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

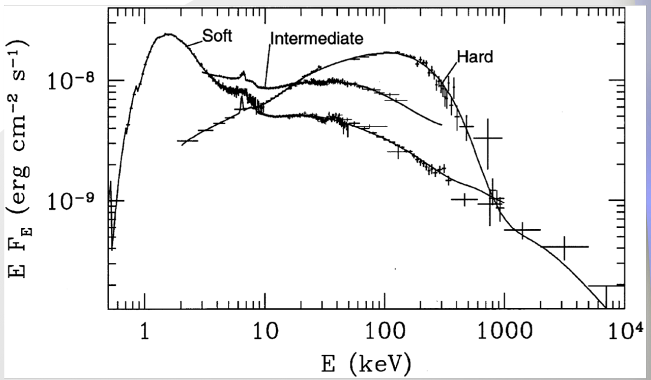
Stingray 3.0: A parallel Python library for spectral-timing

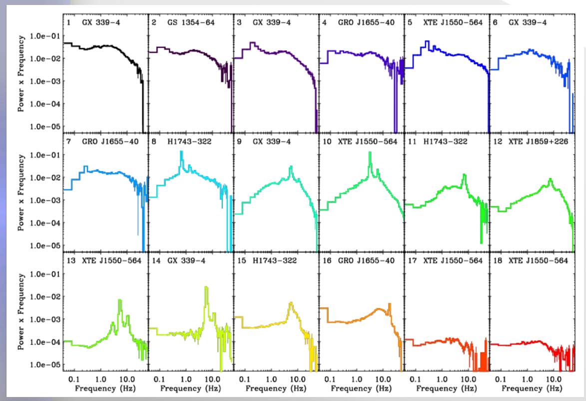
Eleonora Veronica Lai, Matteo Bachetti, Maura Pilia,
+ Daniela Huppenkothen and Stingray developers

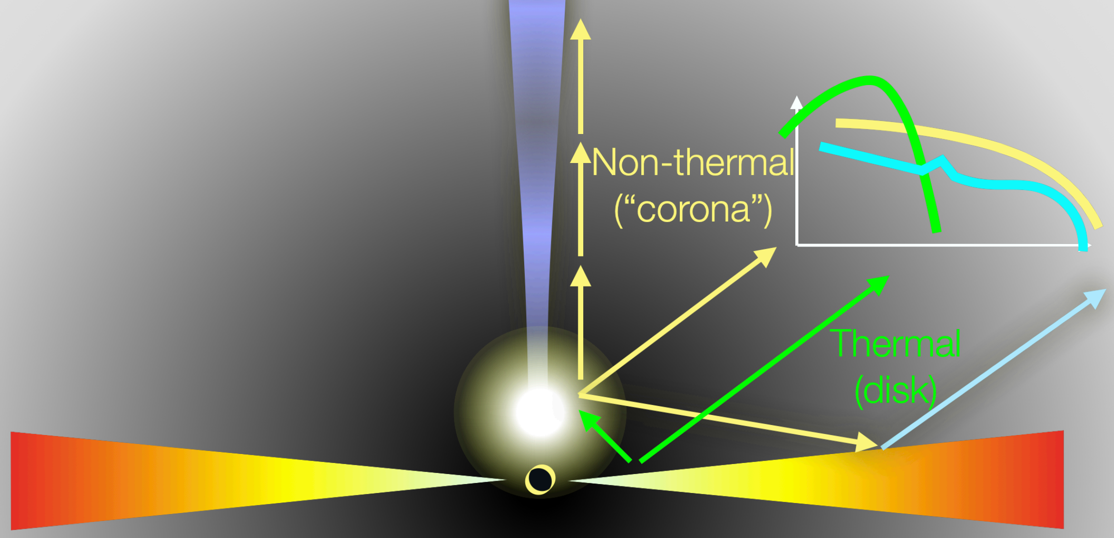
Spoke 3 II Technical Workshop, Bologna Dec 17 -19, 2024

Scientific Rationale

- Some observe spectra, some observe variability. Is it possible to use the full information?
- Example: a variable accretion flow that **propagates** through an atmosphere (corona), that **illuminates** the accretion disk and gets **reflected**. Can we disentangle the emission regions?
- Stingray: ease the learning curve for advanced spectral-timing techniques, with a correct statistical framework







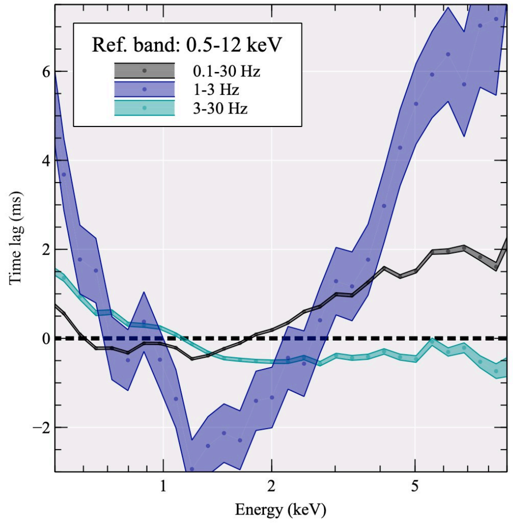
Huppenkothen et al. (2019)

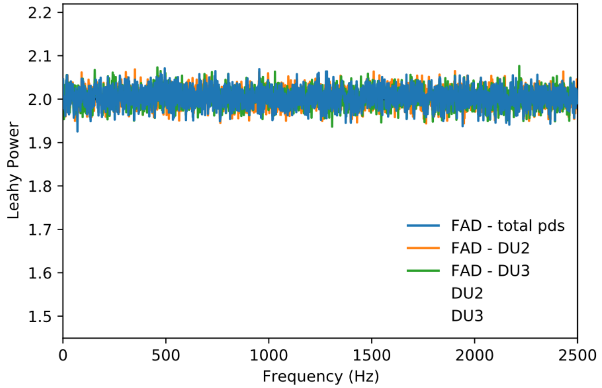
Technical Objectives, Methodologies, and Solutions: what is able to do

- **"Timing" analysis**
  - Pulsation searches and timing
  - Aperiodic variability, periodogram modelling (ML, Bayesian)
- **Spectral analysis -> connect to Xspec, Sherpa**
  - Continuum modeling
  - Broad lines (e.g. Fe complex, cyclotron lines)
- **Polarimetry**
- **Spectral-timing techniques:**
  - Time/Phase lags
  - Coherence
  - Covariance
  - etc.

… all with instrument awareness

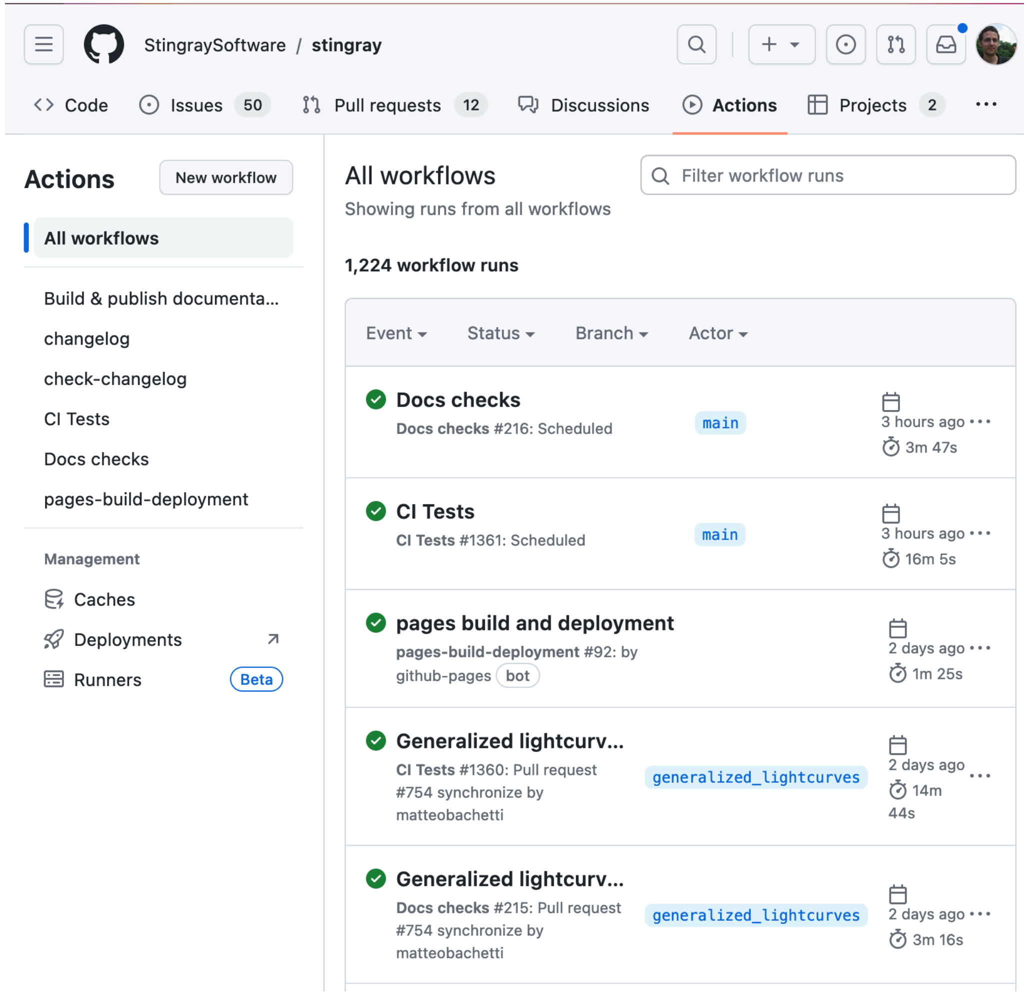
- Be aware of instrumental systematics: dead time, frame time, good time intervals, etc.
- Mission support



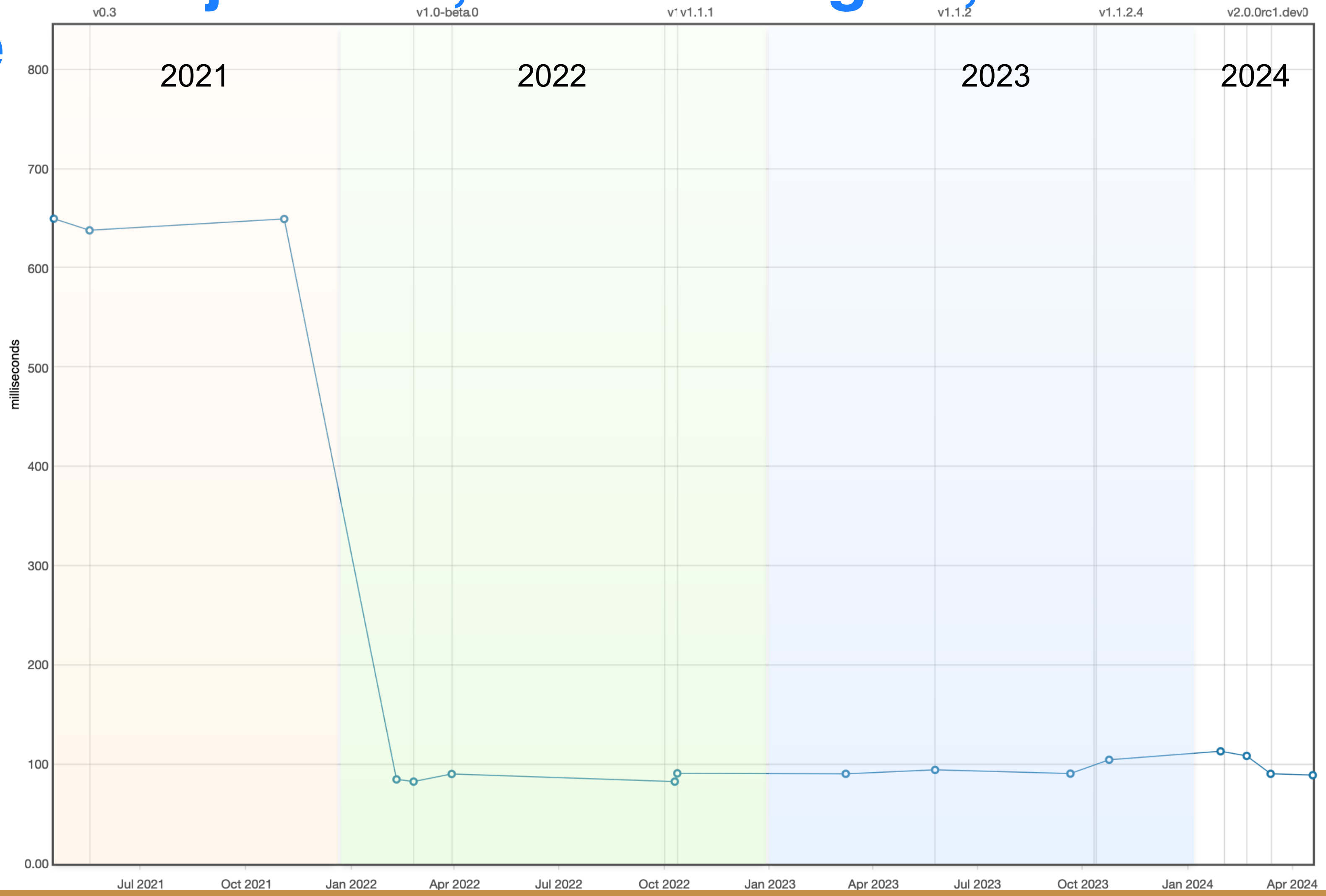


Technical Objectives, Methodologies, and Solutions: reliability and performance testing

- Code correctness
  - Test-based development
  - Literature reproduction
- Regression testing: continuous integration with **Github Actions** and **tox**
  - **Unit tests**
  - **Integration tests**
- Performance
  - Profiling: **line_profiler**, %**time**, **memory_profiler**, etc.
  - Small-dataset testing (< RAM): verify “acceptable” execution times
  - Scalability for larger-than-RAM datasets
- Documentation
  - Use **Sphinx** + **Github Actions** for automatic docs building
  - **Linkcheck** for periodic link checking in the docs



Technical Objectives, Methodologies, and Solutions: in time



Simple benchmarks to track Stingray's performance comparing the different releases through the years to today!

Main Results

-Large data handling

-Parallelization

Main Results

- Large data handling

The large dataset typically recorded in X-rays can last up to a few days (i.e. billions of time bins in a light curve), **but**:
is it necessary to store all of it in memory?

FITSTimeseriesReader (class)

Which loads FITS files as a memory-mapped array and reads the data in chunks.

At the base of the….

Main Results

- Parallel implementations

A lot of **Stingray**'s analysis procedures are based on parallelizable functions: e.g. **Bartlett's periodogram**

This method consists of:

1) time series is cut in equal time segments (or chunks)

2) calculates the FFT of each segment and compute the periodogram (i.e. the square modulus)

3) average of all periodograms

Main Results

- Parallel implementations

A lot of **Stingray**'s analysis procedures are based on parallelizable functions: e.g. **Bartlett's periodogram**

This method consists of:

1) time series is cut in equal time segments (or chunks)

2) calculates the FFT of each segment and compute the periodogram (i.e. the square modulus)

Parallelizable!

3) average of all periodograms

Main Results

- Parallel implementations

A lot of **Stingray**'s analysis procedures are based on parallelizable functions: e.g. **Bartlett's periodogram**

This method consists of:

1) time series is cut in equal time segments (or chunks)

2) calculates the FFT of each segment and compute the periodogram (i.e. the square modulus)

3) average of all periodograms

Parallelizable!

→ Multiprocessing

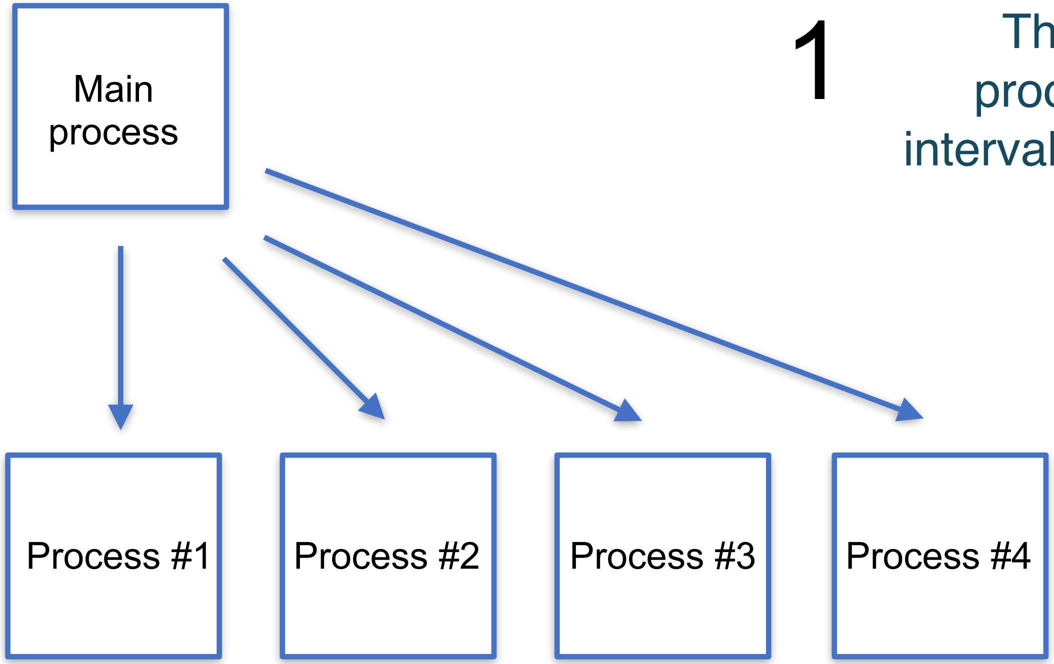
→ MPI

Main Results

-Multiprocessing implementation

Main Results

- Multiprocessing implementation



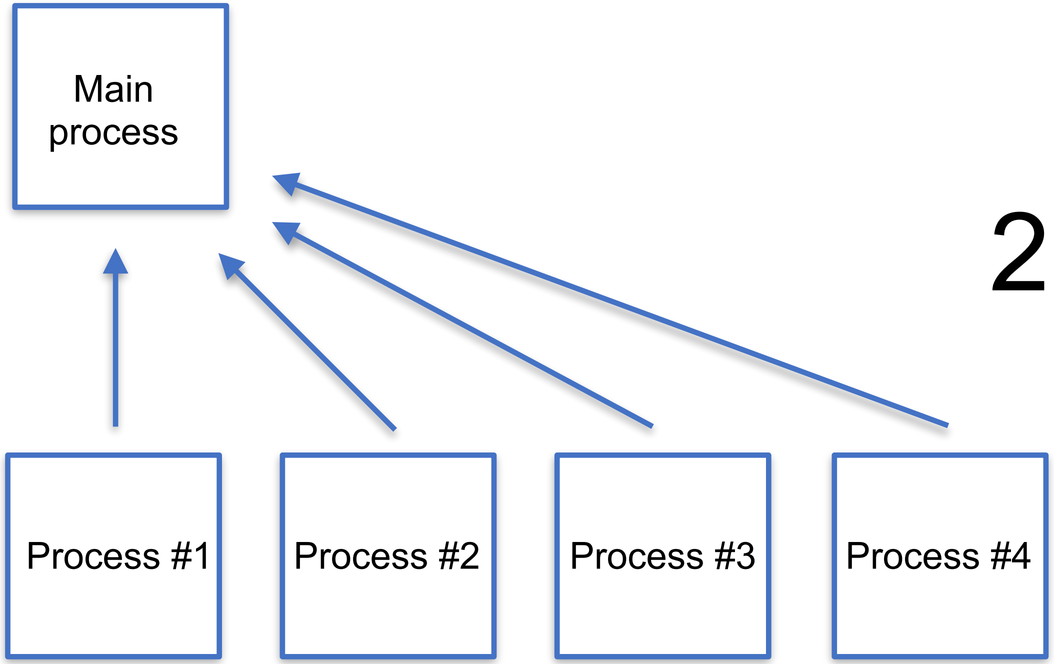
1

The main process spawns *M* processes and splits the *N* time intervals among them in an unordered way

$= M$ t_spawn

Main Results

- Multiprocessing implementation



2 Nodes calculate the periodograms for every single time interval, summing them and send back the results to the main process

$$= N/M\ t\_per + (N/M - 1)\ t\_sum + M\ t\_comm$$

Main Results

- Multiprocessing implementation

Main process

3 The main process calculates the weighted average periodogram

= (M -1) t_sum

Process #1

Process #2

Process #3

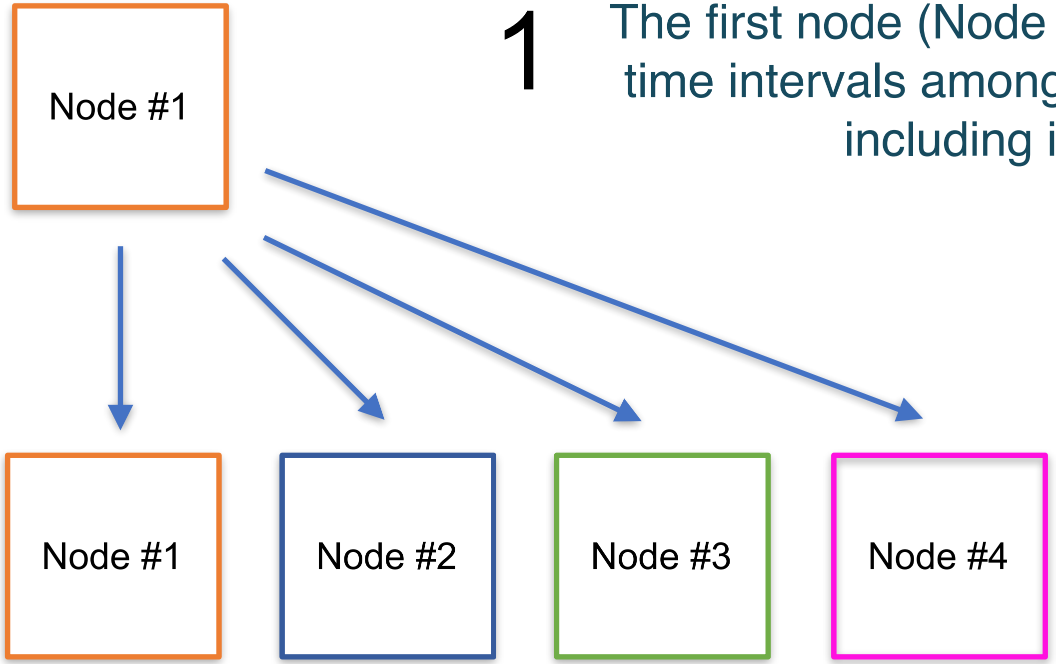
Process #4

Main Results

-MPI implementation

Main Results

- MPI implementation



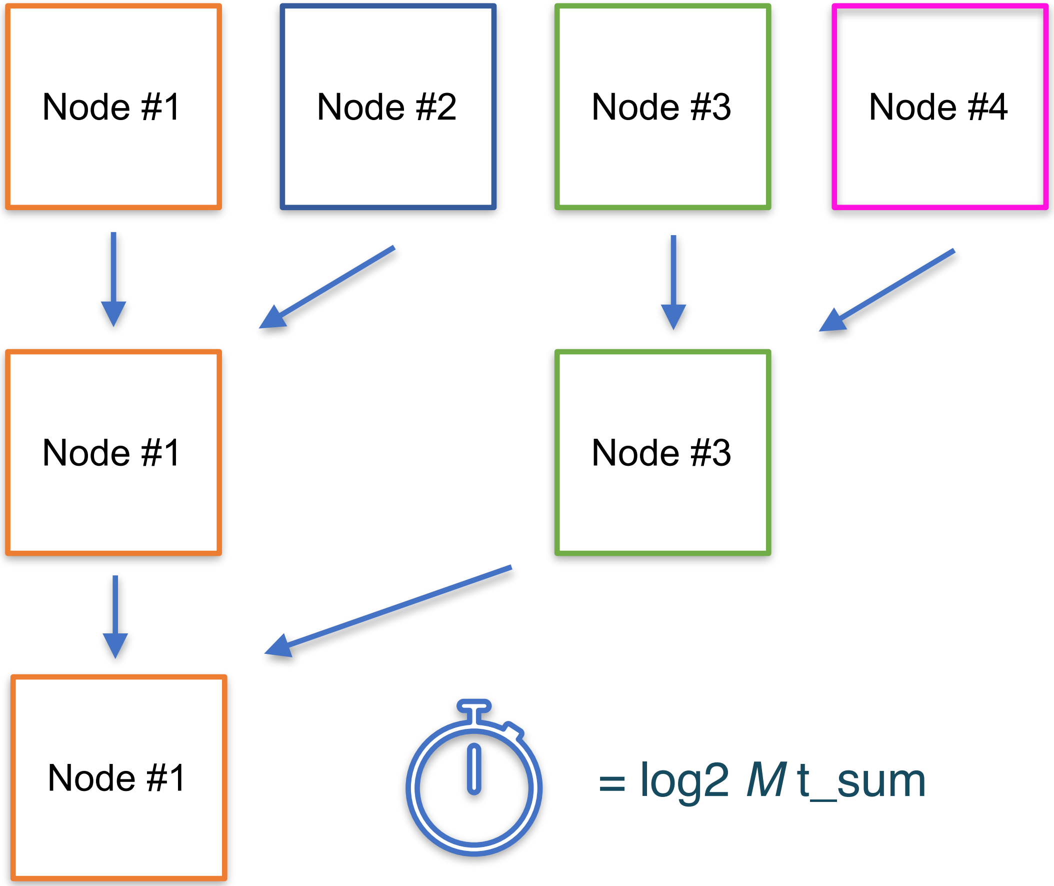
1 The first node (Node #1) splits the N time intervals among all the nodes, including itself

2 As in multiprocessing, each node calculates a Bartlett periodogram for its time intervals

= N/M t_per

Main Results

- MPI implementation



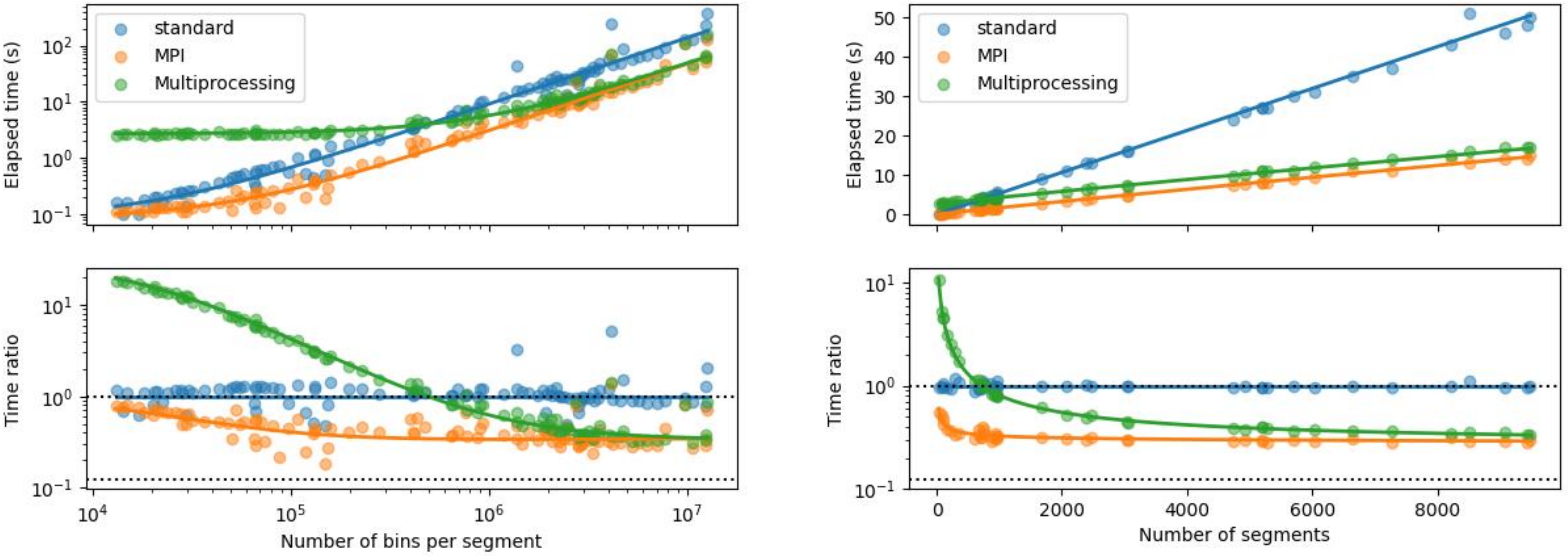
= log2 *M* t_sum

3 Nodes do not communicate their results to the first node, but they undergo instead to some intermediate steps where pairs of results are averaged together

= (N/M -1) t_sum + log2 *M* t_comm

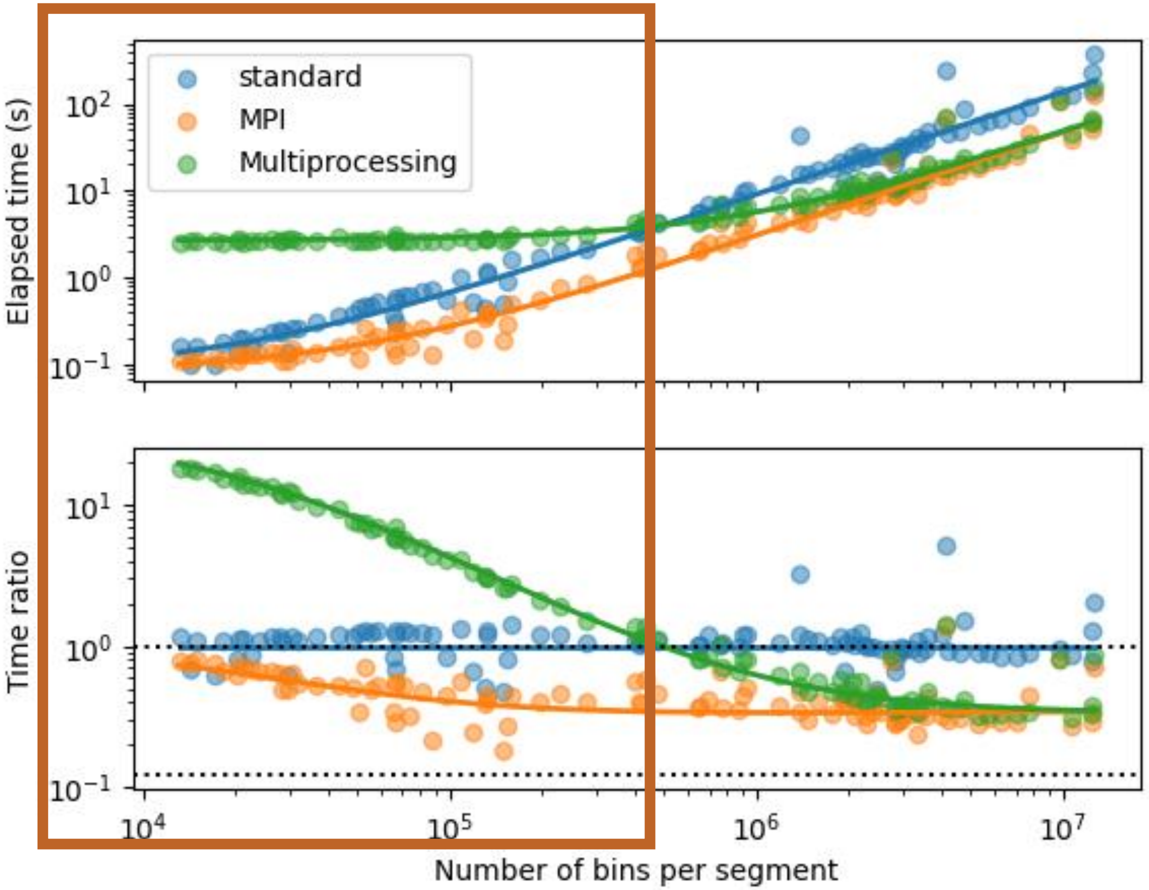
Main Results

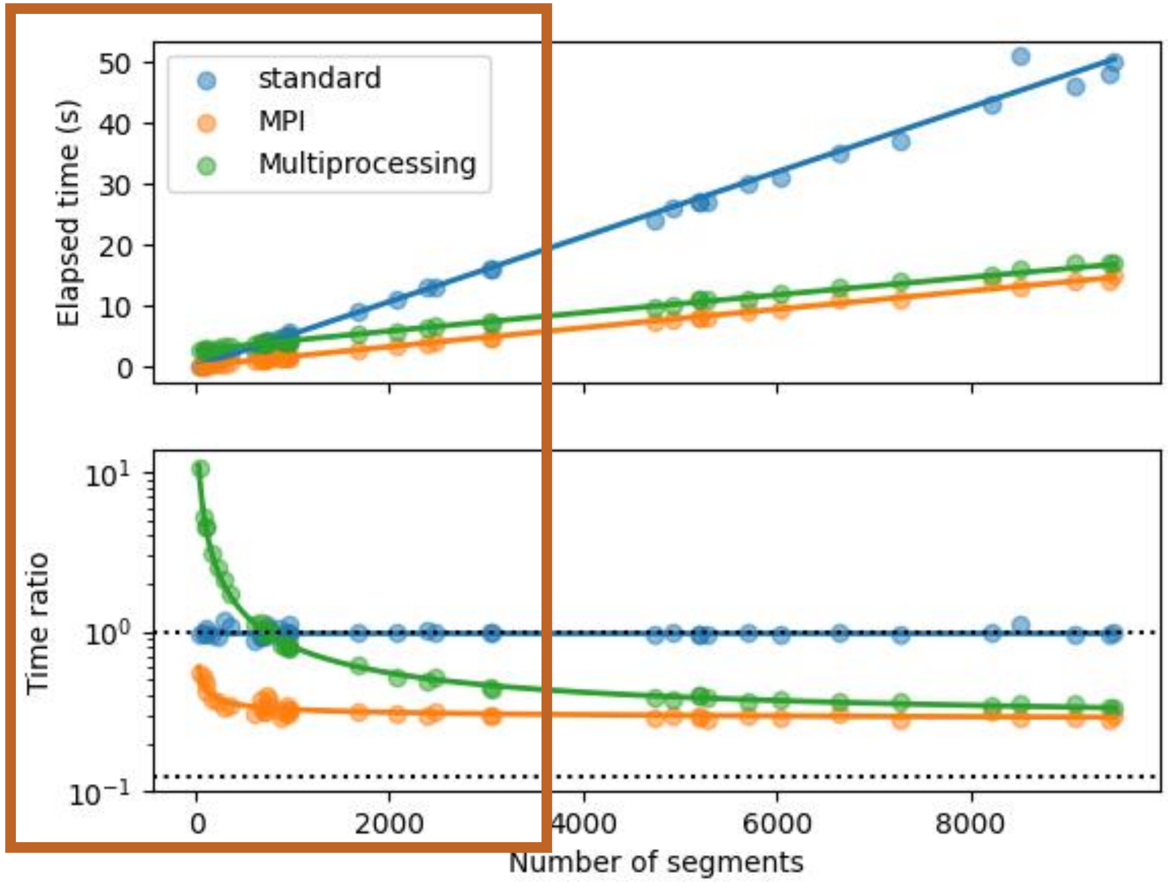
Comparison of performance on simulated data compared to the serial approach



Main Results

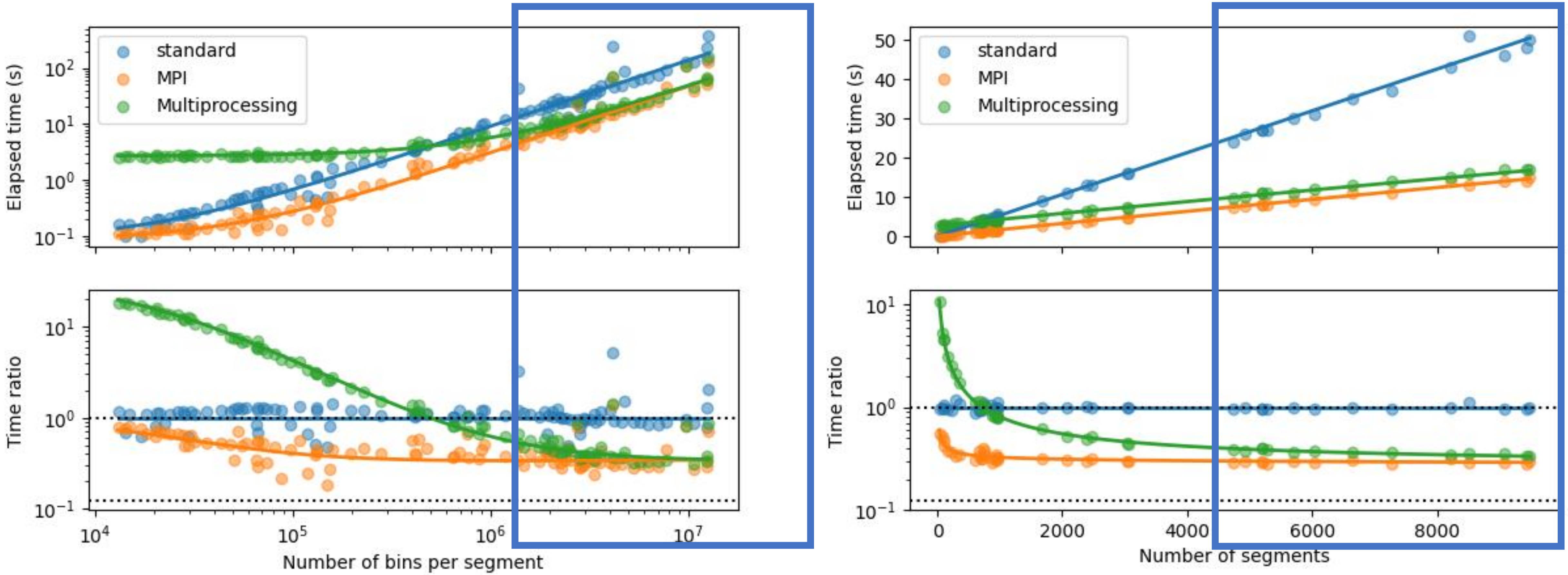
Comparison of performance on simulated data compared to the serial approach





Main Results

Comparison of performance on simulated data compared to the serial approach



Final steps

Final Steps

- **Main question to answer from *UGO* and *GIULIANO*:**

Percentage of completion? Degree of advancement?

We completed 60% of the project

M10

To reach 90% (considering 10% for adjustments)

Final Steps

- Next steps for M10:

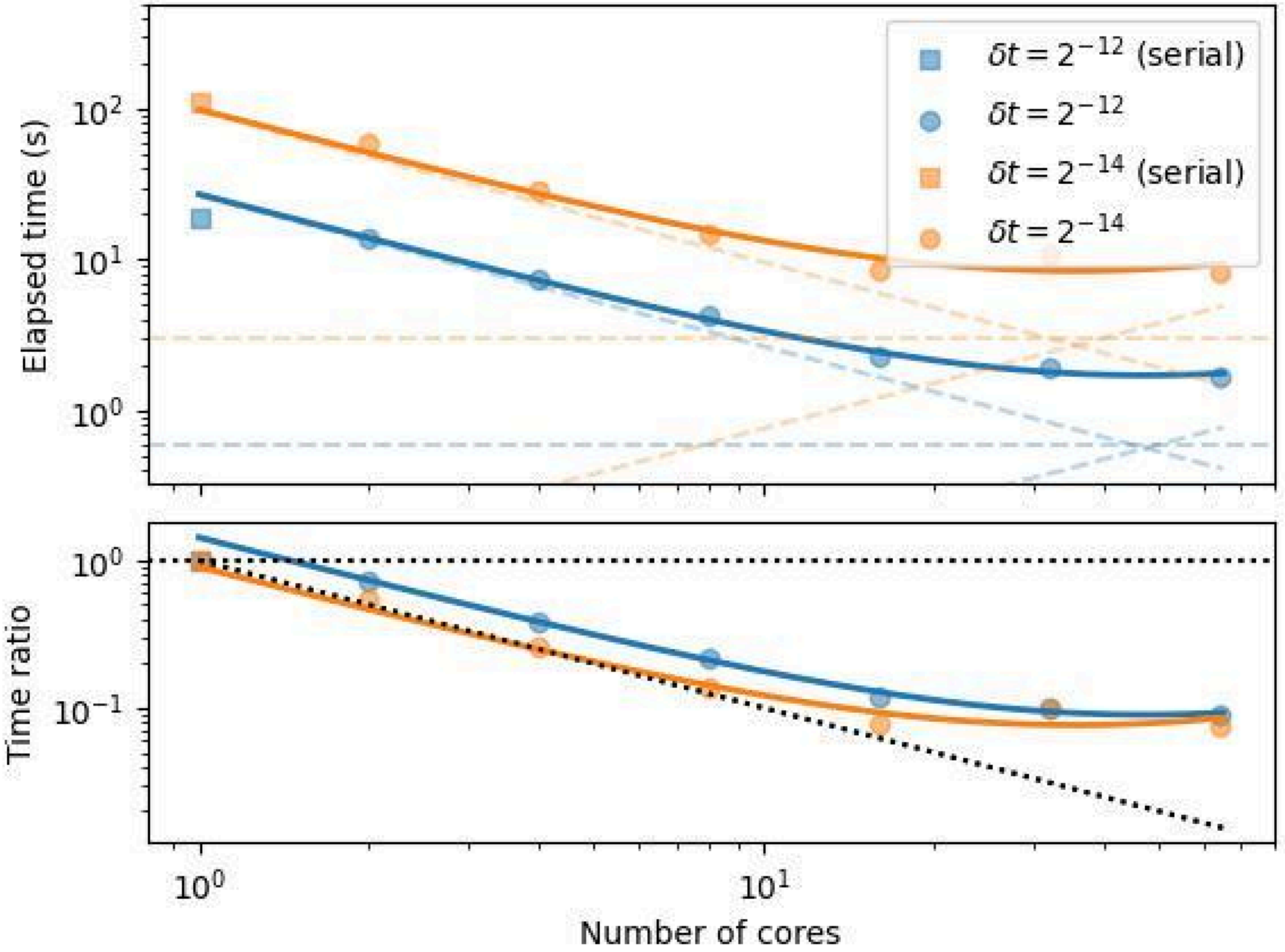
1) Include the parallel implementations in the codebase

2) Cluster to test the scalability with increasing number of cores (CED at OACa)

3) Porting of some functions (i.e. *histogram*, *get_flux_from_iterable*) on GPUs

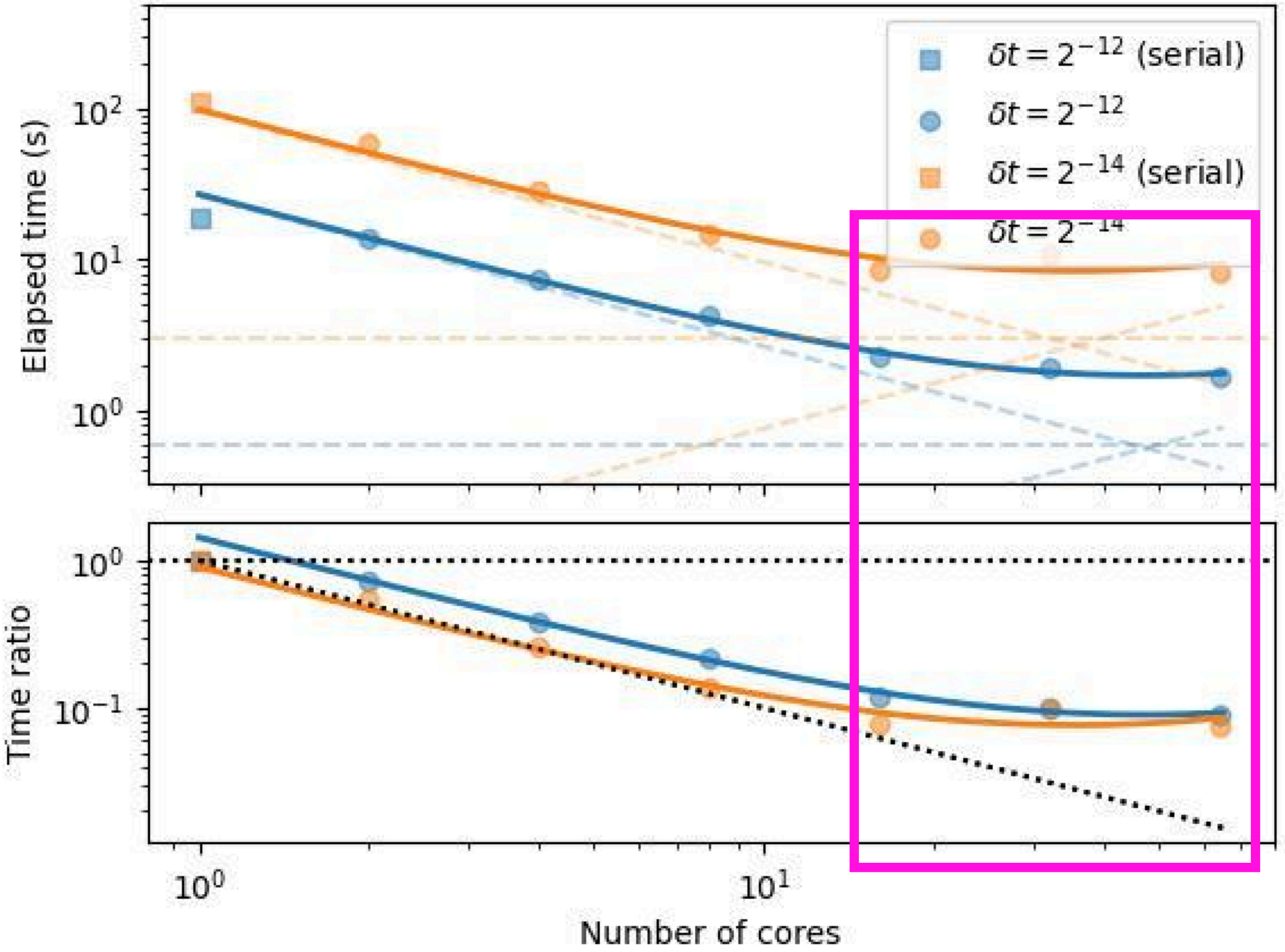
- Preliminary results

Preliminary scalability tests of the MPI implementation
using the cluster at OACa
using up to 64 cores



- Preliminary results

Preliminary scalability tests of the MPI implementation
using the cluster at OACa
using up to 64 cores



Linear behaviour
up to the plateau at >10 cores

What happens at #cores > 64?
Still plateau? Dependence on the # of cores?

- **Main questions to answer from our WP leaders:**

  1) Are the results in line with timescale, milestones and KPIs identified?

YES!

And you can find all the latest steps in the ICSC repository

high-performance-stingray Public

Forked from StingraySoftware/stingray

Anything can happen in the next half hour (including spectral timing made easy)!

Python · 0 · MIT · 147 · 2 (2 issues need help) · 0 · Updated last week

https://github.com/ICSC-Spoke3/high-performance-stingray

- **Main questions to answer from our WP leaders:**

2) Could you complete the project by February 2026? If not, why?

We can say that **most likely** we will be **able to complete** the project by February 2026.

3) What are the key bottlenecks or critical issues preventing timely completion?

Up to now, we found the relevant **bottlenecks** in the code that we **managed to "correct"**. **Some other** bottlenecks **could appear** in the next steps.

4) Additionally, what resources or support would be needed to ensure that the project is finished on schedule?

Up to now we tested the code in our own machines, but we started using the cluster to test the code for an increasing number of cores. Provided that our tests in our cluster succeed, we will ask for resources on **Leonardo** to further **test the scalability to thousands of cores**.

Thank you for the attention