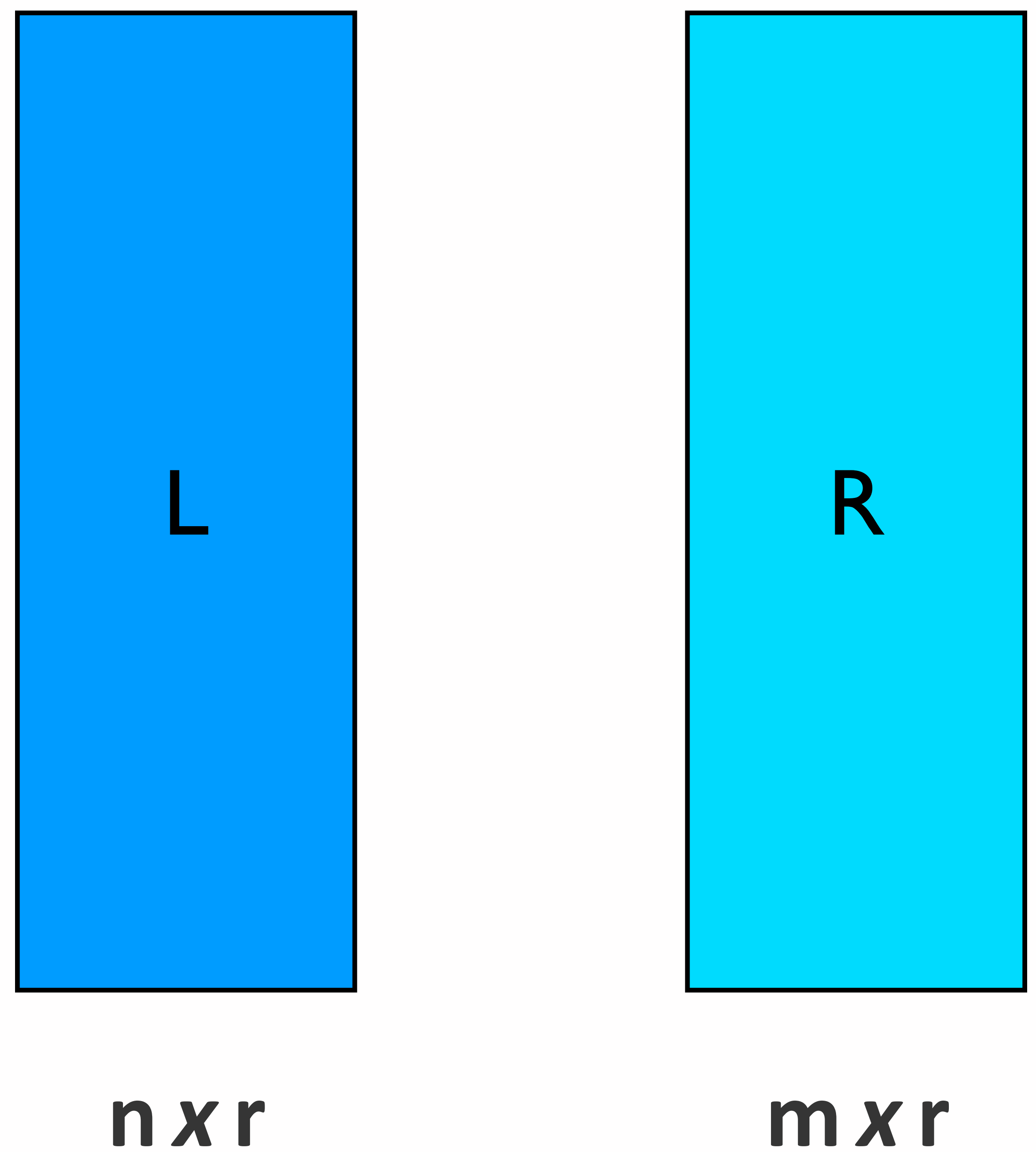
DCLR implementation
(Dedicated Communicators for L & R factors)

Dedicate a worker to store each **L** and **R** factor, and handle all messaging related to factor updates.

Remaining workers will store distributed data, **b**, and solve **L** & **R** on local portion of **b**.

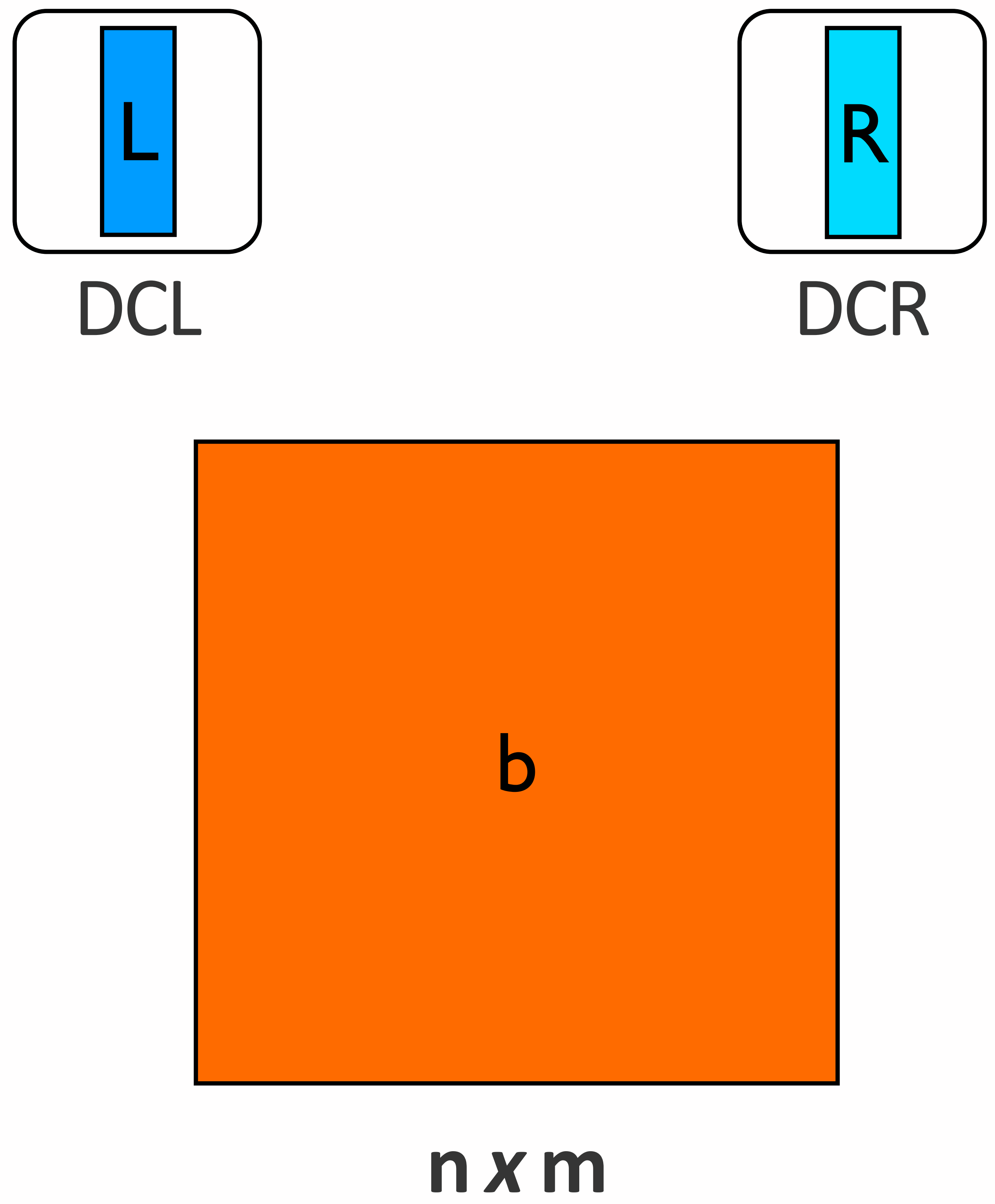
DCLR implementation
(Dedicated Communicators for L & R factors)

1) Dedicate a worker to store each **L** & **R** factor, and handle all messaging related to factor updates.



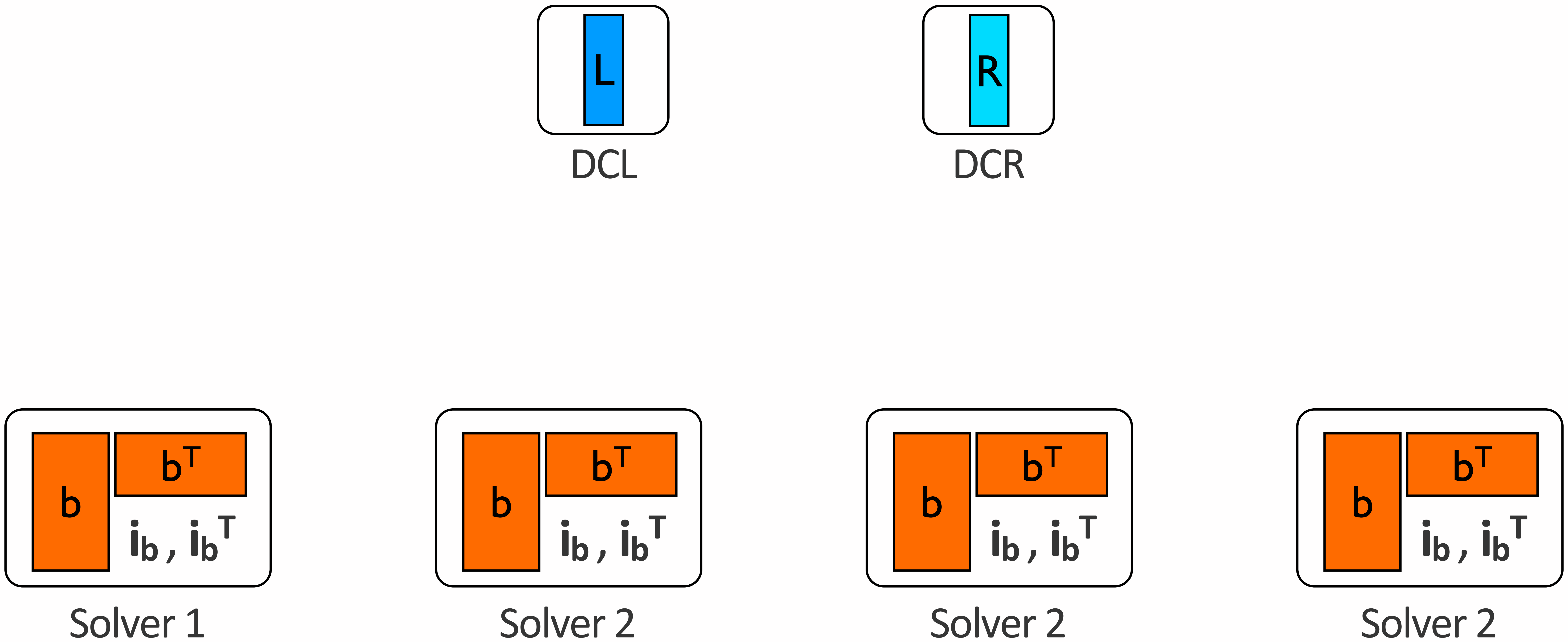
DCLR implementation
(Dedicated Communicators for L & R factors)

2) Remaining workers will store distributed data, **b**, and solve **L** & **R** on only local portion of **b**.



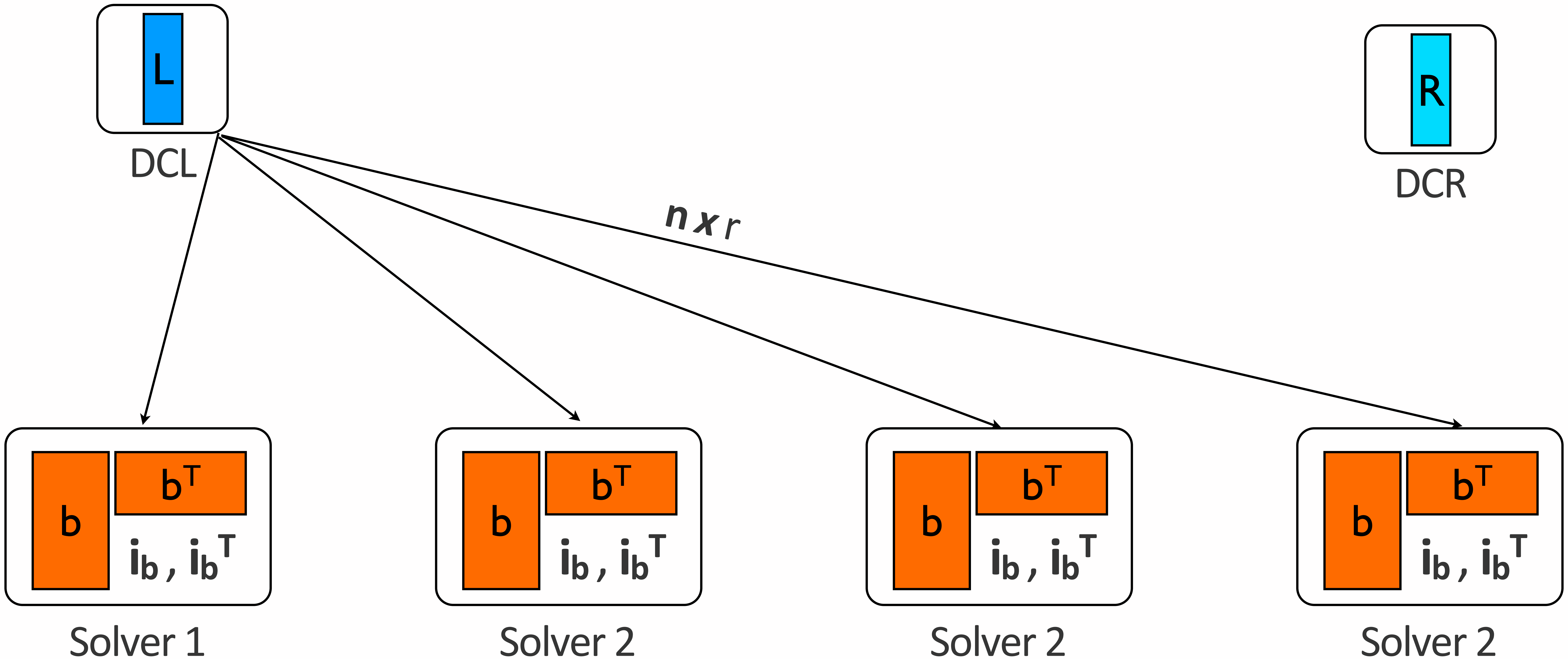
DCLR implementation
(Dedicated Communicators for L & R factors)

3) Distribute non-zero elements of each column of **b** & $\mathbf{b}^T$, and corresponding indexes for non-zero values in each column, $\mathbf{i_b}$ & $\mathbf{i_b}^T$



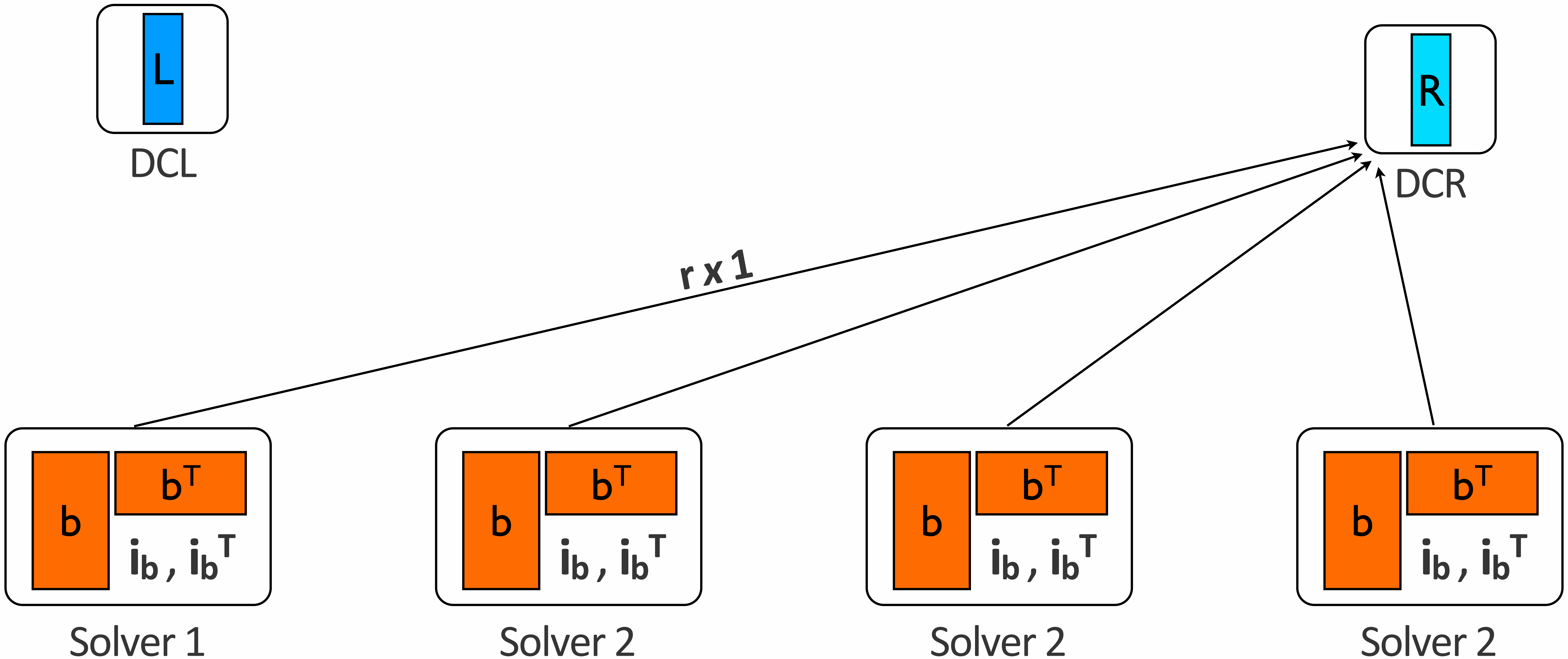
DCLR implementation
(Dedicated Communicators for L & R factors)

4) Solvers get latest **L** update from **DCL**



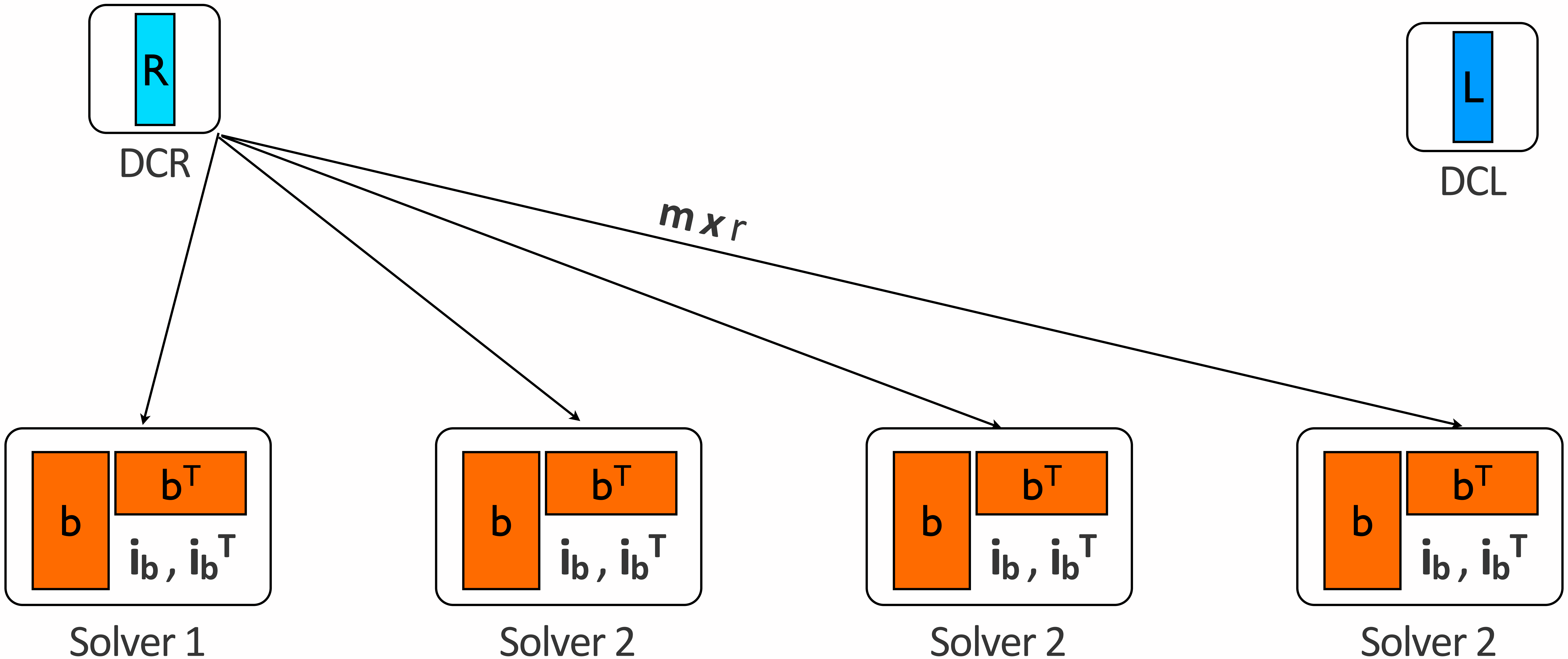
DCLR implementation
(Dedicated Communicators for L & R factors)

4) Solvers compute a row update of **R,** using **L** and first column of local part of **b**



DCLR implementation
(Dedicated Communicators for L and R factors)

4) Vice versa for **R** update



Outline

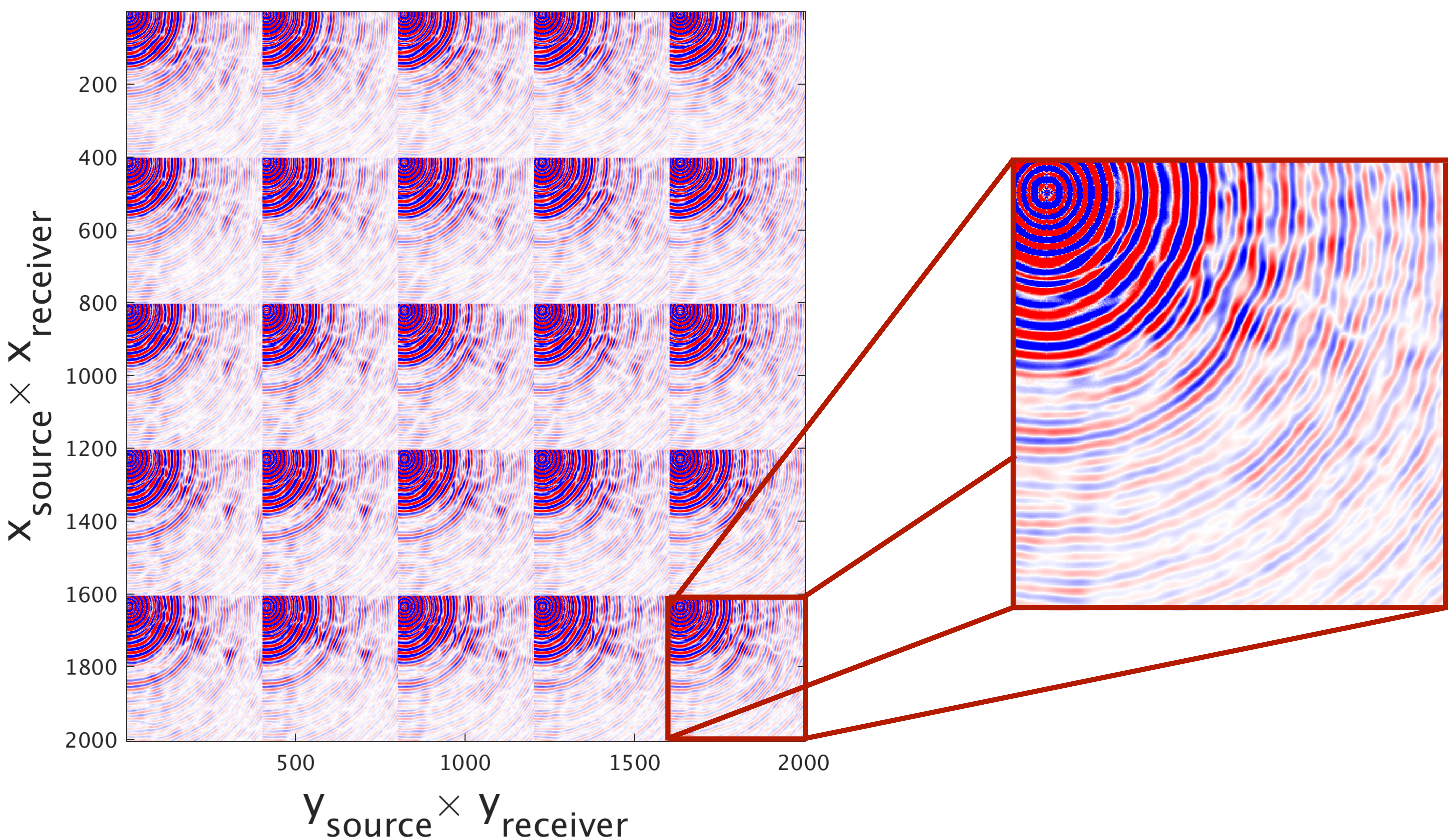
- Matrix completion
  - alternating least squares
  - decoupling method
- Parallel implementation in Julia
- Numerical Experiments

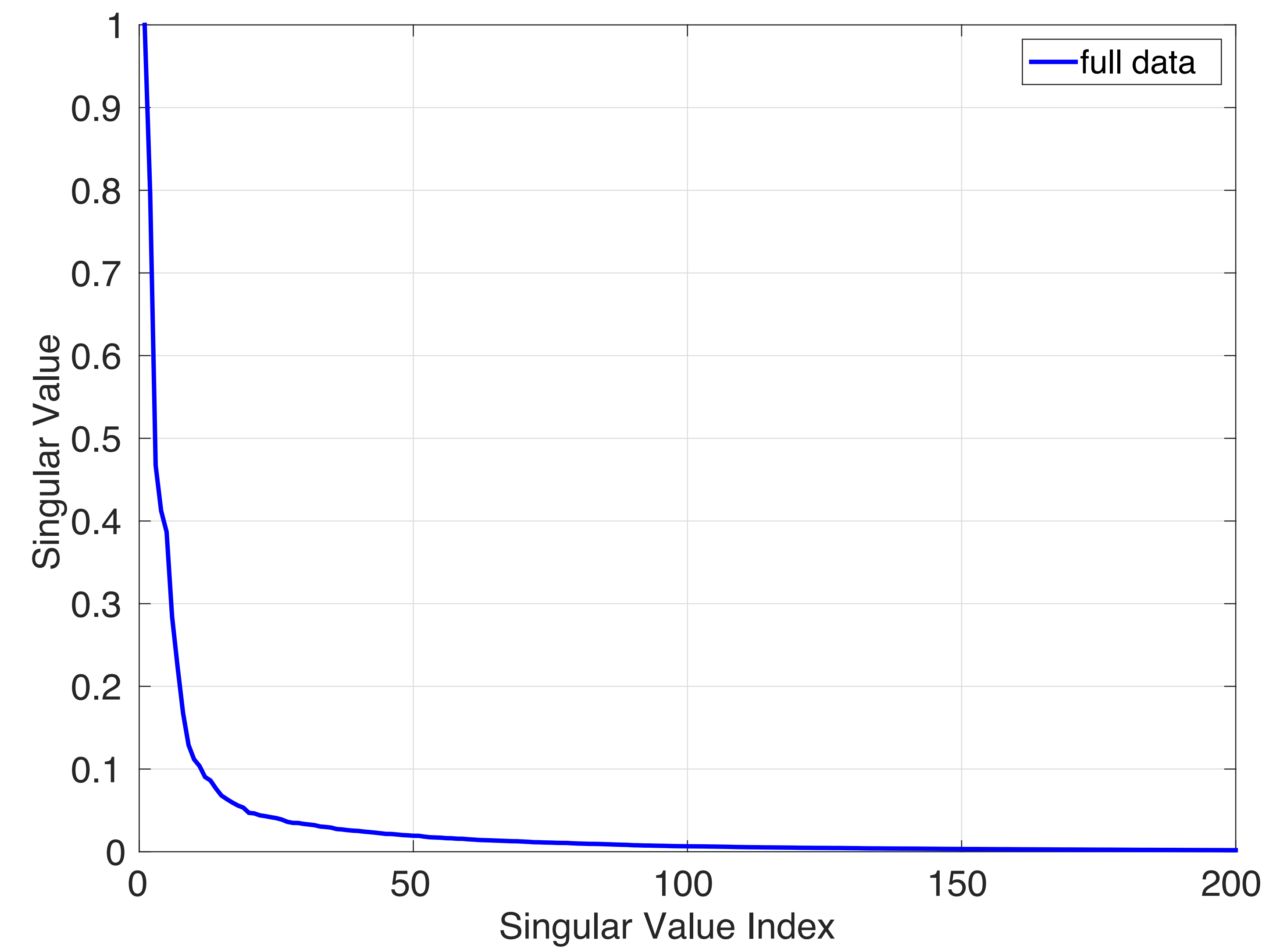
Interpolation: Synthetic BG 3D Model

- 68 x 68 sources with 401 x 401 receivers
- Data at 7.34 Hz and 12.3 Hz.
- Matricize in “(rec,rec)”-form

Data Matricized - (rec,rec) form

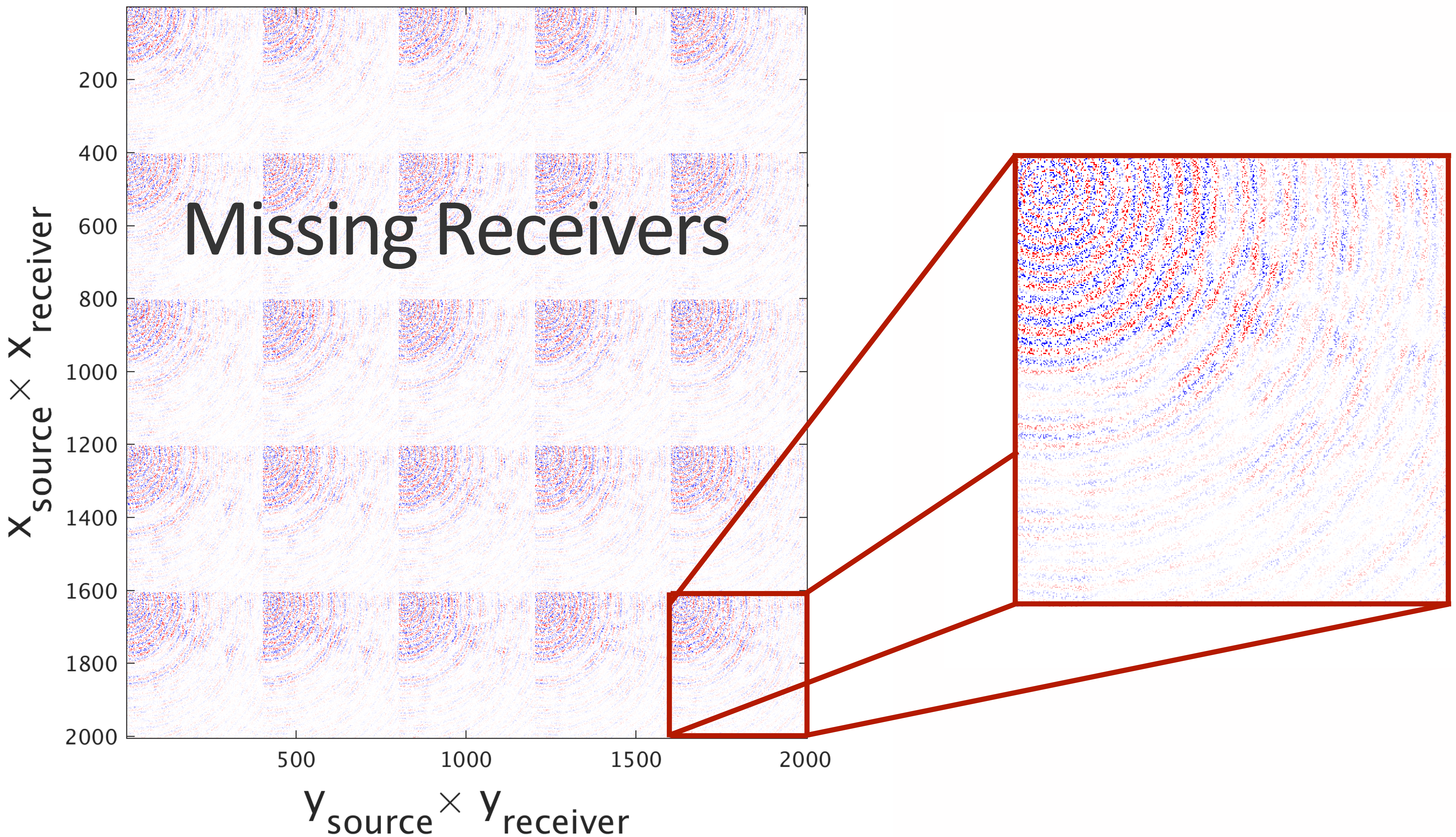
BG 3D Dataset 7.35 Hz

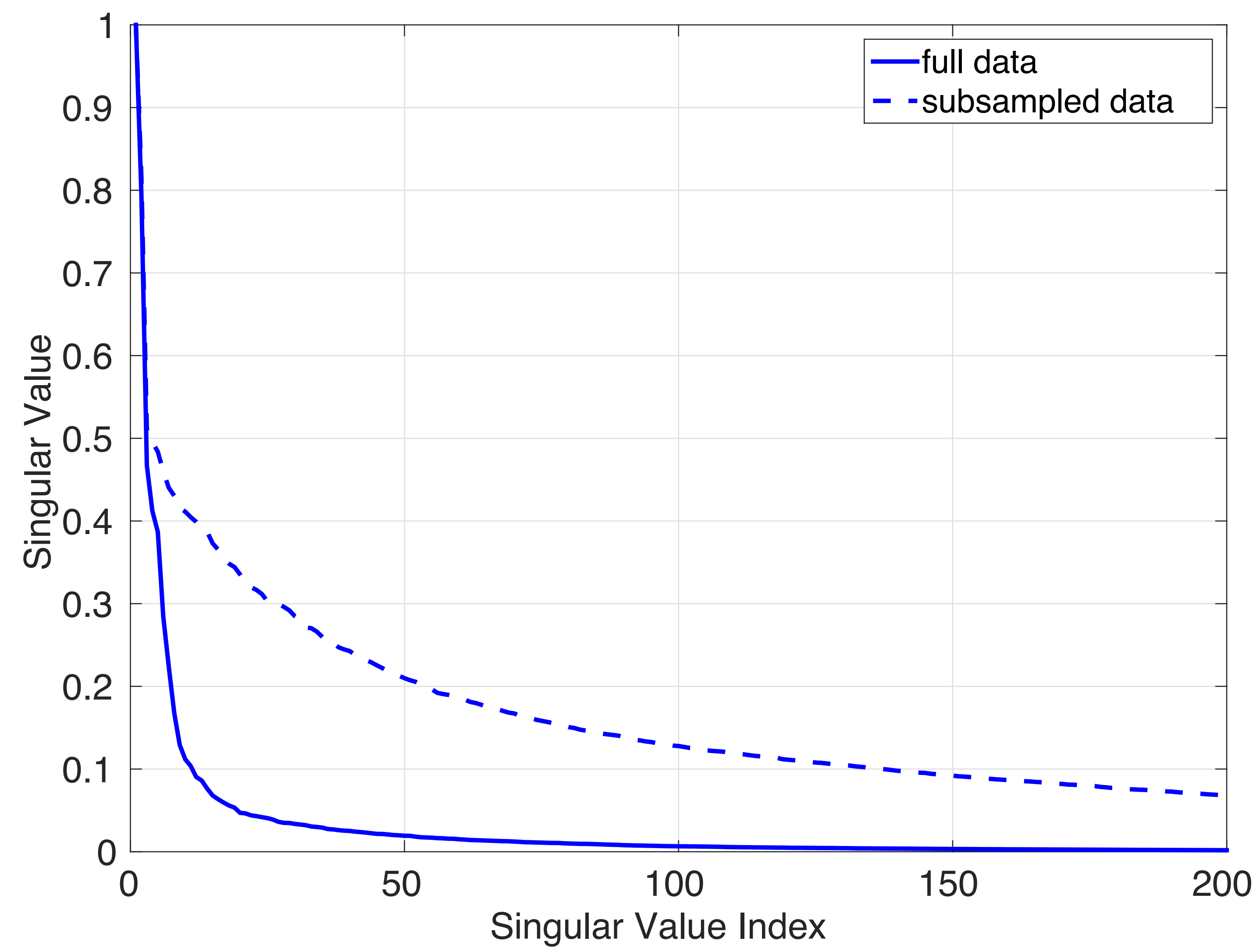




Data Matricized - (rec,rec) form

BG 3D Dataset 7.35 Hz

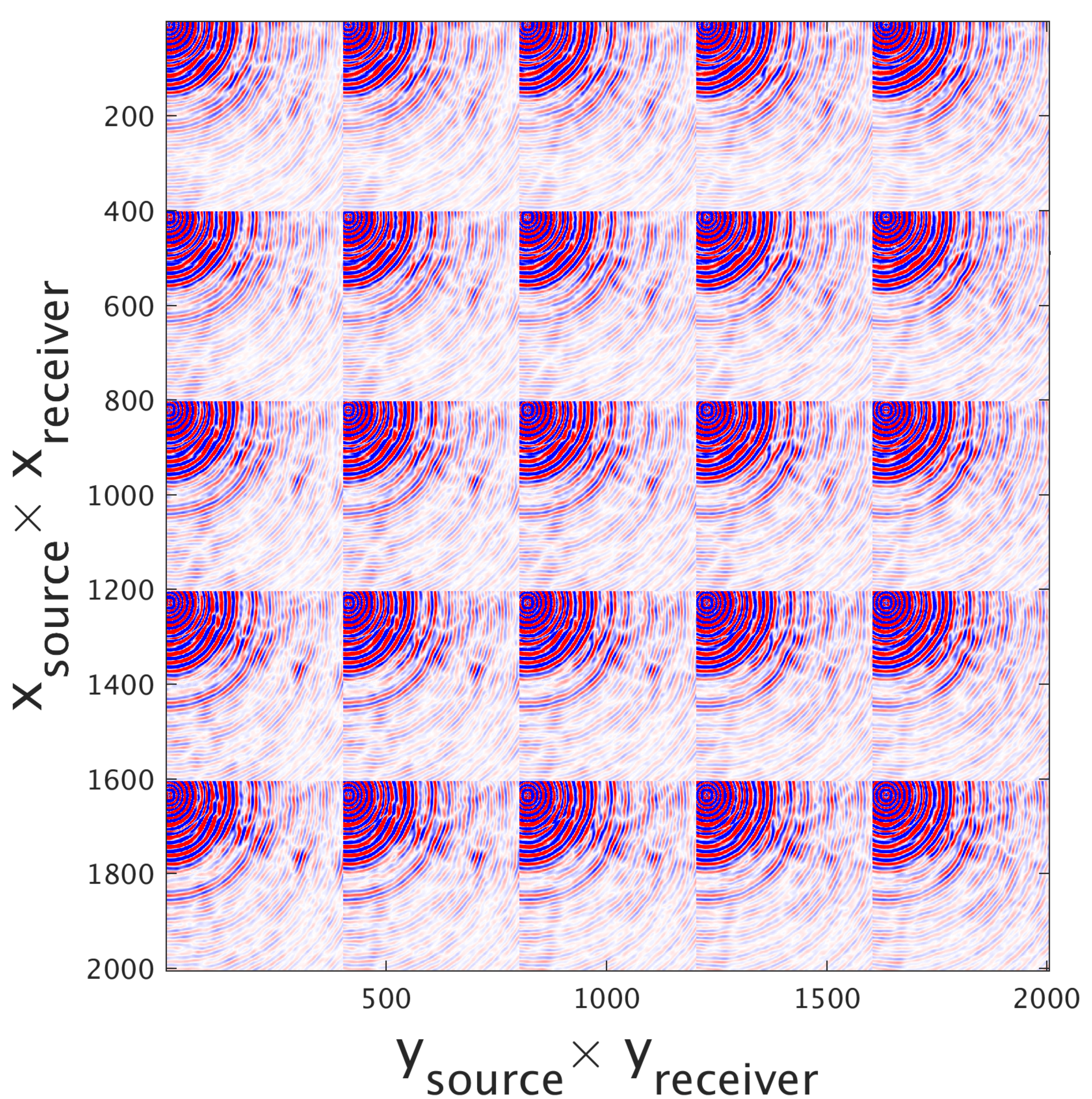




3D Interpolation Experiment

BG 3D Dataset

7.35 Hz



Size: 27,268 x 27,268

(full slice, no windowing)

Remove 80 % of Receivers randomly

Compare Interpolation via:

- SPG-LR

- Decoupling method

How to choose the rank parameter?

Issue: need

$$\mathbf{R}^{t+1}(\ell,:) = \left(P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t)\right)^{-1} P_{\Omega_\ell}(\mathbf{L}^t)^* \mathbf{b}(:,\ell)$$

How do we know $P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t) \in \mathbb{C}^{r\times r}$ is invertible?

How to choose the rank parameter?

Theorem:

Let Ω be chosen uniformly at random.

Let $\mathbf{L} \in \mathbb{C}^{n \times r}$ be full rank, define $\tilde{\mathbf{L}} = \mathrm{orth}(\mathbf{L})$ and $\beta := \max_{k,\ell} \left(\tilde{\mathbf{L}}_{k,\ell}\right)^2$.

Then if $|\Omega| \geq \alpha \frac{8}{3} \beta n r \log(nr)$

$P_{\Omega_\ell}(\mathbf{L}^t)^* P_{\Omega_\ell}(\mathbf{L}^t)$ is invertible for every $\ell \in \{1, 2, ..., n\}$
with probability $\geq 1 - 2n^{1-\alpha}$

SLIM

How to choose the rank parameter?

$$|\Omega| \geq \alpha \frac{8}{3} \beta n r \log(nr)$$

In our case: $|\Omega| = .2 \cdot nm$

$$n = m = 27,268$$

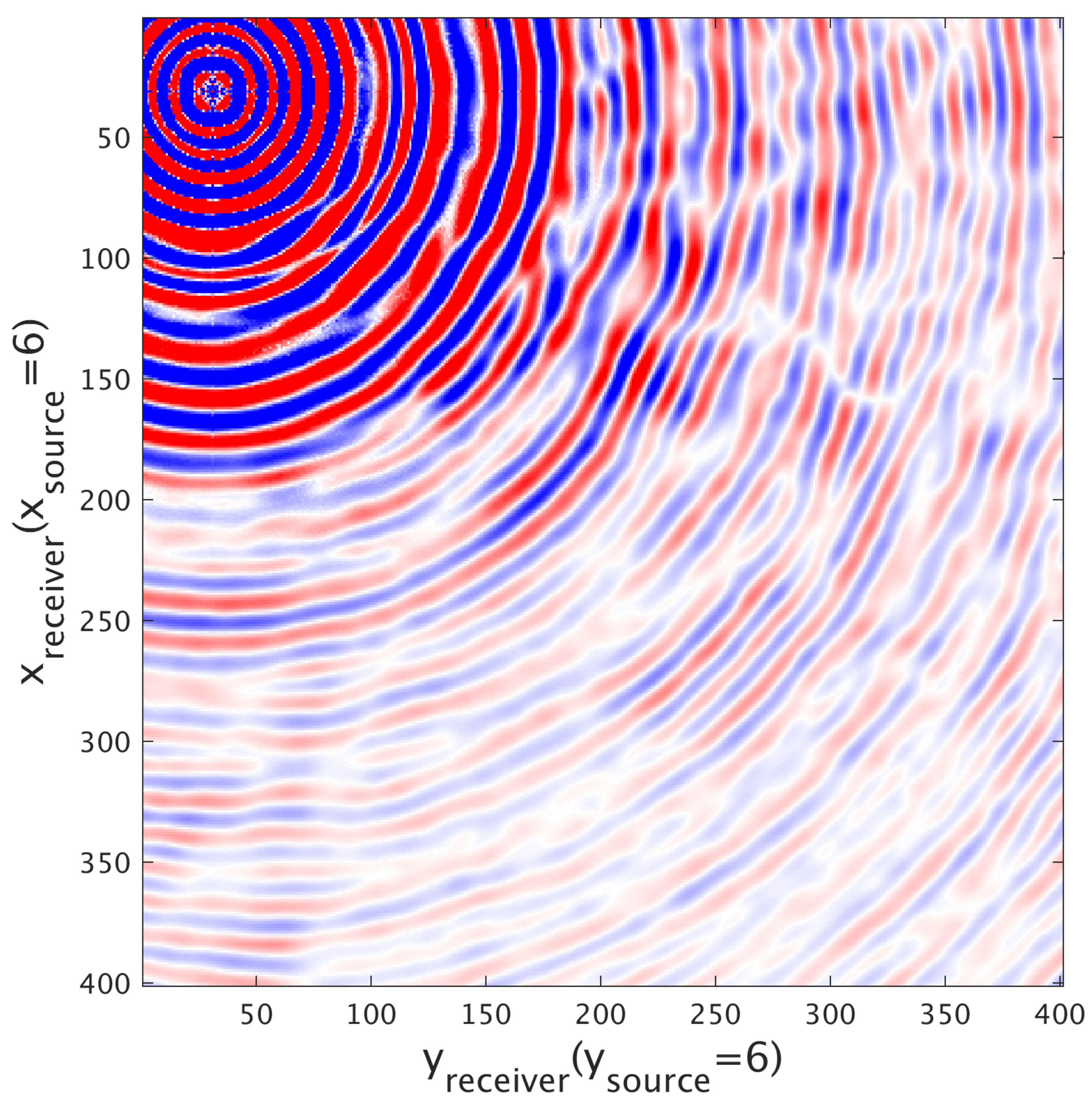
(ignoring constants) $\implies$ $r \leq 534$

choose upper bound as rank, gives well defined procedure.

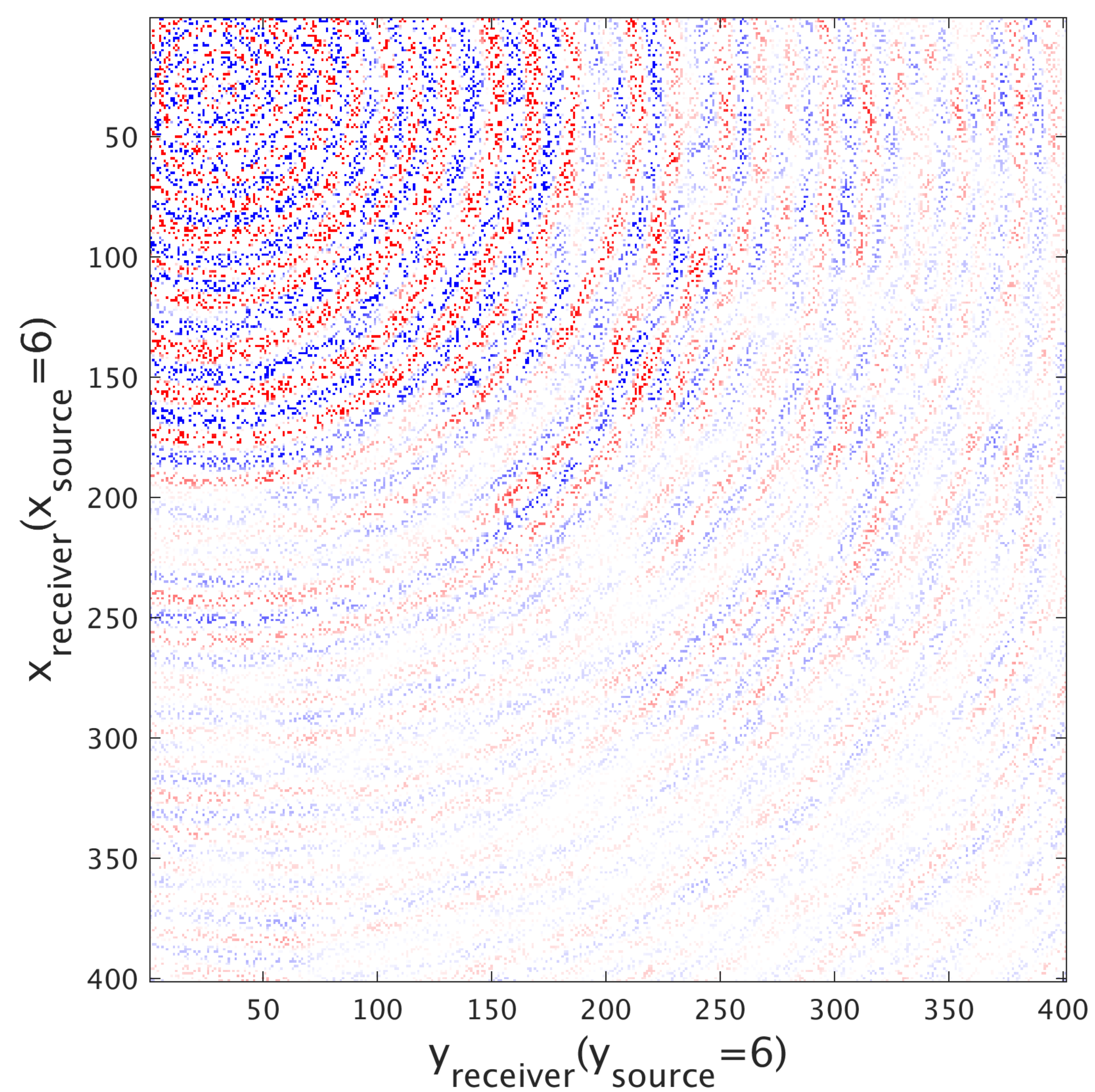
SLIM

Common Source Gather

True Source Gather



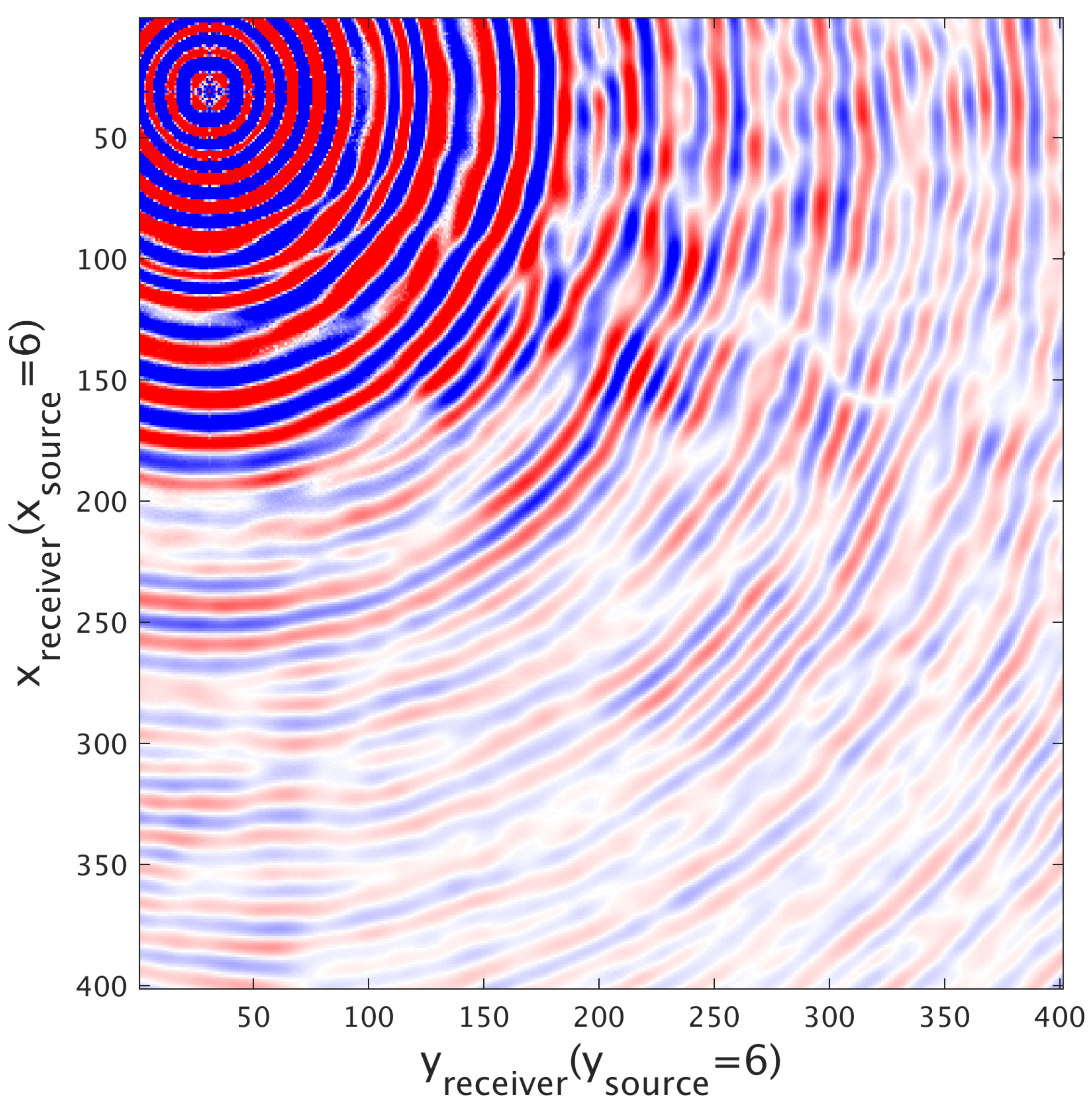
Subsampled Source Gather



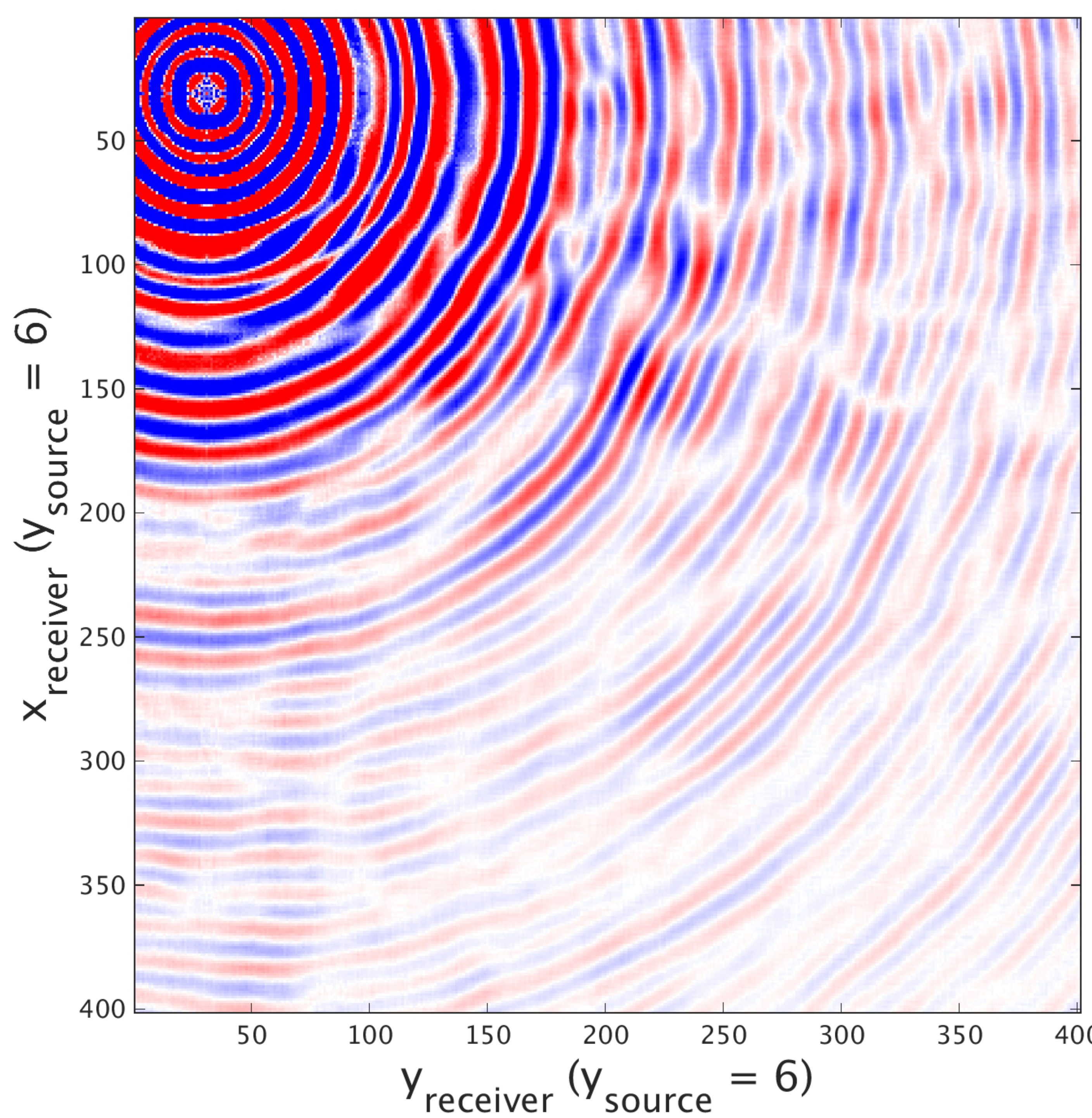
Remove 80 % of Receivers randomly

Results: SPG-LR

True Source Gather



Recovered Source Gather

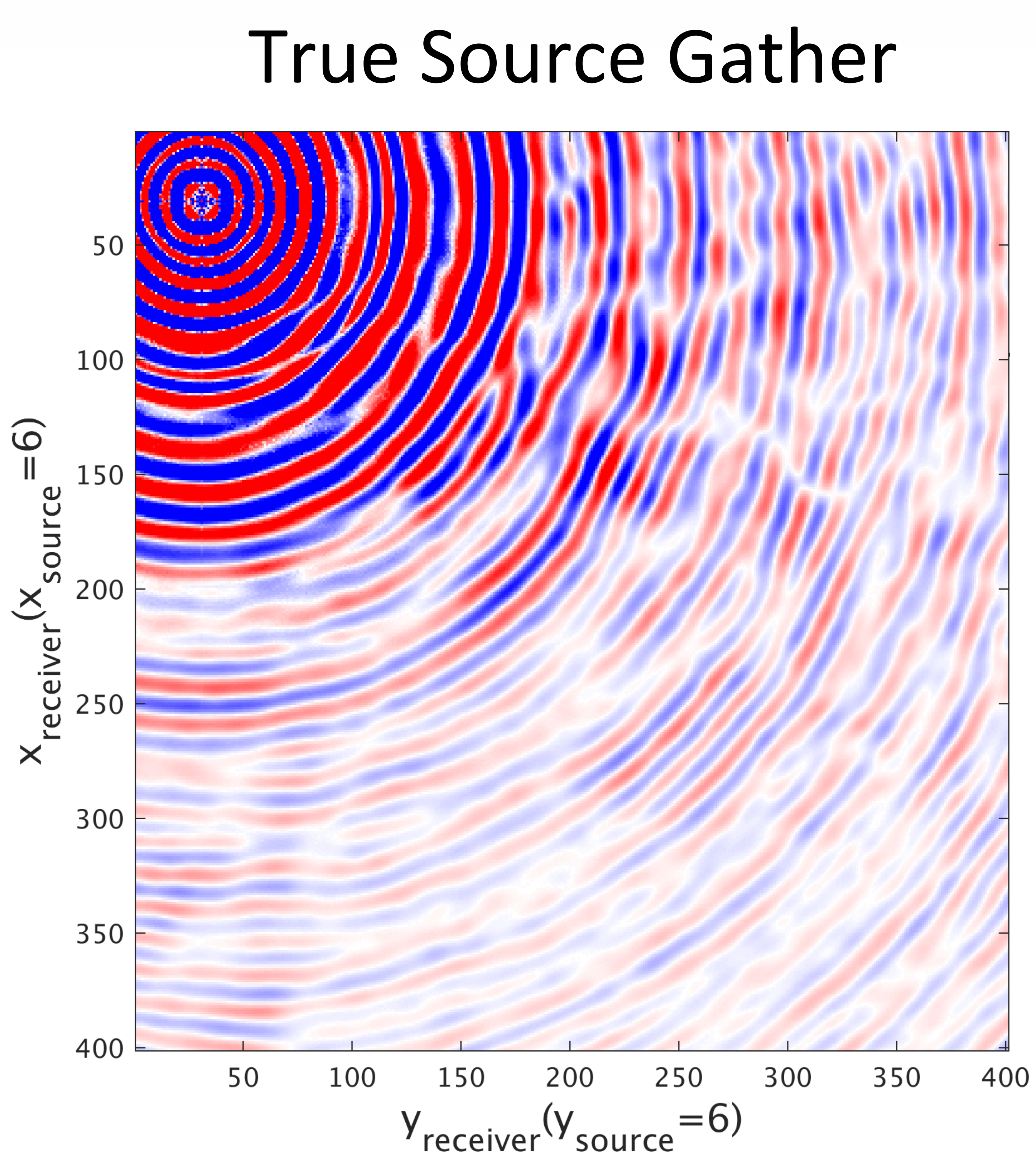


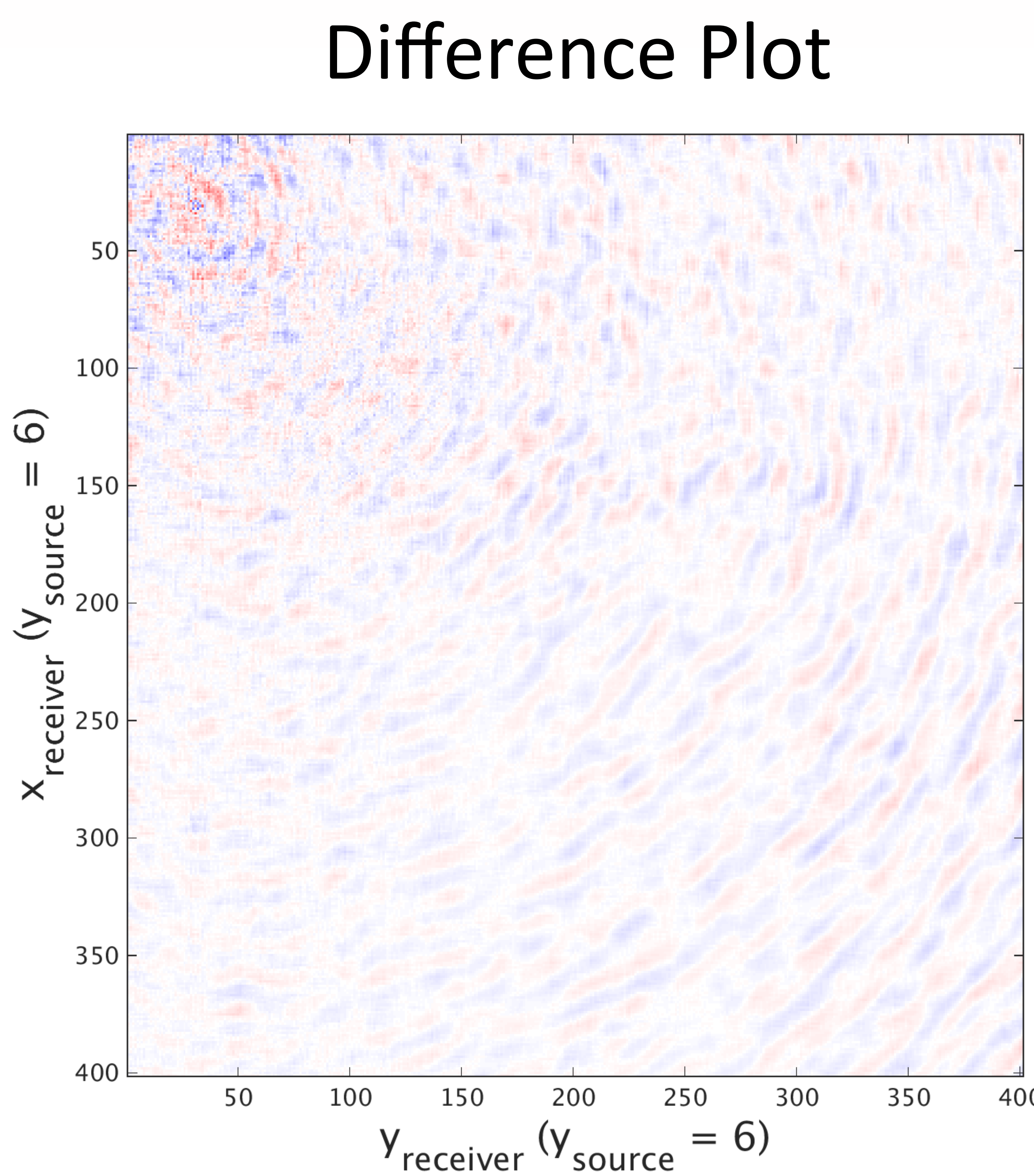
SPG-LR iterations: 400

SNR = 26.1 dB

Time = 82 hrs and 40 min

Results: SPG-LR





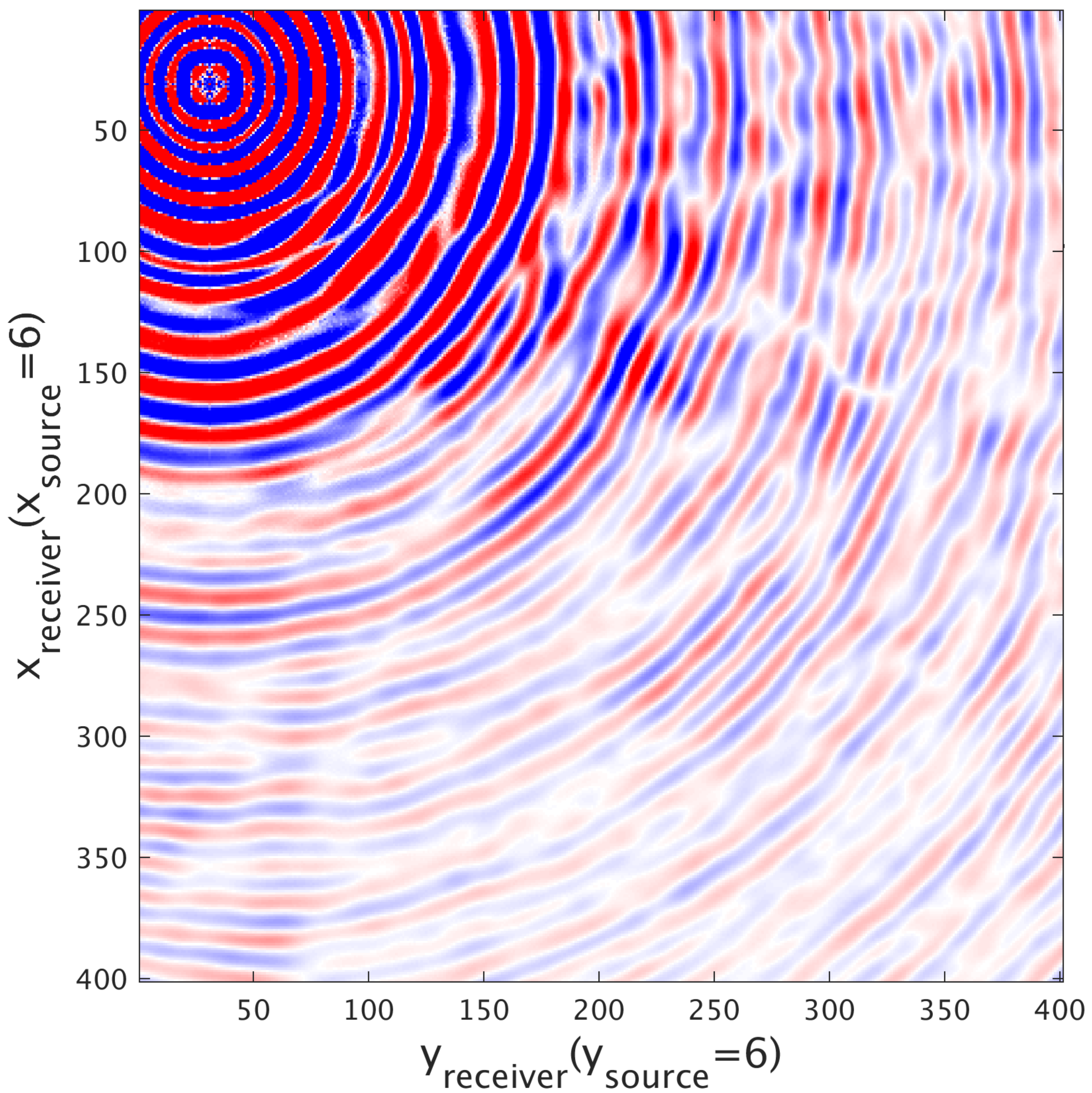
SPG-LR iterations: 400

SNR = 26.1 dB

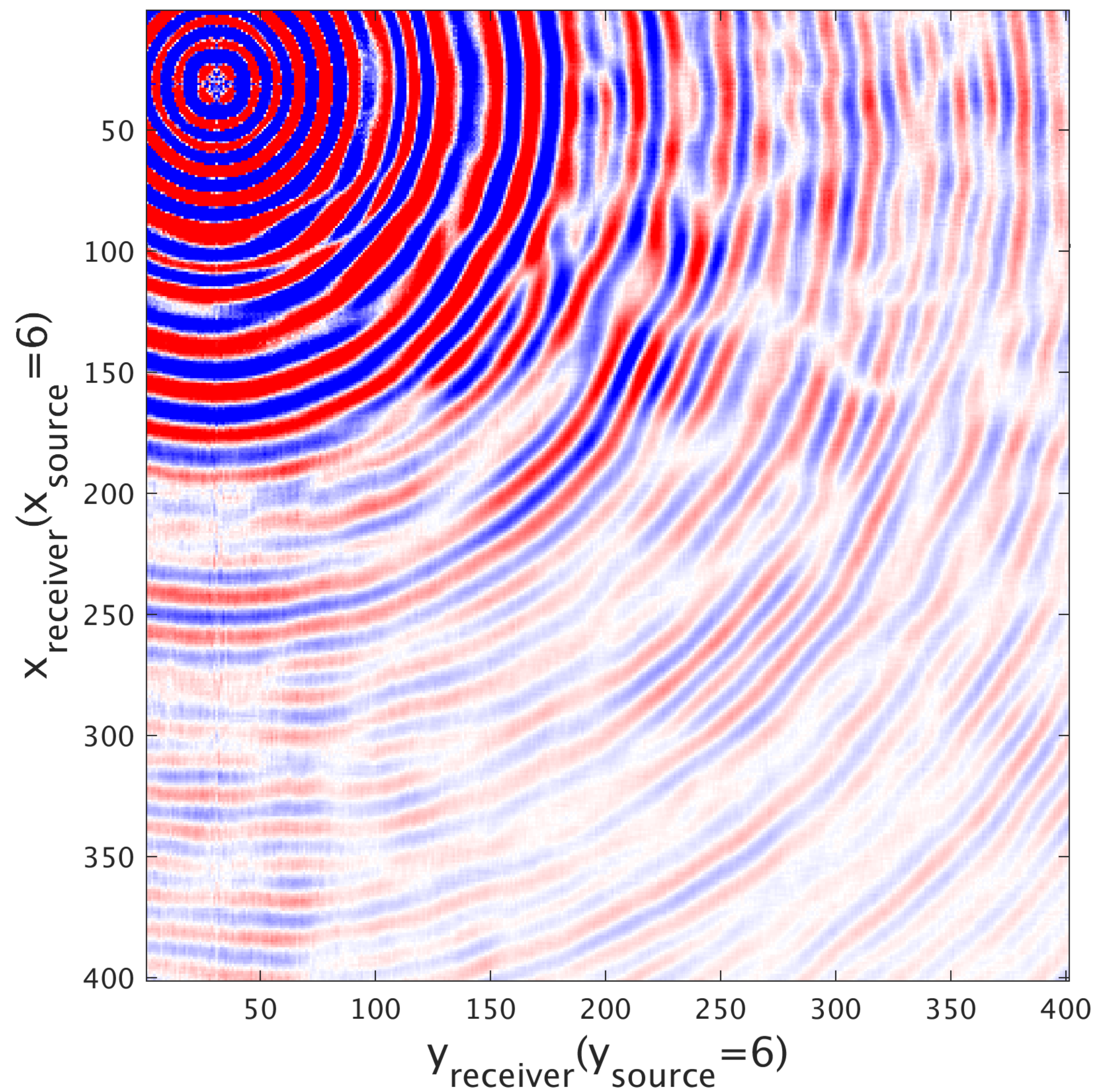
Time = 82 hrs and 40 min

Results: Decoupling method

True Source Gather



Recovered Source Gather



60 workers

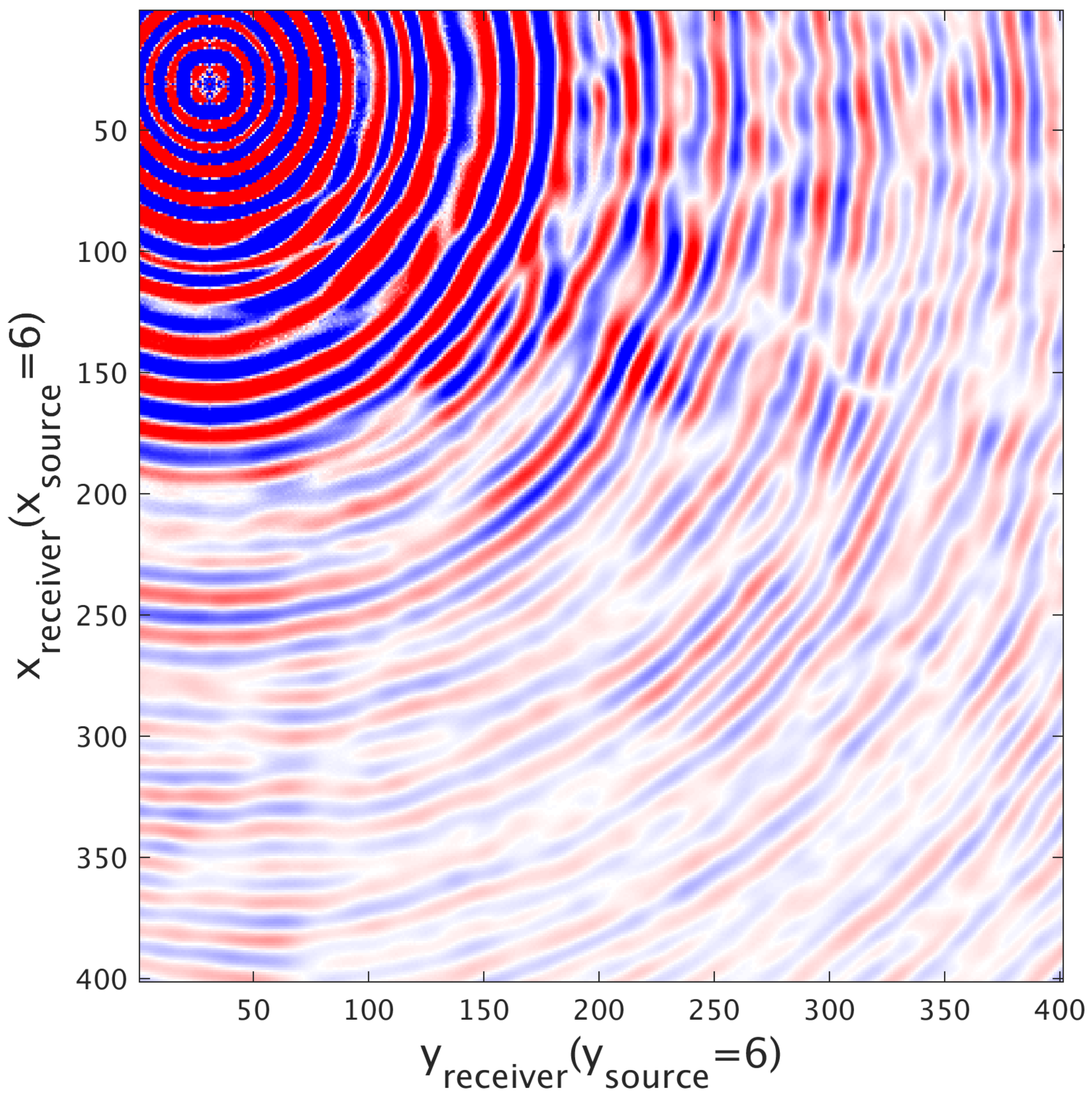
Alternations: 5

SNR = 24.2 dB

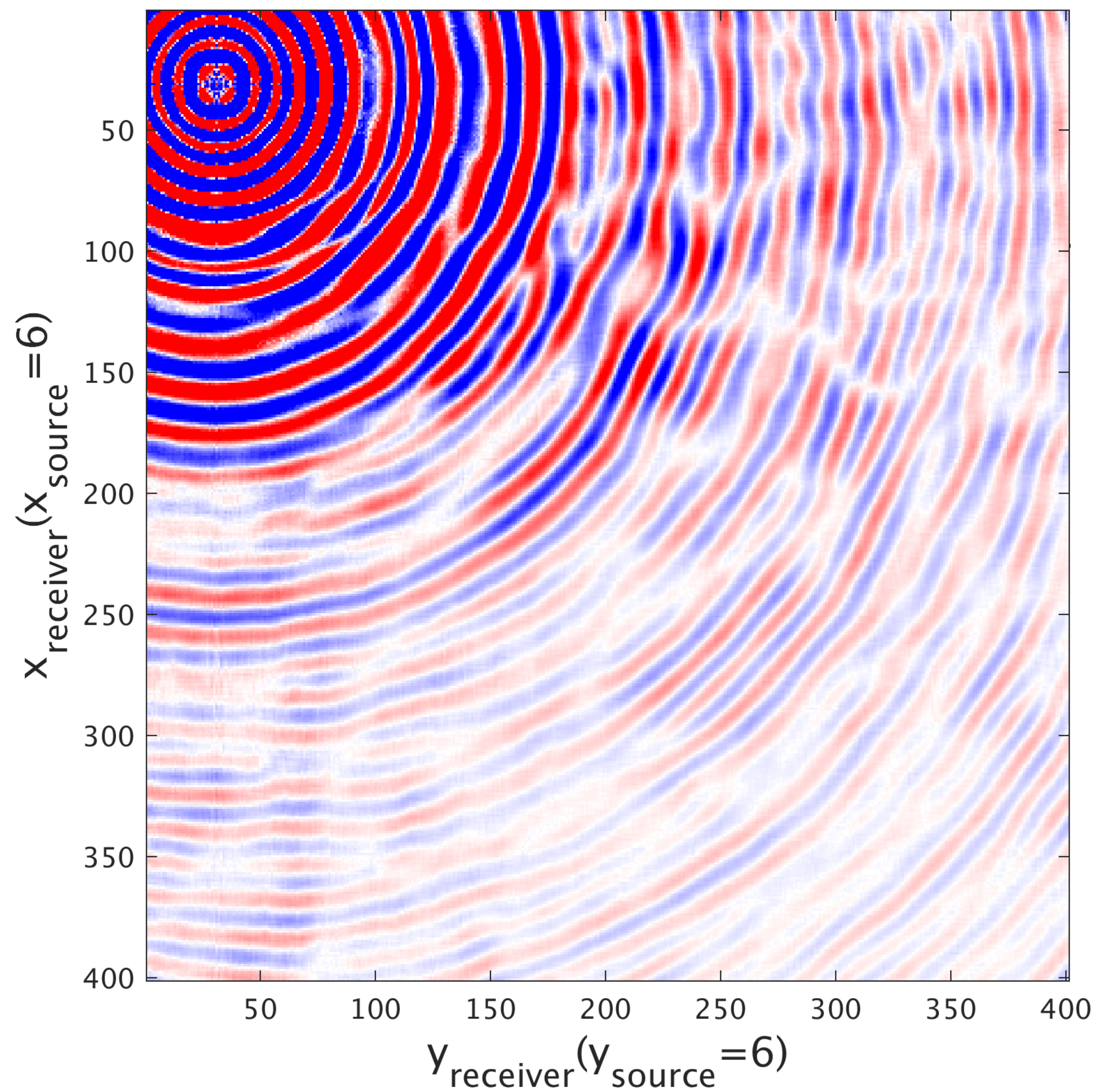
Time = 1 hr and 7 mins

Results: Decoupling method

True Source Gather



Recovered Source Gather



60 workers

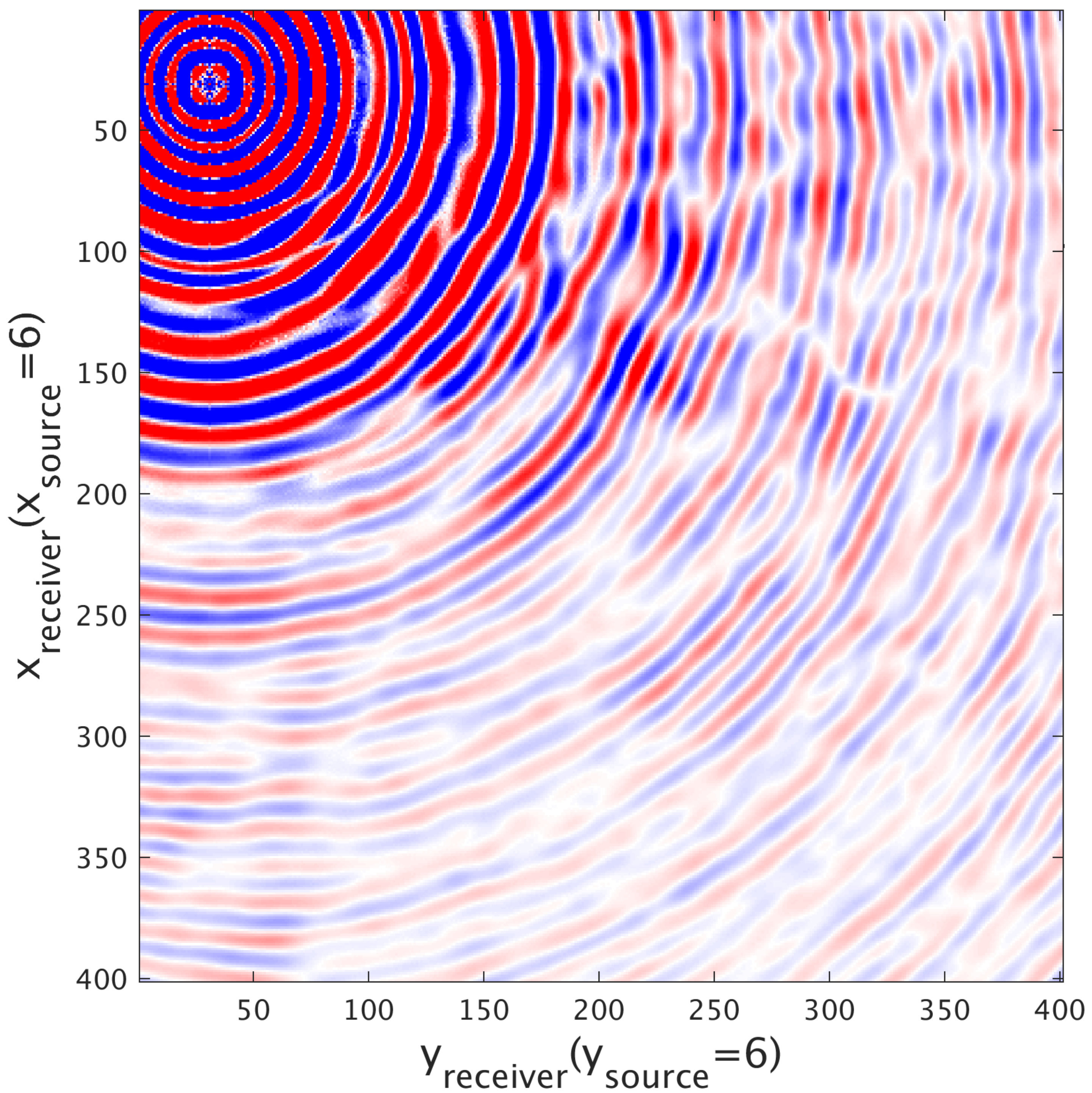
Alternations: 7

SNR = 25.1 dB

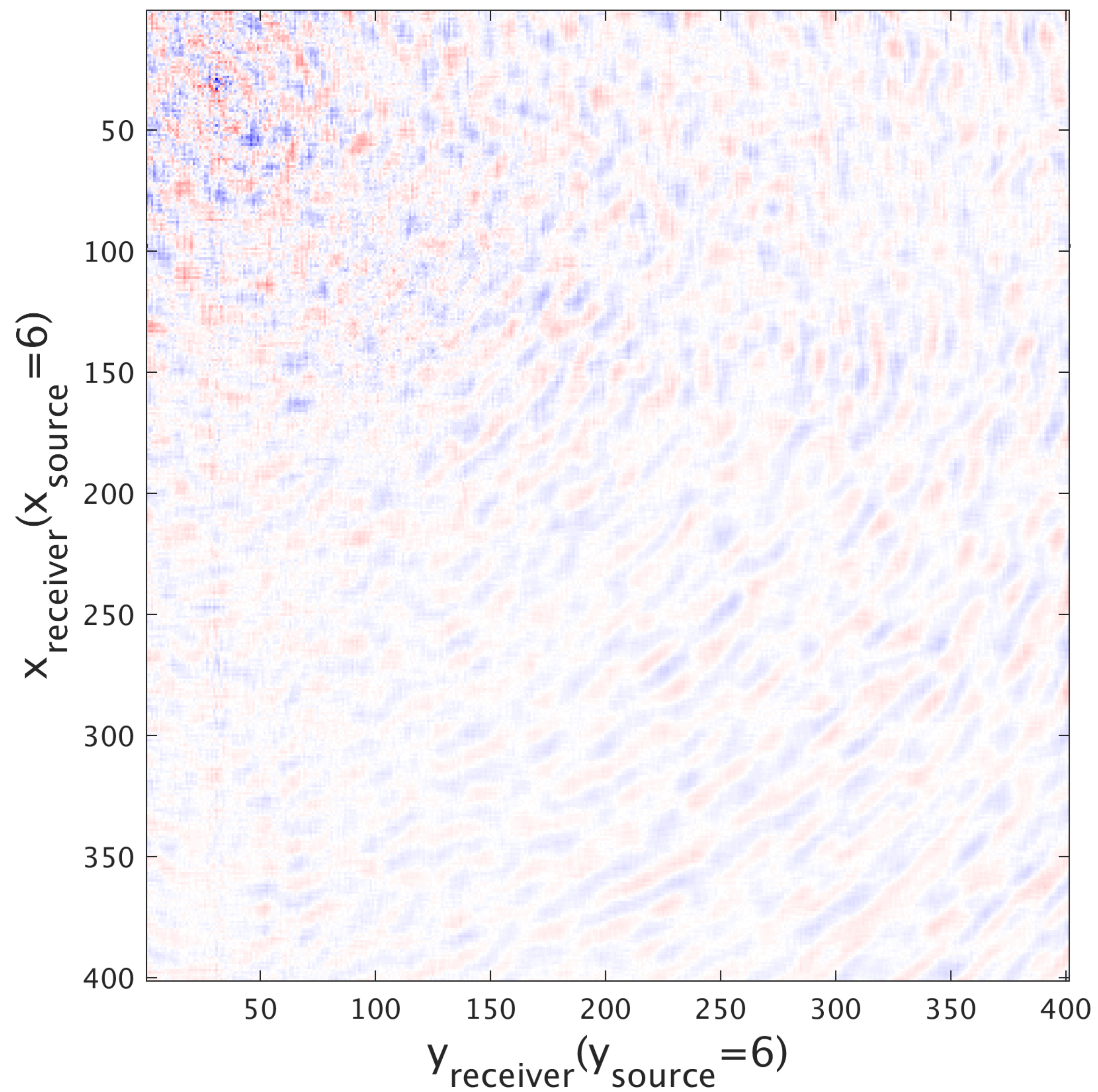
Time = 1 hr and 33 mins

Results: Decoupling method

True Source Gather



Difference Plot



60 workers

Alternations: 7

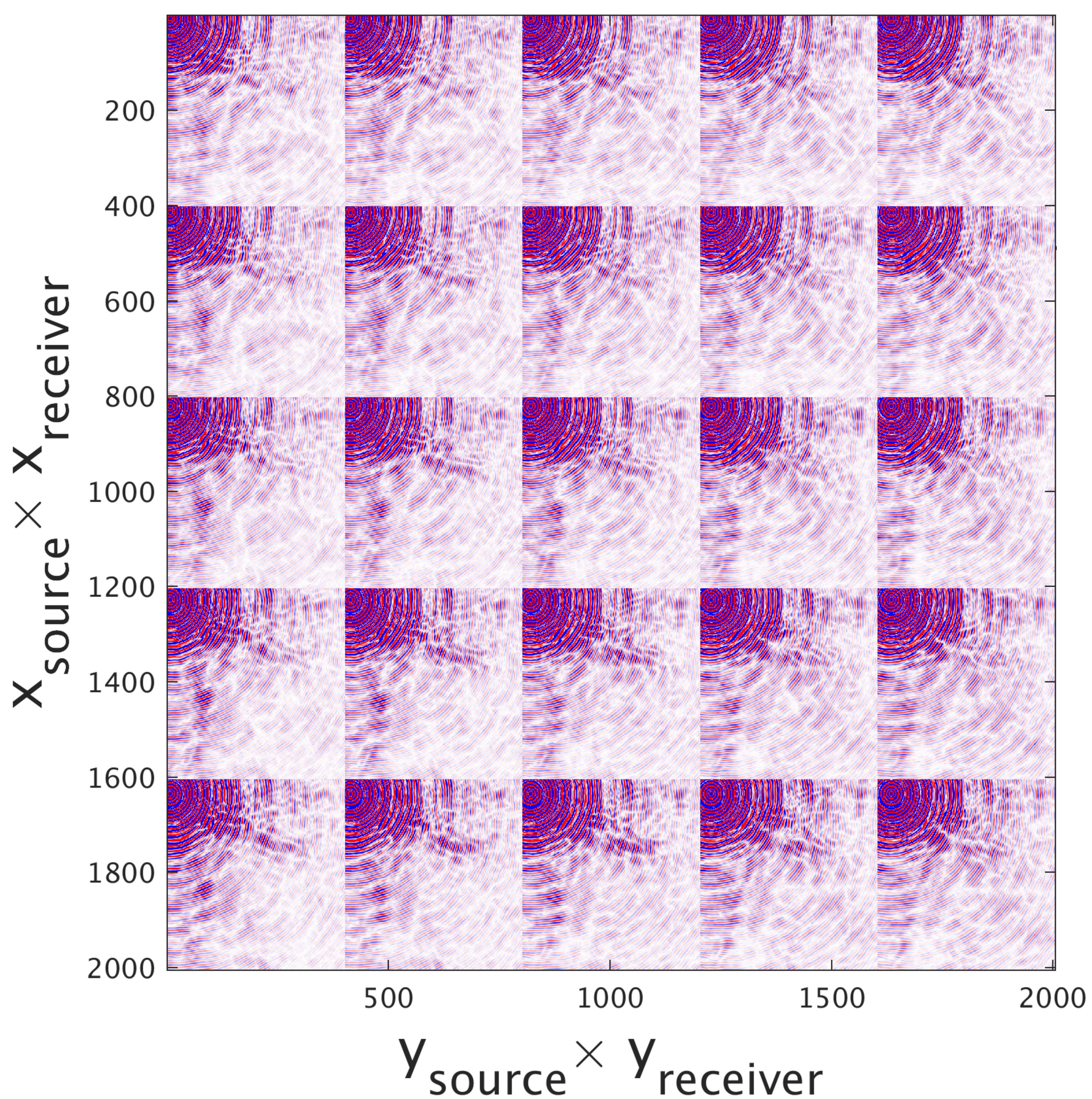
SNR = 25.1 dB

Time = 1 hr and 33 mins

3D Interpolation Experiment

BG 3D Dataset

12.3 Hz



Size: 27,268 x 27,268

(full slice, no windowing)

Remove 80 % of Receivers randomly

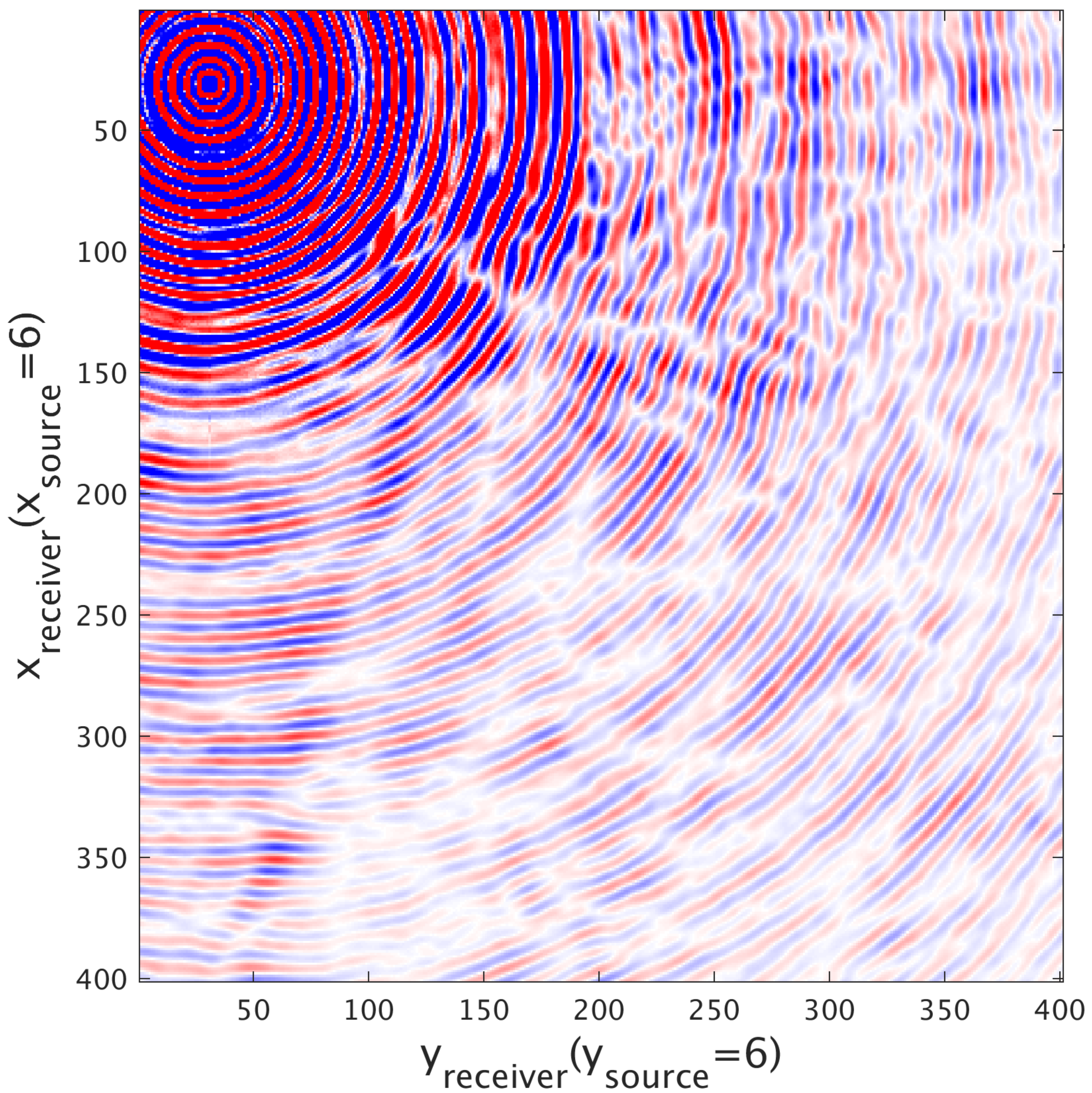
Compare Interpolation via:

- SPG-LR

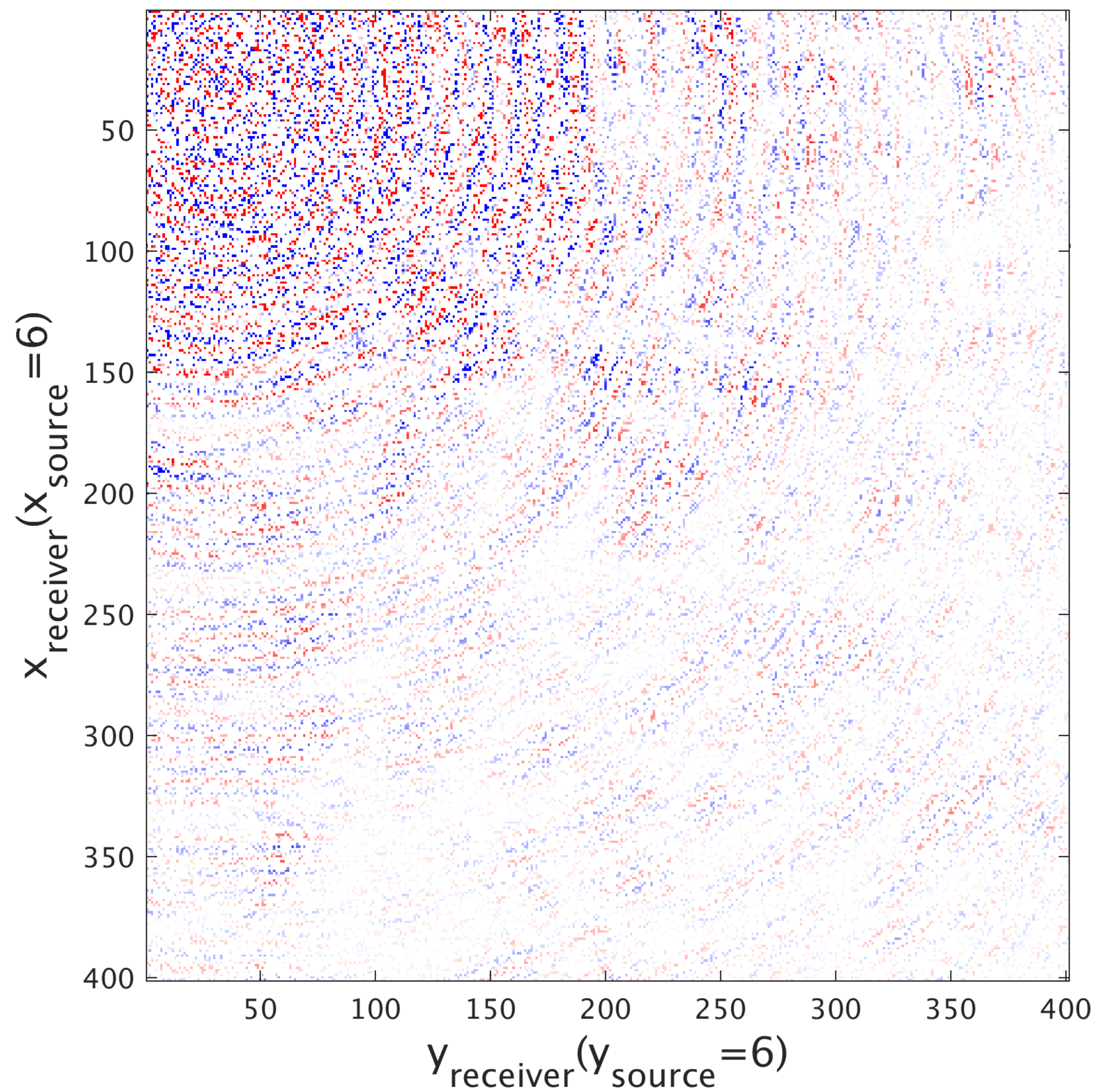
- Decoupling method

Common Source Gather

True Source Gather



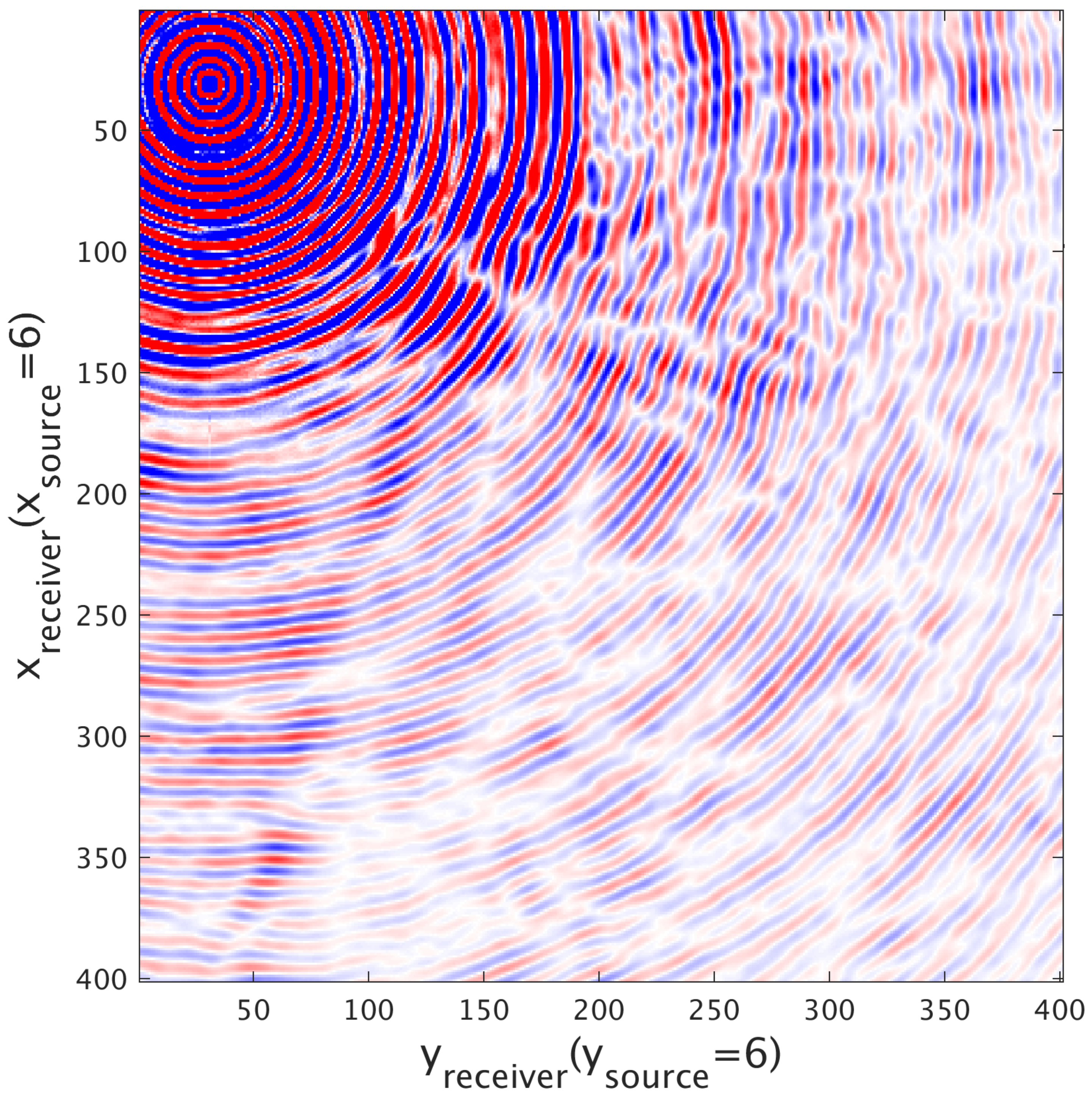
Subsampled Source Gather



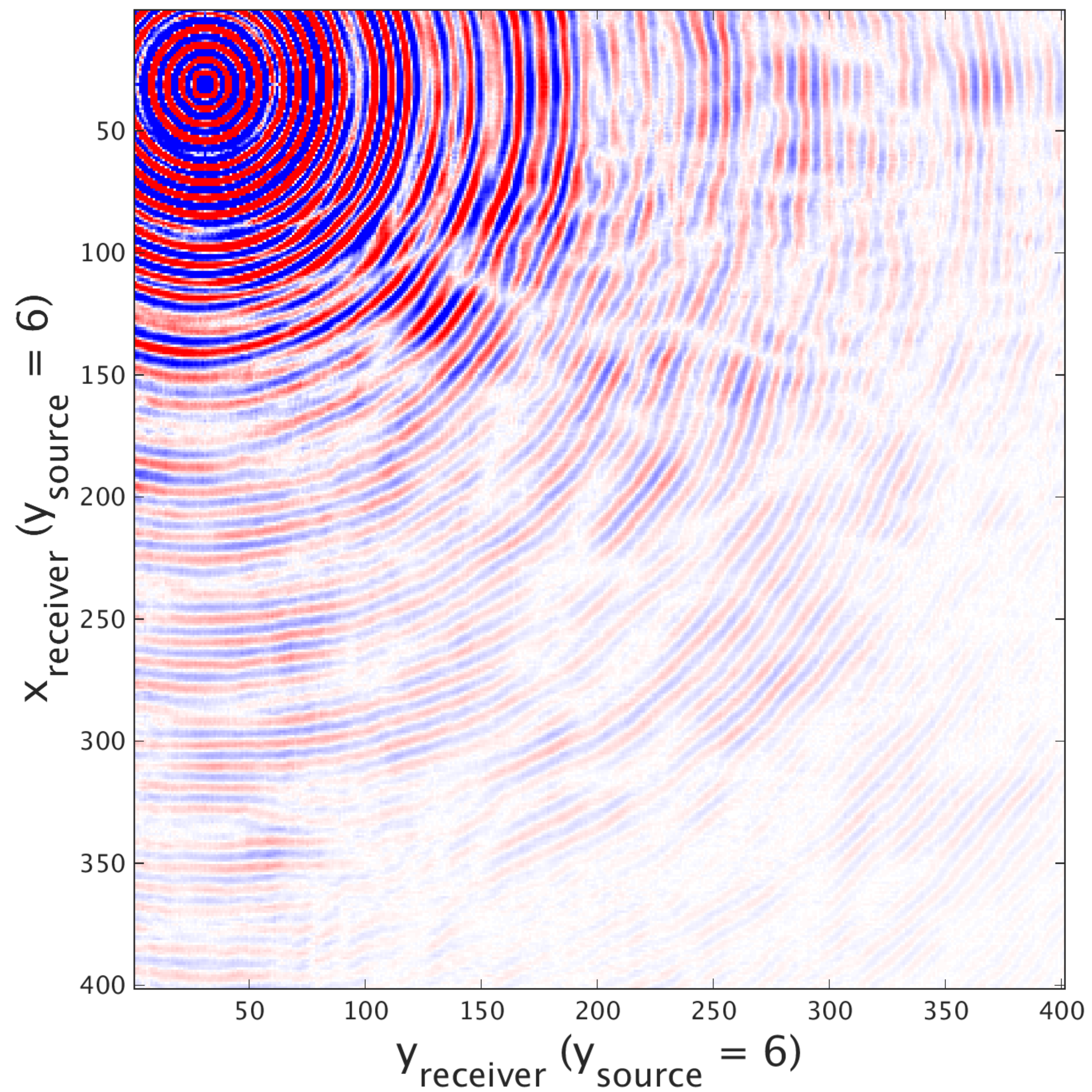
Remove 80 % of Receivers randomly

Results: SPG-LR

True Source Gather



Recovered Source Gather



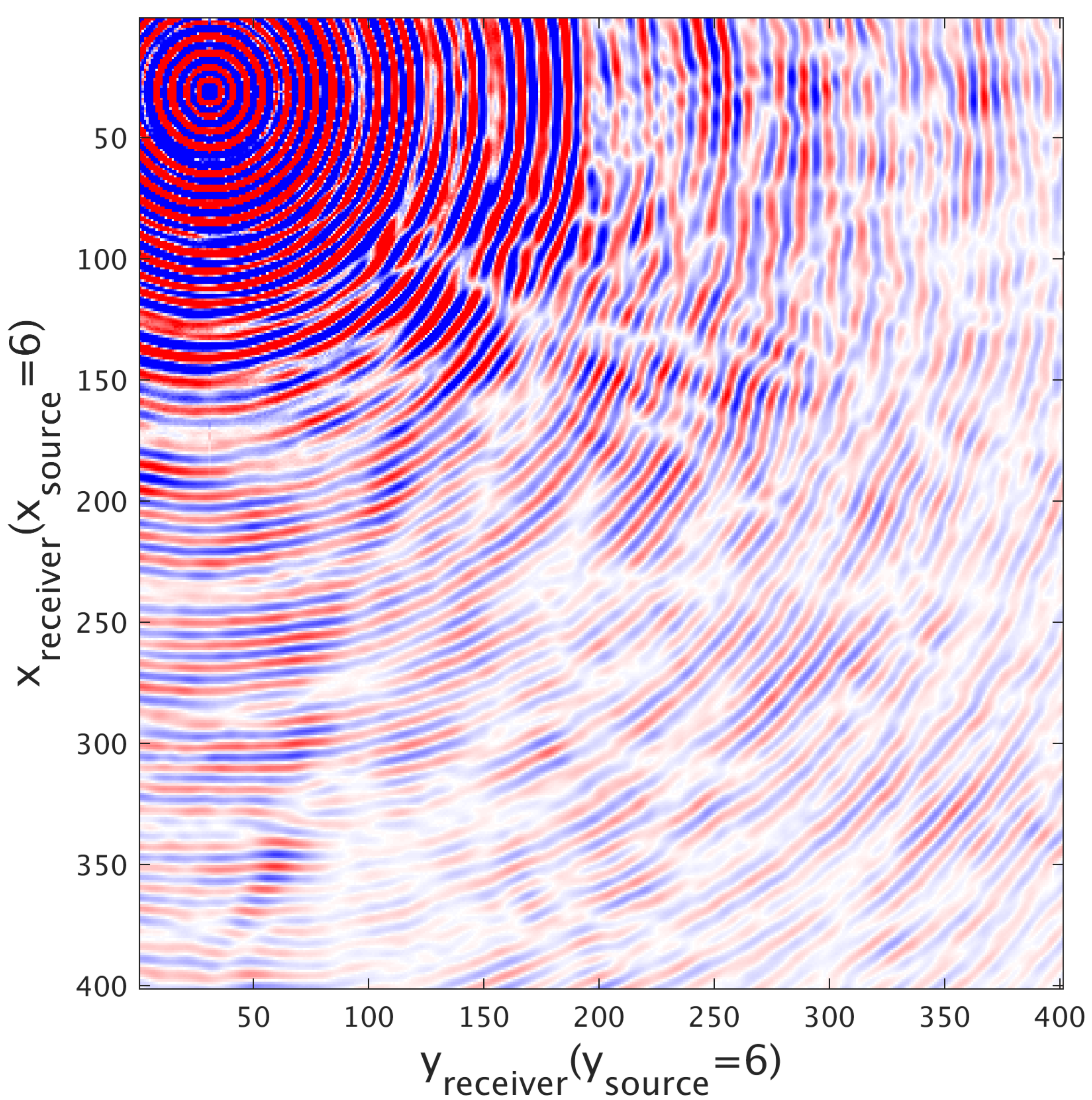
SPG-LR iterations: 400

SNR = 20.5 dB

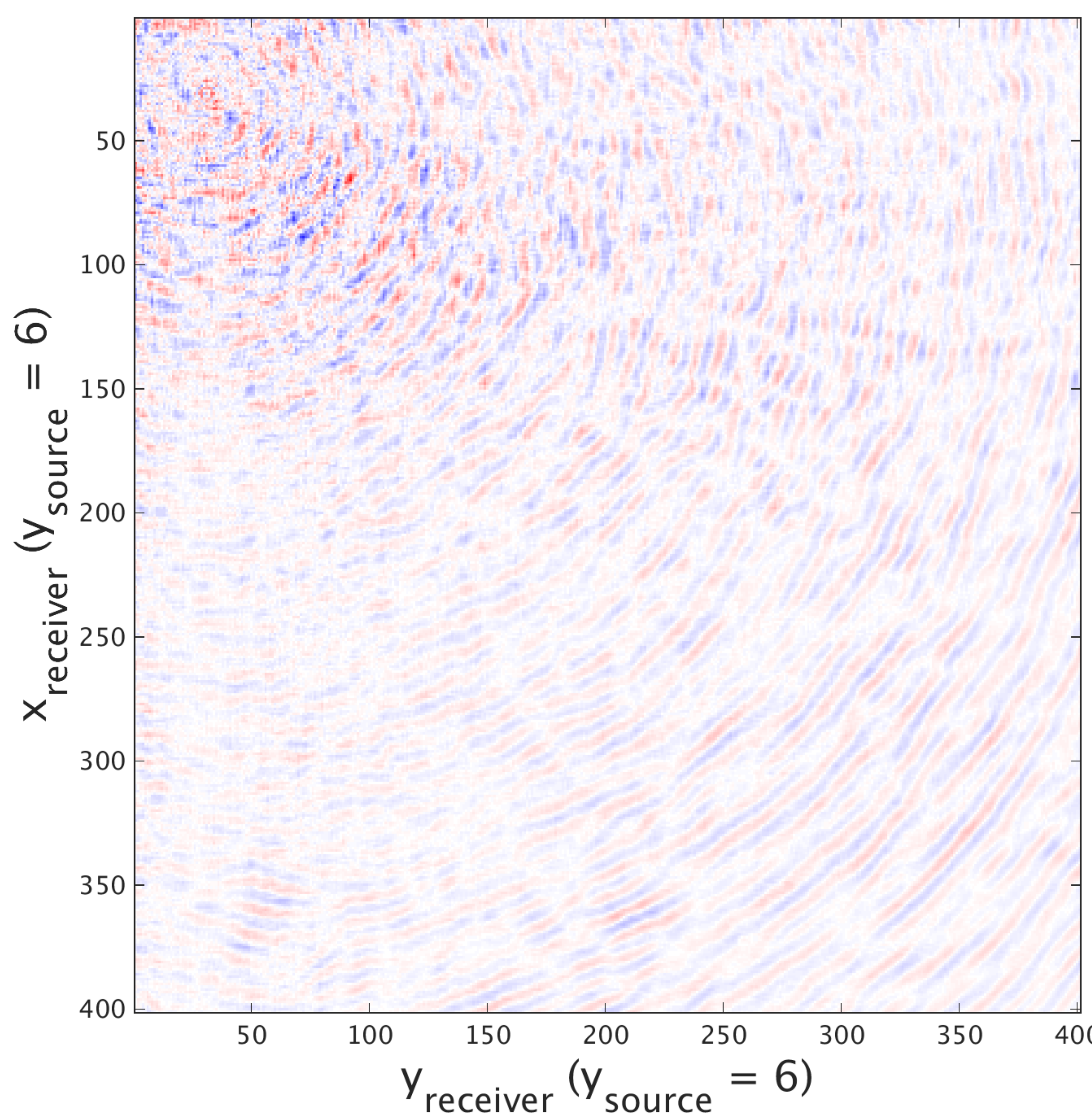
Time = 137 hrs and 20 min

Results: SPG-LR

True Source Gather



Difference Plot

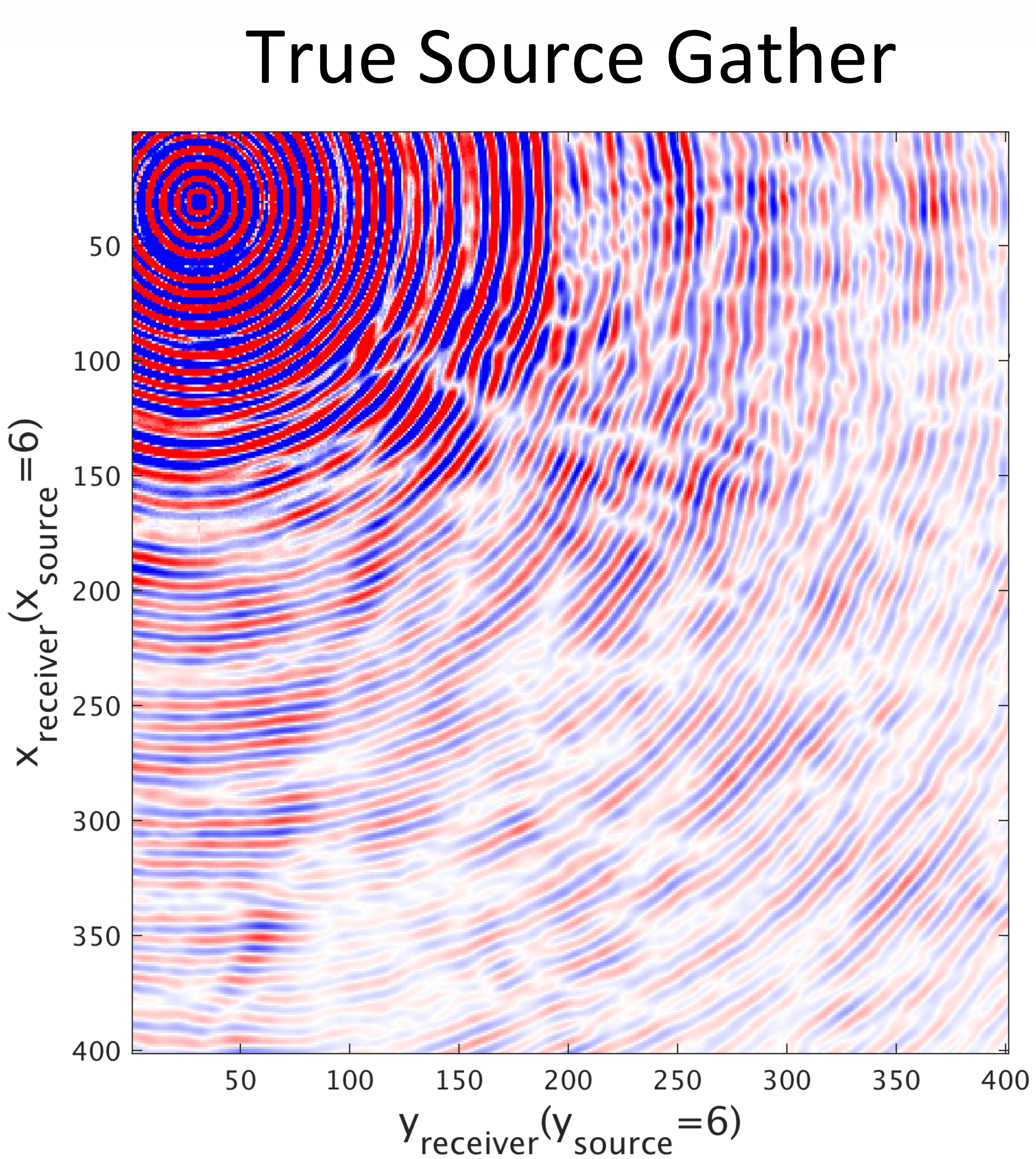


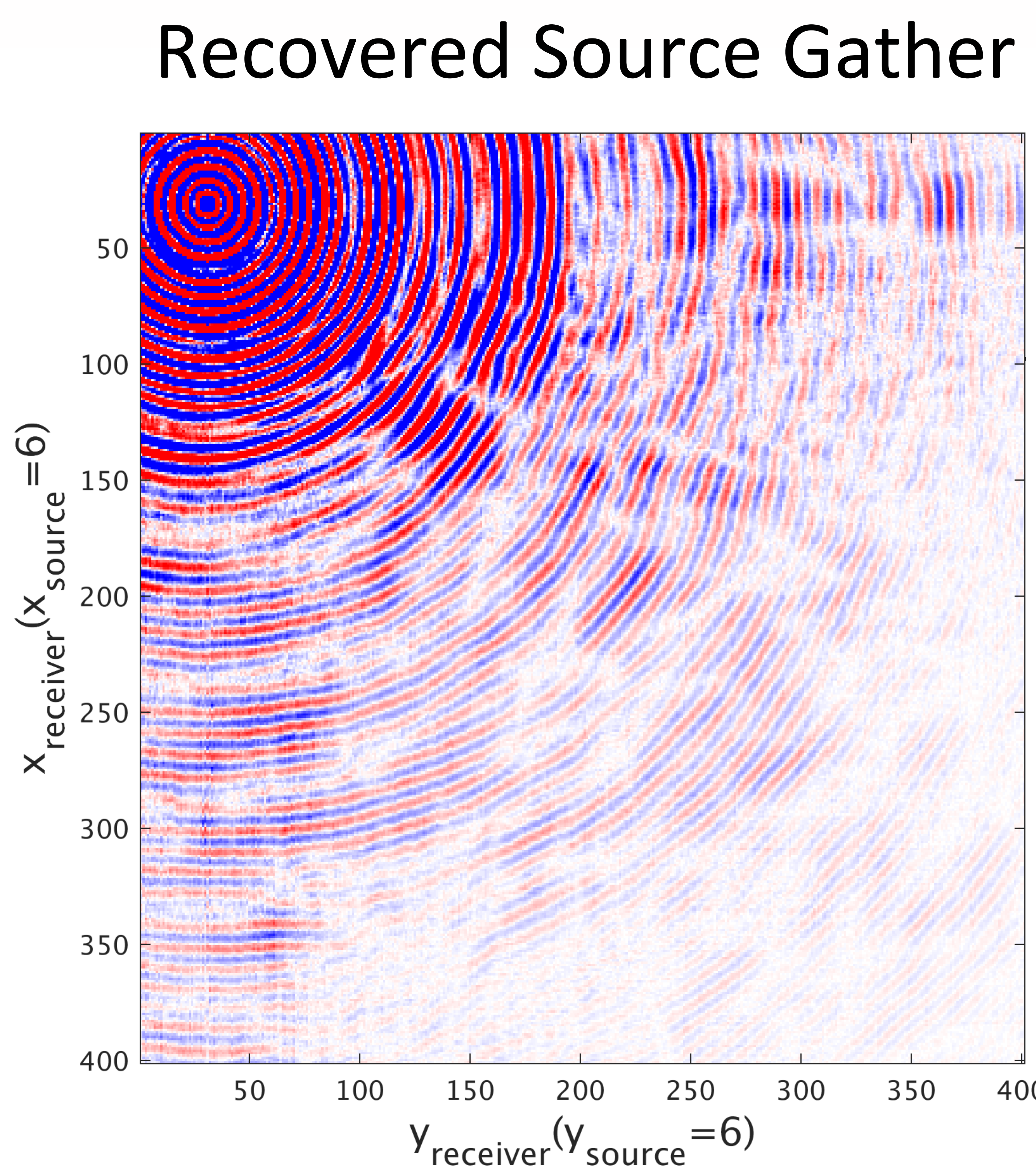
SPG-LR iterations: 400

SNR = 20.5 dB

Time = 137 hrs and 20 min

Results: Decoupling method





60 workers

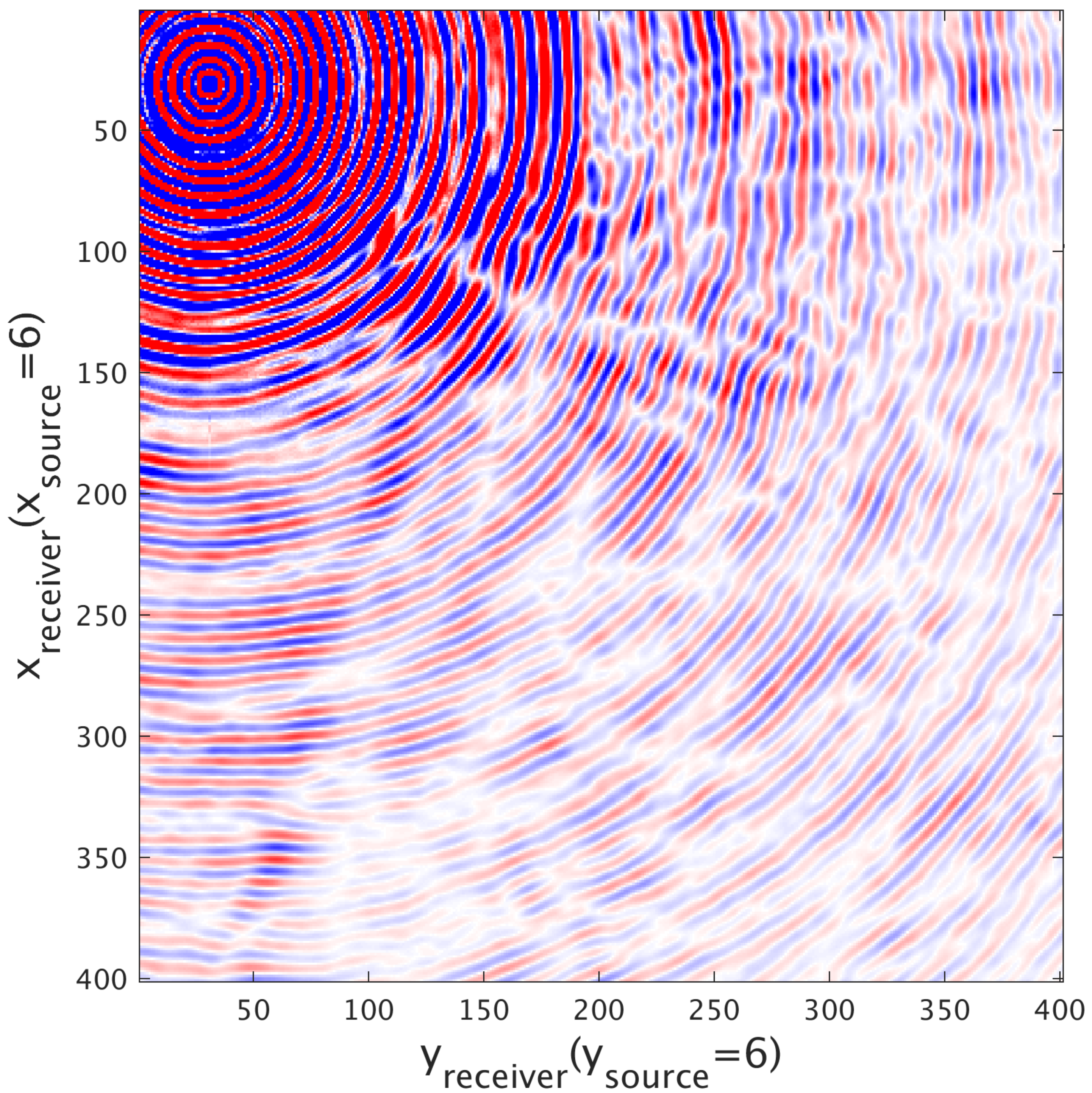
Alternations: 5

SNR = 19 dB

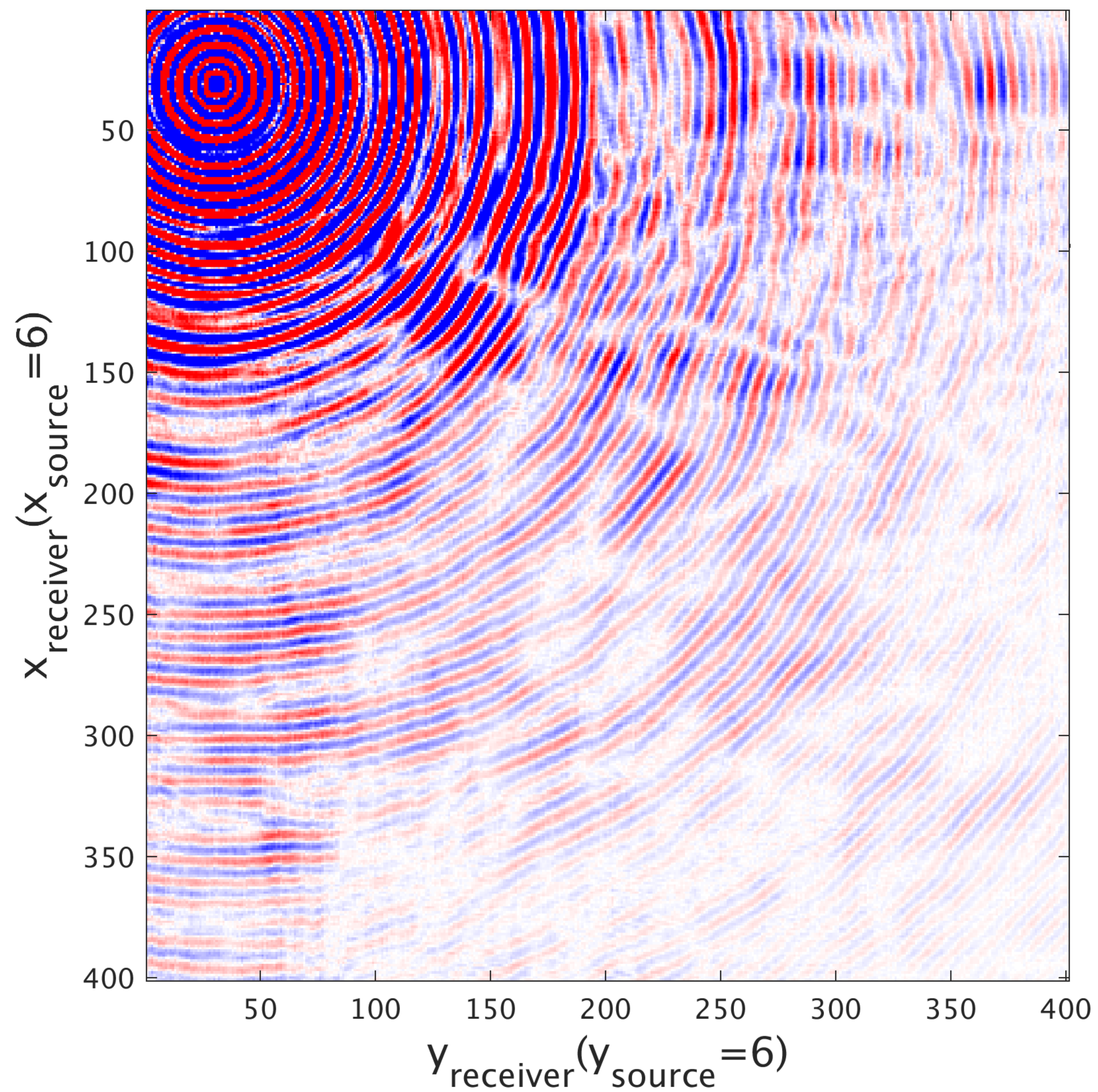
Time = 1 hrs and 7 mins

Results: Decoupling method

True Source Gather



Recovered Source Gather



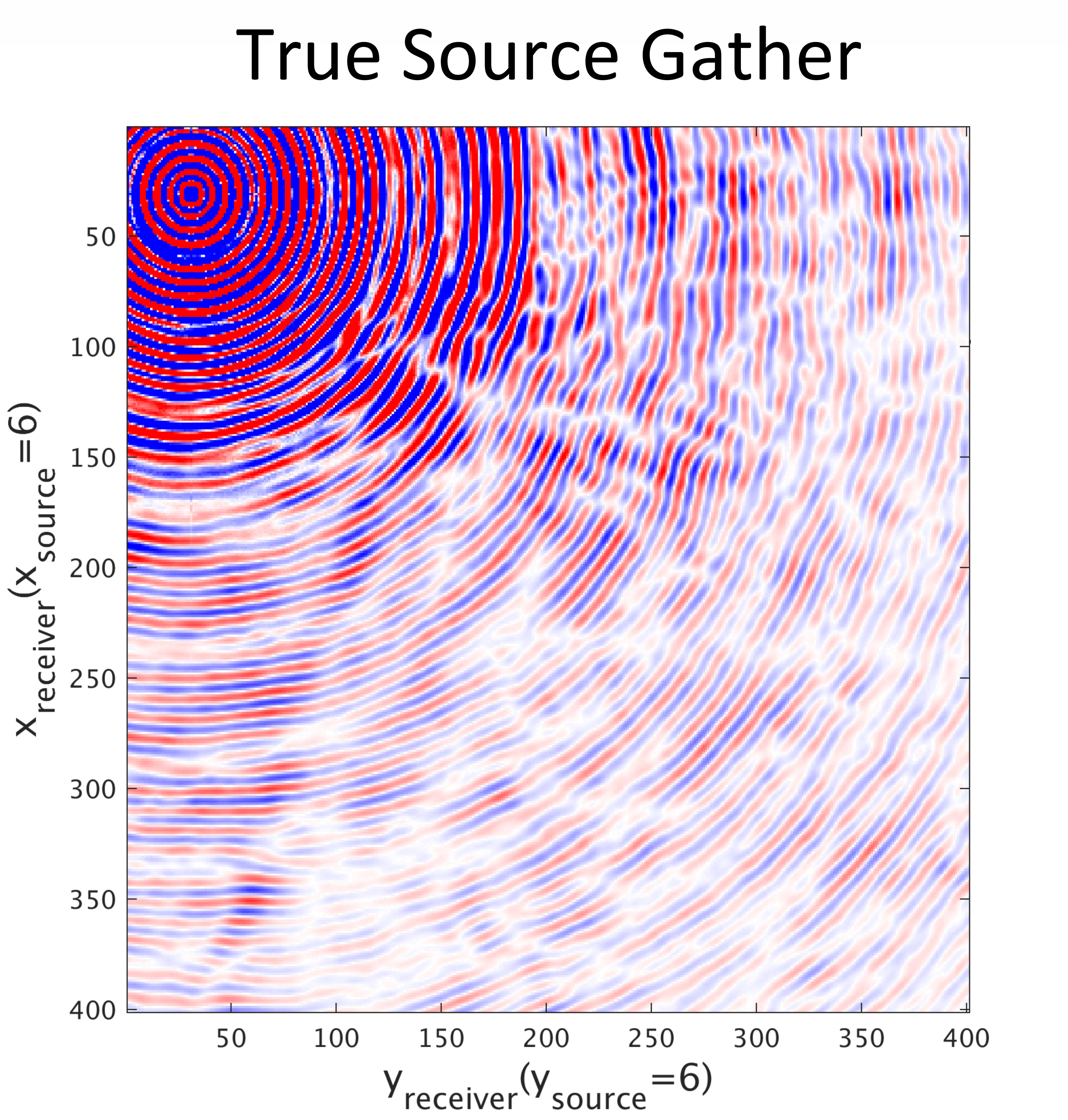
60 workers

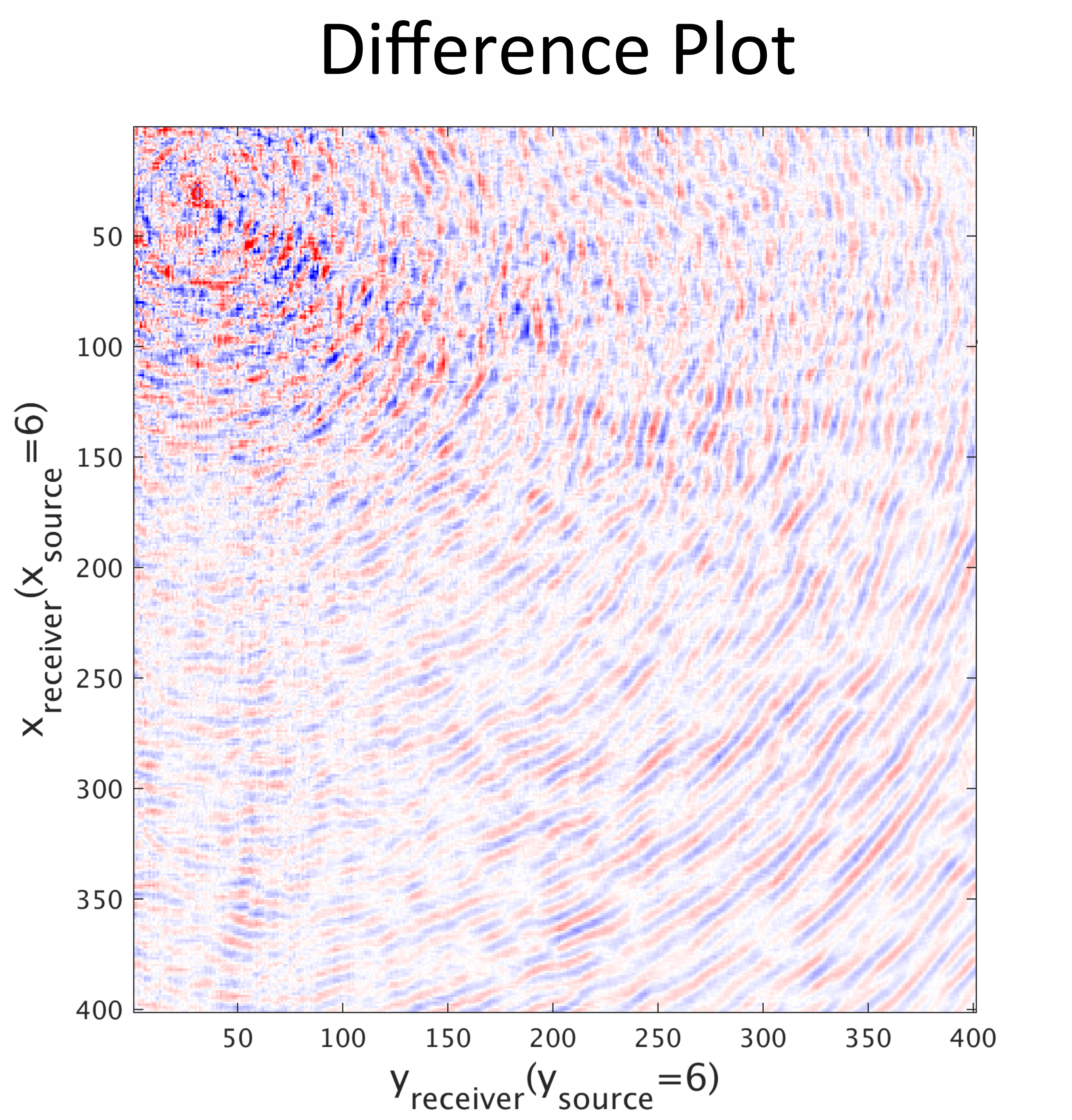
Alternations: 7

SNR = 20 dB

Time = 1 hrs and 36 mins

Results: Decoupling method





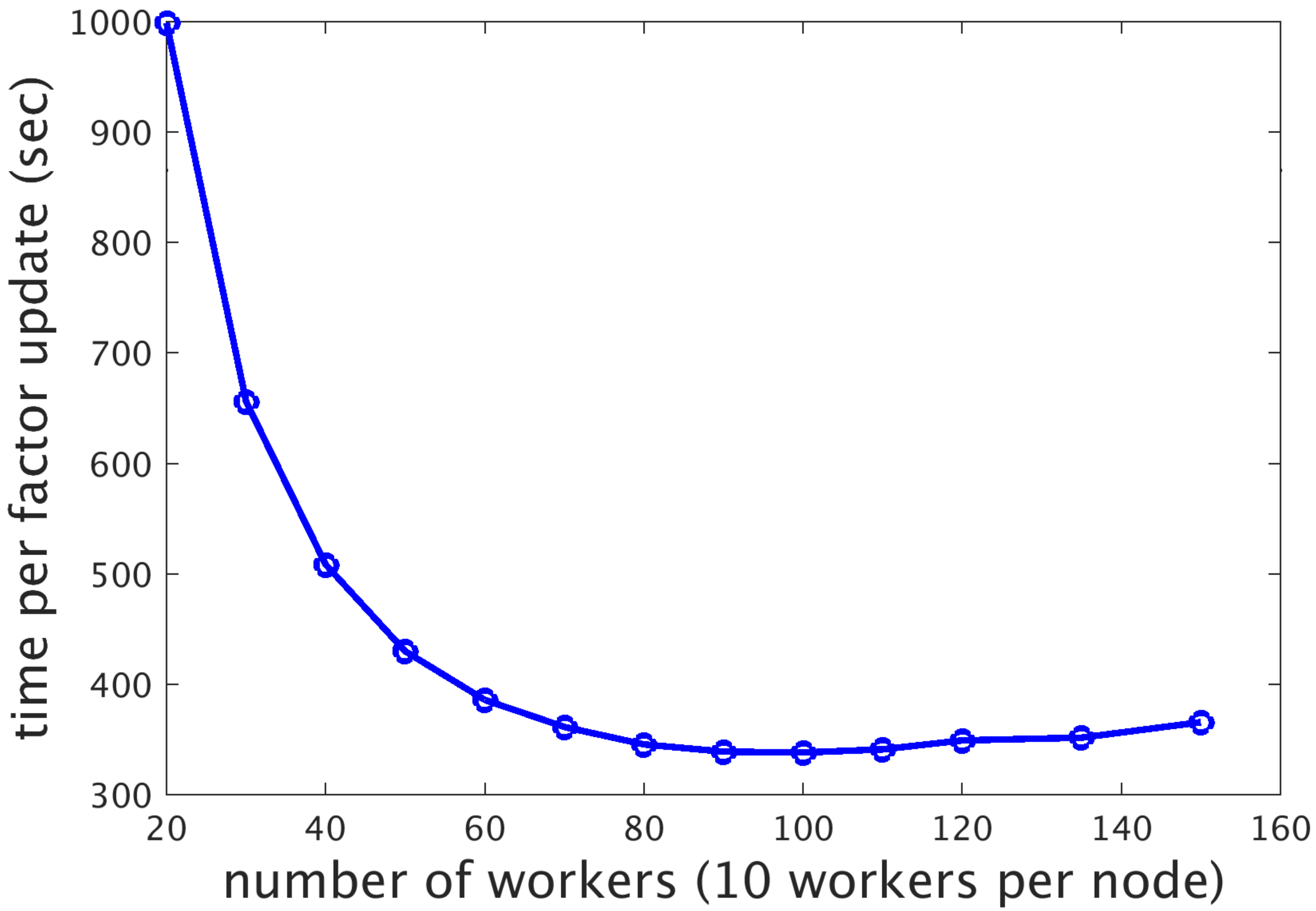
60 workers

Alternations: 7

SNR = 20 dB

Time = 1 hrs and 36 mins

Scalability: time vs # workers



Matrix Size: 27,268 x 27,268

(full slice, no windowing)

rank = 534

missing 80% receivers

To window or not to window?

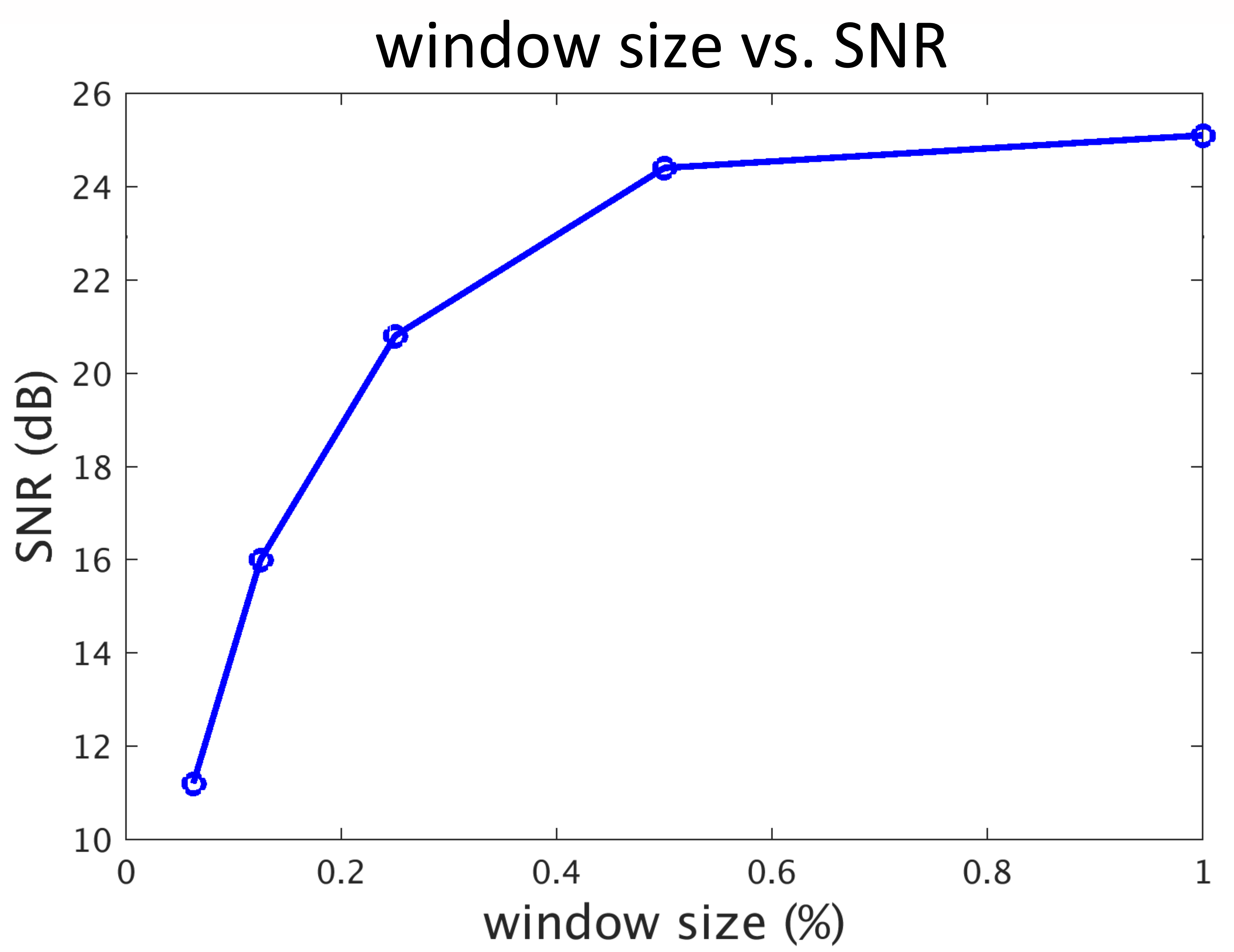
Look at recovery with various windows sizes

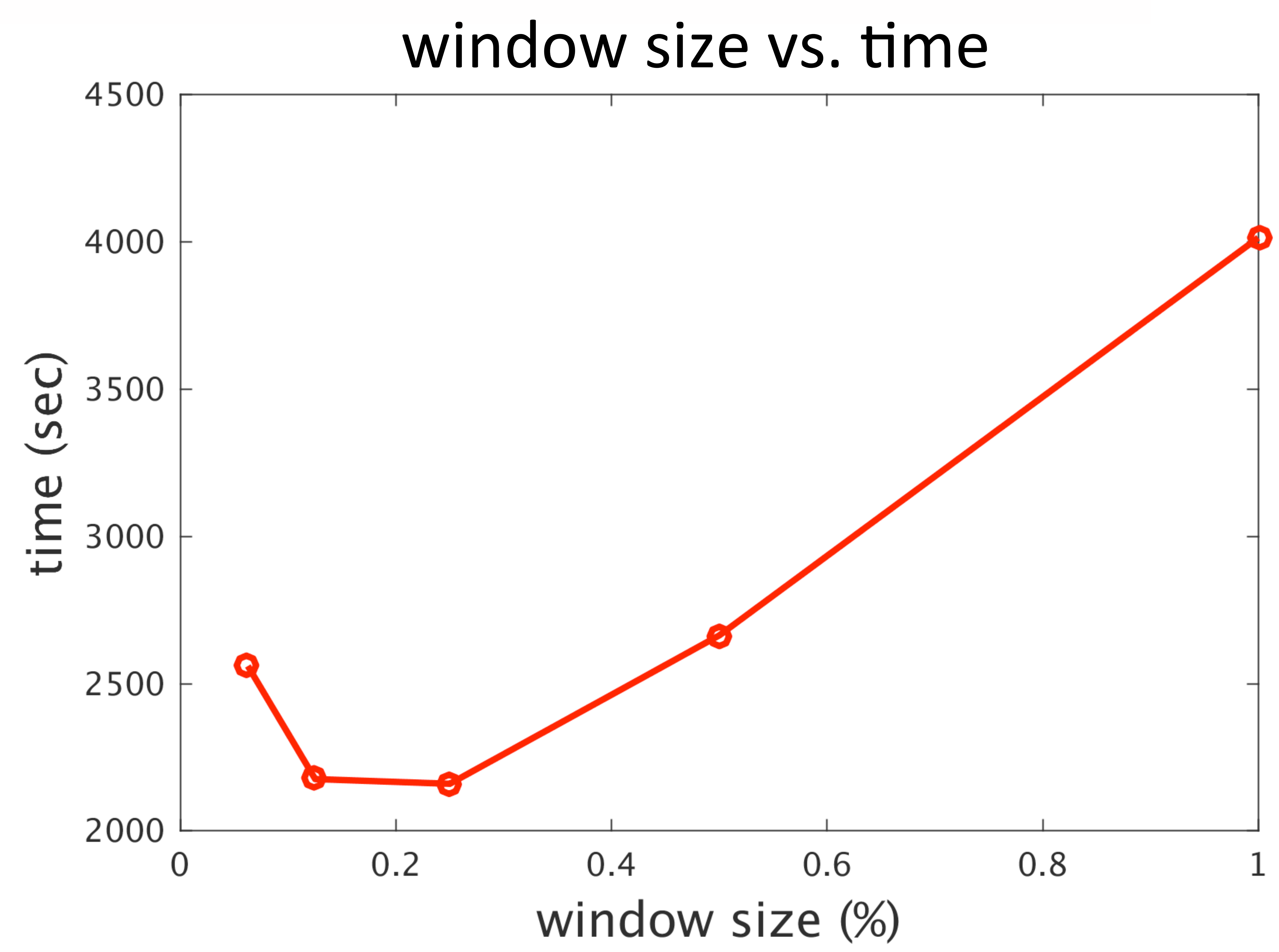
Rank chosen according to window size (as before)

Missing 80% receivers

7 Alternations

To window or not to window?





Conclusions

- Significant improvement in computation time
- Equivalent SNR output
- Optimized communication time between workers
- Parameter free

SLIM

R. Kumar, O. Lopez, D. Davis, A. Aravkin and F. Herrmann. "Beating-Level Set Methods for 5D Seismic Data Interpolation: A Primal Dual Alternating Approach"

RCAM
(Residual Constrained Alternating Minimization)

Distributed implementation to penalize norm with noise constraint

$$\mathbf{L}^t = \underset{\mathbf{L}\in\mathbb{C}^{n\times r}}{\arg\min} \|\mathbf{L}\|_F^2 \;\; \text{s.t.} \;\; \|P_\Omega(\mathbf{L}(\mathbf{R}^t)^*) - \mathbf{b}\|_F \leq \eta$$

$$\mathbf{R}^{t+1} = \underset{\mathbf{R}\in\mathbb{C}^{m\times r}}{\arg\min} \|\mathbf{R}\|_F^2 \;\; \text{s.t.} \;\; \|P_\Omega(\mathbf{L}^t\mathbf{R}^*) - \mathbf{b}\|_F \leq \eta$$

- Avoid overfitting noise
- Robust (e.g., not sensitive to overshooting rank)
- time(ALS) < time(RCAM) << time(SPG-LR)

Future Work

- Design for other measurement operators, $\mathcal{A}$.
  - incorporate "off-the-grid" measurements
  - source separation

Acknowledgements

This research was carried out as part of the SINBAD project with the support of the member organizations of the SINBAD Consortium.

Software release available

https://github.com/SINBADconsortium/RCAM.jl