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

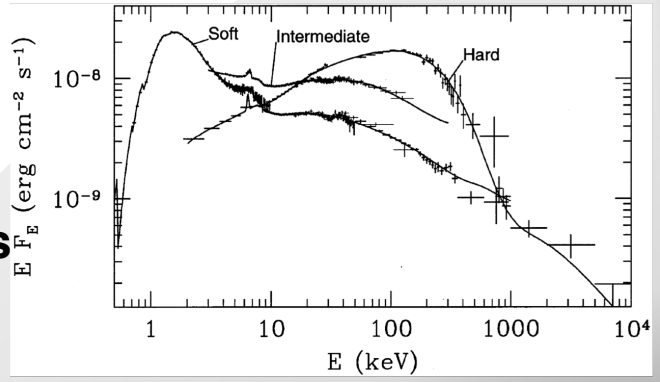
STINGRAY - Temporal analysis of large data archives from X-ray missions

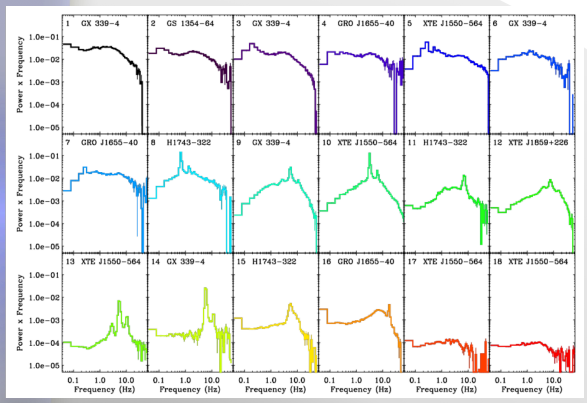
Eleonora Veronica Lai, Matteo Bachetti, Maura Pilia, +Daniela Huppenkothen and Stingray developers

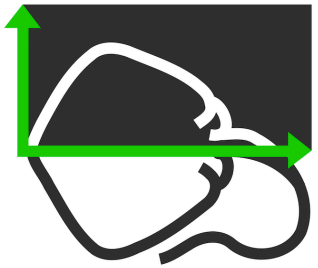
Monthly Meetings, WP 1-2 June 10, 2025

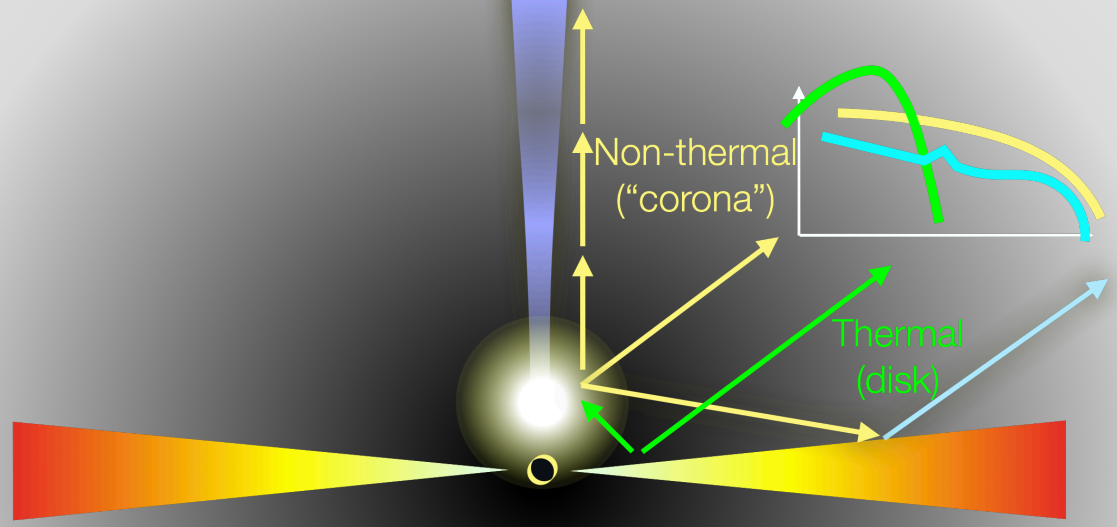
Scientific Rationale

- Some observe spectra, some observe variability. Is 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





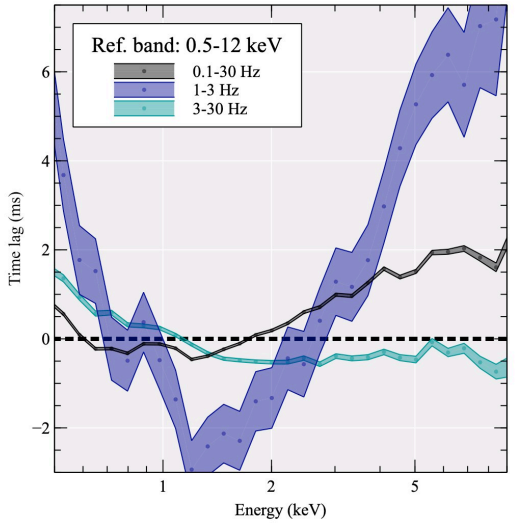


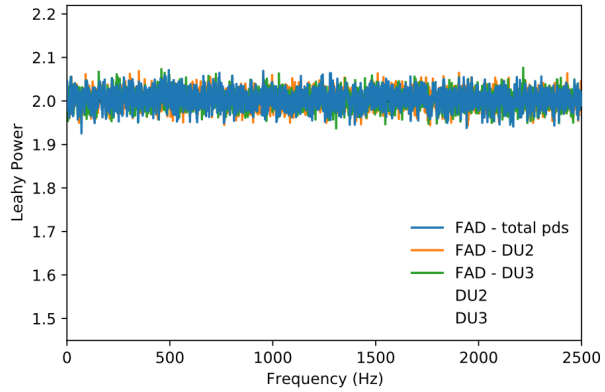


Huppenkothen et al. (2019) Bachetti et al. (2024)

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



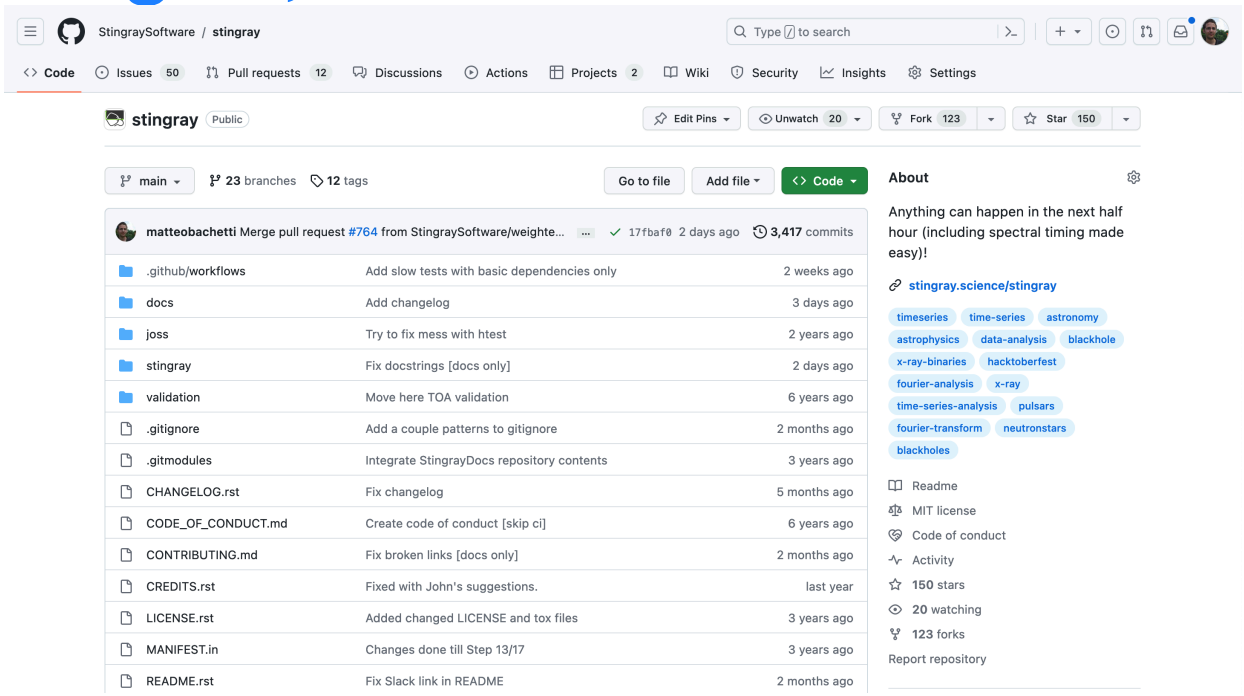


Technical Objectives, Methodologies, and Solutions: an open development model

- **Github-based** workflow:
  - Issue tracking
  - Assignments
  - Pull Requests

- Community outreach:
  - Public **Slack** channel
  - Talks
  - Hackatons/Tutorials
  - OpenAstronomy involvement
  - Astropy affiliated package

- Developers:
  - Astronomers
  - Google Summer of Code students



UNIAM
PRIN MIUR 2017

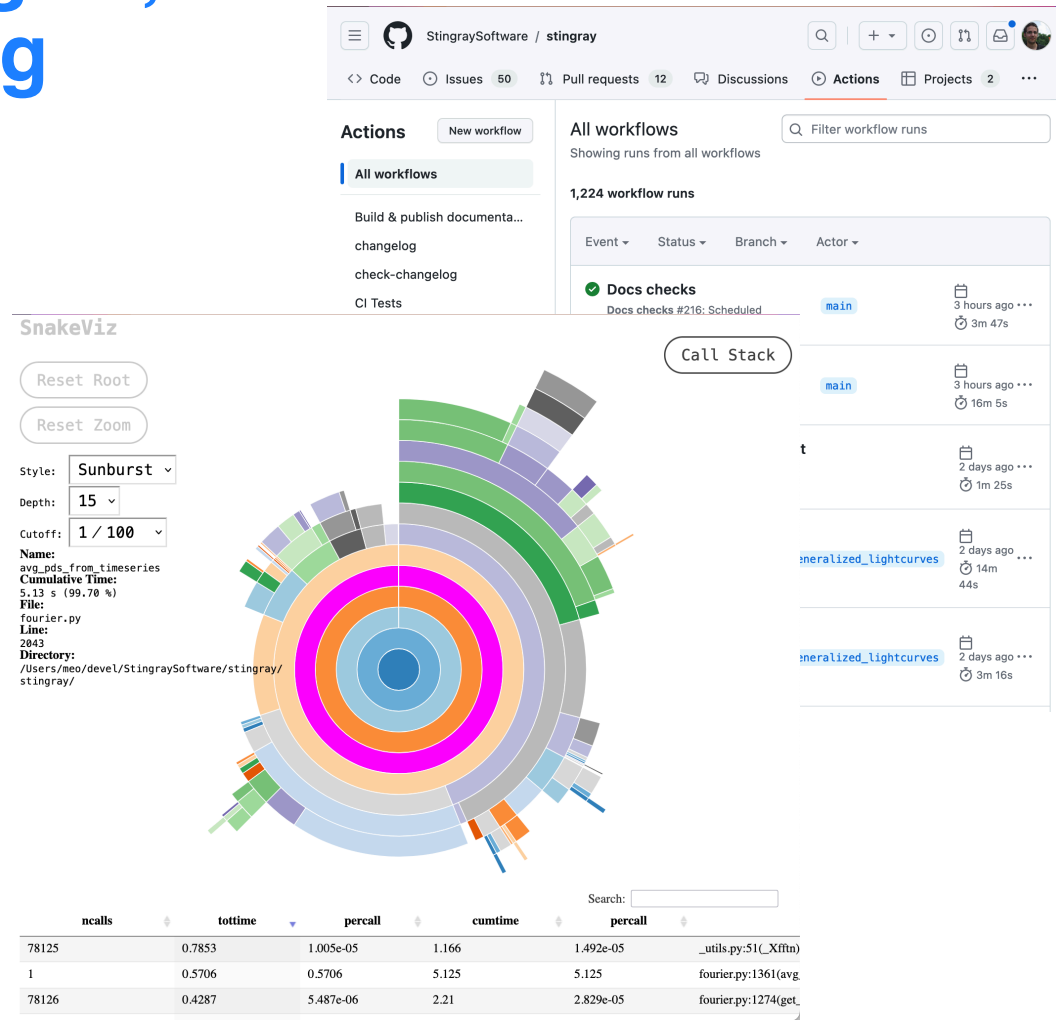
SpecTemPolar!
PRIN INAF 2019

INAF
ISTITUTO NAZIONALE DI ASTROFISICA

Google Summer of Code 2016, 2017, 2018, 2020, 2021, 2022, 2023, 2024, 2025

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



Main Results

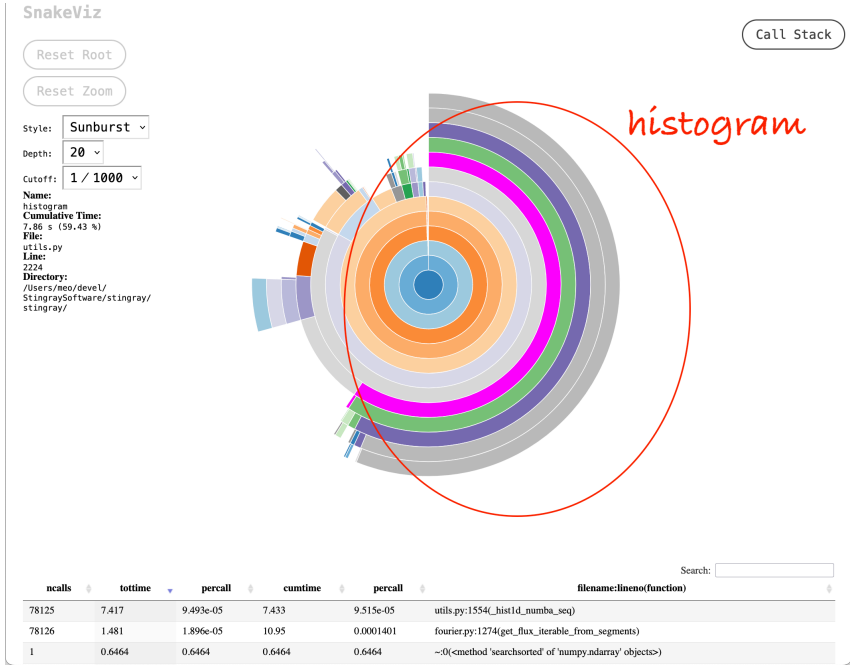
LARGE data sets (> 32GB)

1. Data loading

2. Parallelisation

SMALL data sets (< 32GB)

1. GPU porting



Main Results: LARGE data sets (> 32GB)

1. Data loading

Let's be "lazy": lazy loading with FITSTimeseriesReader

Now, let's try not to even pre-load the events. What will happen? First of all, we use the new class `FITSTimeseriesReader` to lazy-load the data, meaning that the data remain in the FITS file until we try to access them. This occupies very little memory.

[16]:
```
from stingray.io import FITSTimeseriesReader
%memit fitsreader = FITSTimeseriesReader(fname, data_kind="times")
```

```
peak memory: 2245.34 MiB, increment: 0.00 MiB
```

[17]:
```
from stingray.gti import time_intervals_from_gtis
start, stop = time_intervals_from_gtis(fitsreader.gti, segment_size)
%memit interval_times = np.array(list(zip(start, stop)))
```

```
peak memory: 2245.36 MiB, increment: 0.00 MiB
```

Let's create an iterable that uses the FITSTimeseriesReader to send AveragedPowerspectrum the pre-binned light curves for each segment. Events will be read in chunks from the FITS file, and streamed as light curve segments on the fly.

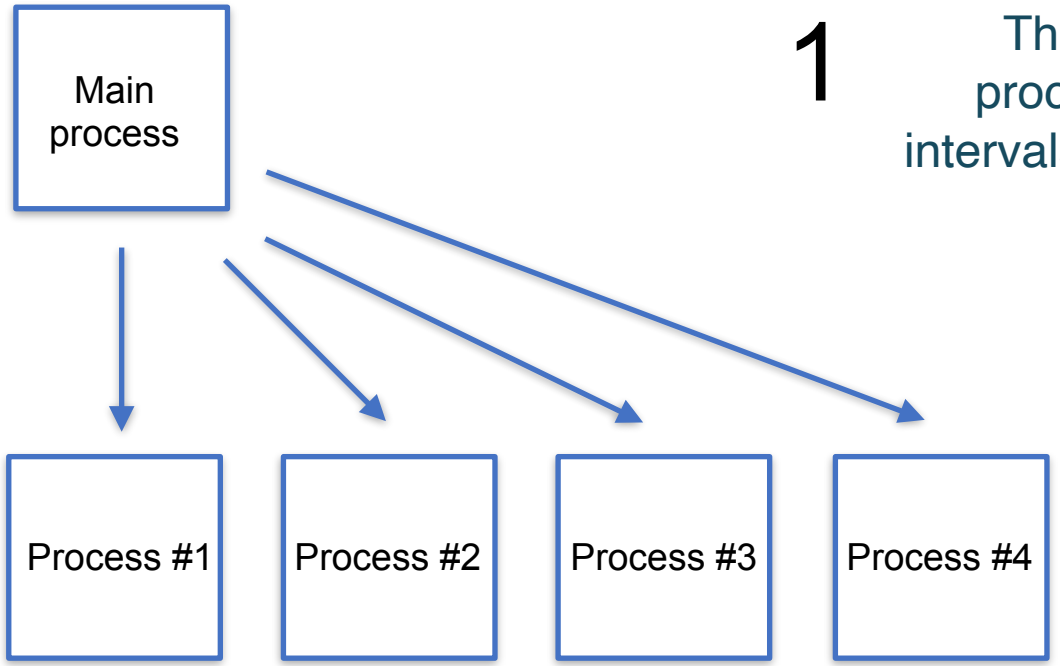
Main Results: LARGE data sets

2. Parallelisation

-Multiprocessing implementation

Main Results: LARGE data sets

- **Multiprocessing implementation**

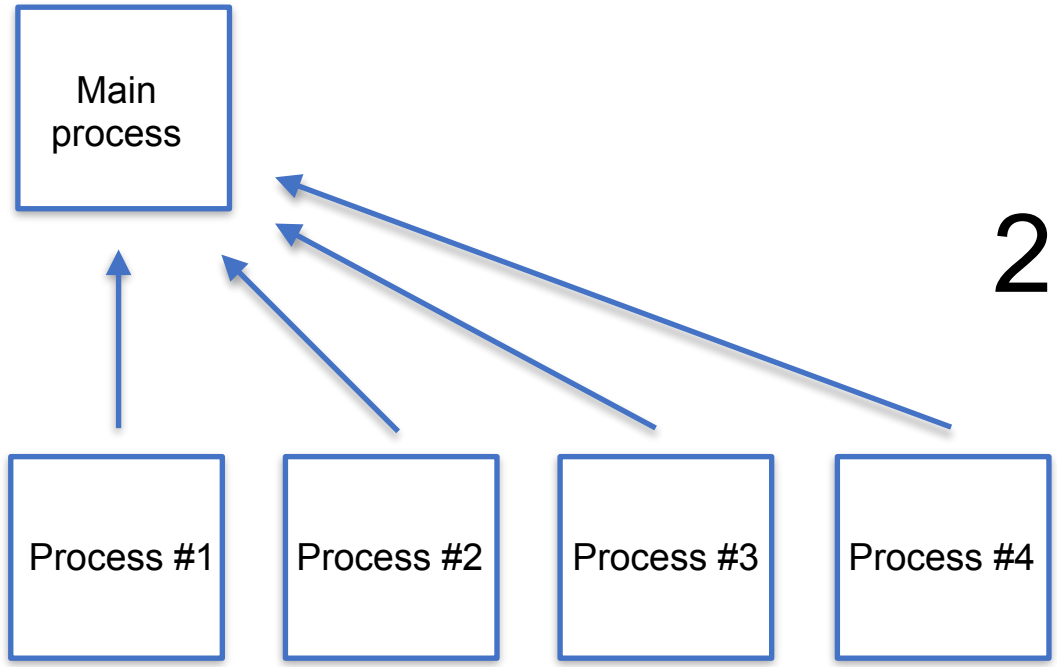


1 The main process spawns *M* processes and splits the *N* time intervals among them in an unordered way

= *M* t_spawn

Main Results: LARGE data sets

- **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: LARGE data sets

- 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

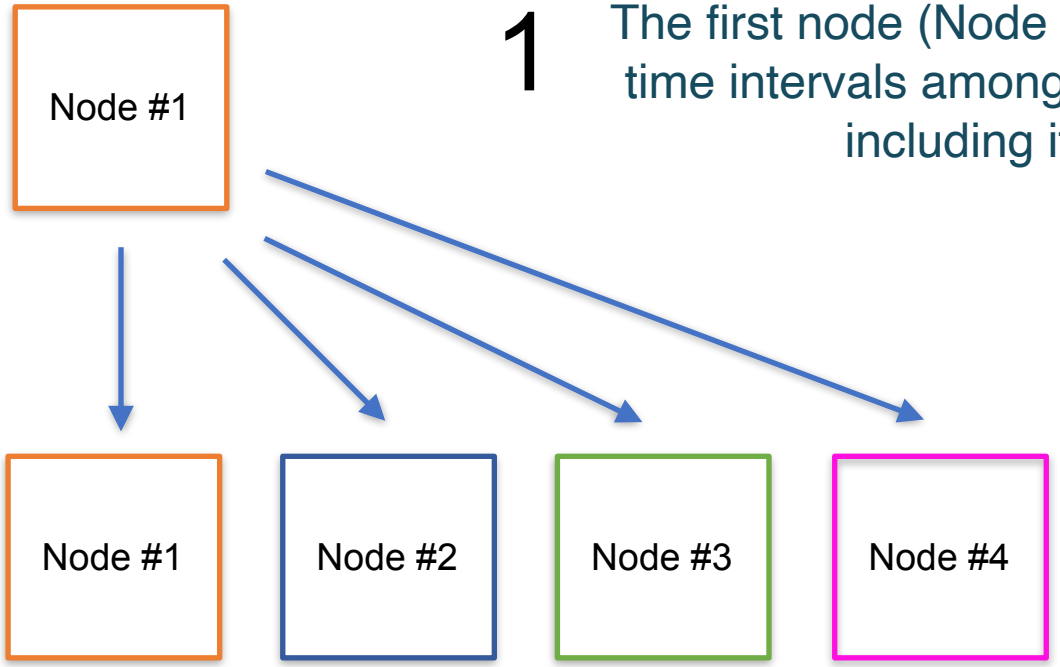
Main Results: LARGE data sets

2. Parallelisation

- MPI implementation

Main Results: LARGE data sets

- **MPI implementation**



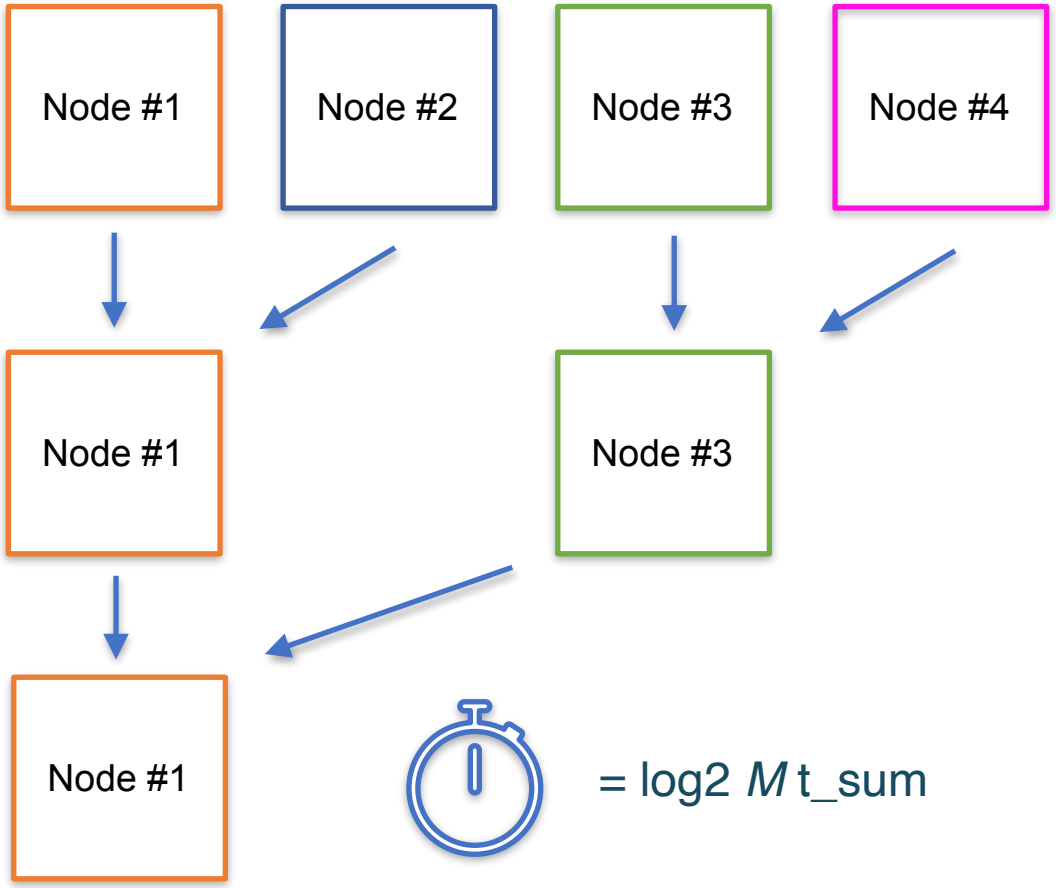
1 The first node (Node #1) splits the *N* time intervals among all the nodes, including itself

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

= *N*/*M* t_per

Main Results: LARGE data sets

- **MPI implementation**



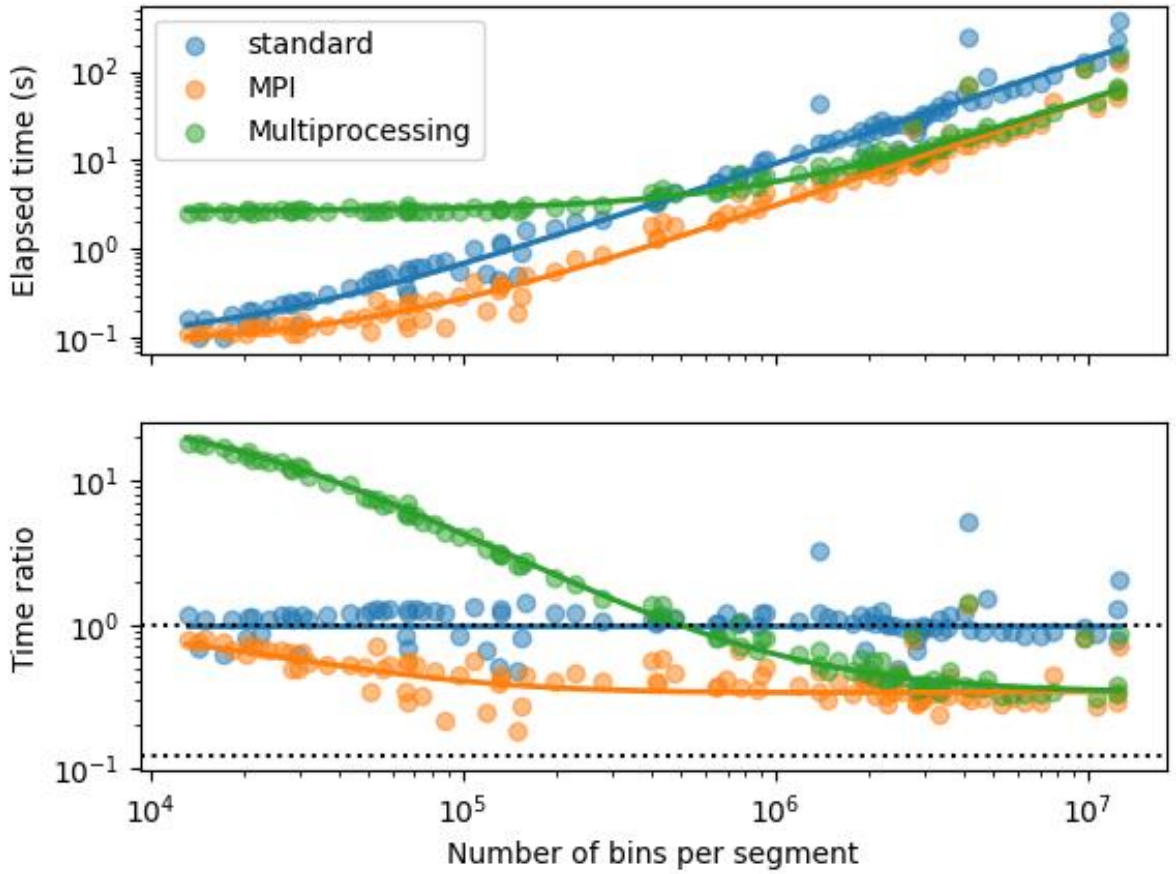
3

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

= log2 *M* t_sum

Main Results: LARGE data sets



MPI is *faster*
than the multiprocessing

Main Results: SMALL data set (< 32GB)

SnakeViz

Reset Root

Reset Zoom

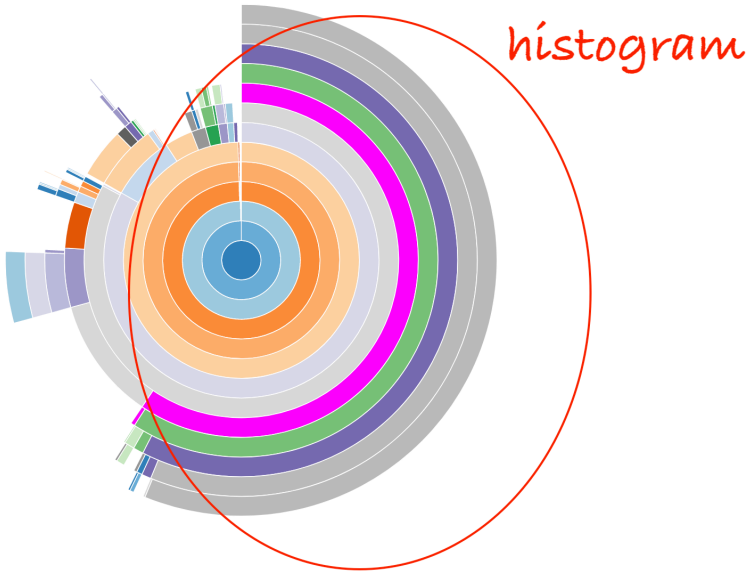
Call Stack

Style: Sunburst

Depth: 20

Cutoff: 1 / 1000

Name:
histogram
Cumulative Time:
7.86 s (59.43 %)
File:
utils.py
Line:
2224
Directory:
/Users/meo/devel/StingraySoftware/stingray/stingray/

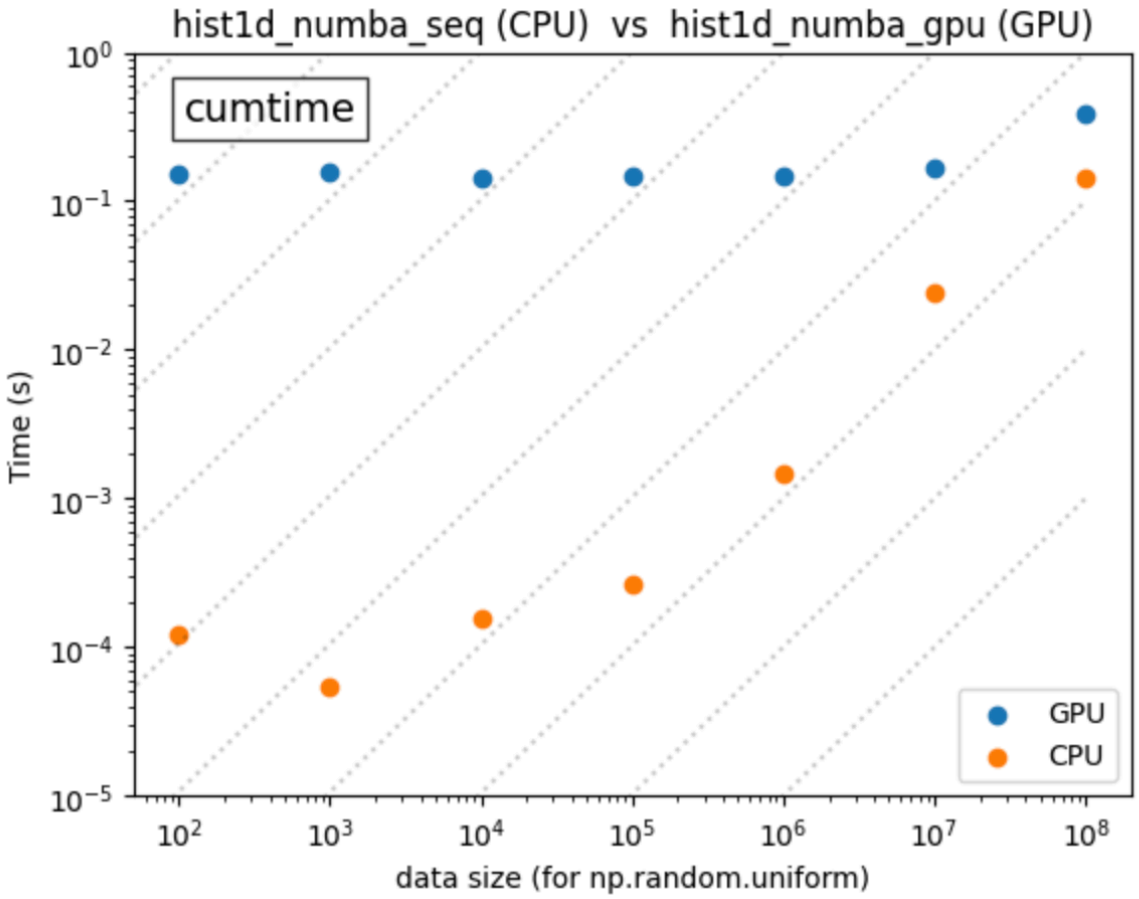


Search:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
78125	7.417	9.493e-05	7.433	9.515e-05	utils.py:1554(_hist1d_numba_seq)
78126	1.481	1.896e-05	10.95	0.0001401	fourier.py:1274(get_flux_iterable_from_segments)
1	0.6464	0.6464	0.6464	0.6464	~:0(<method 'searchsorted' of 'numpy.ndarray' objects>)

Porting in GPU?

Main Results: SMALL data set (< 32GB)

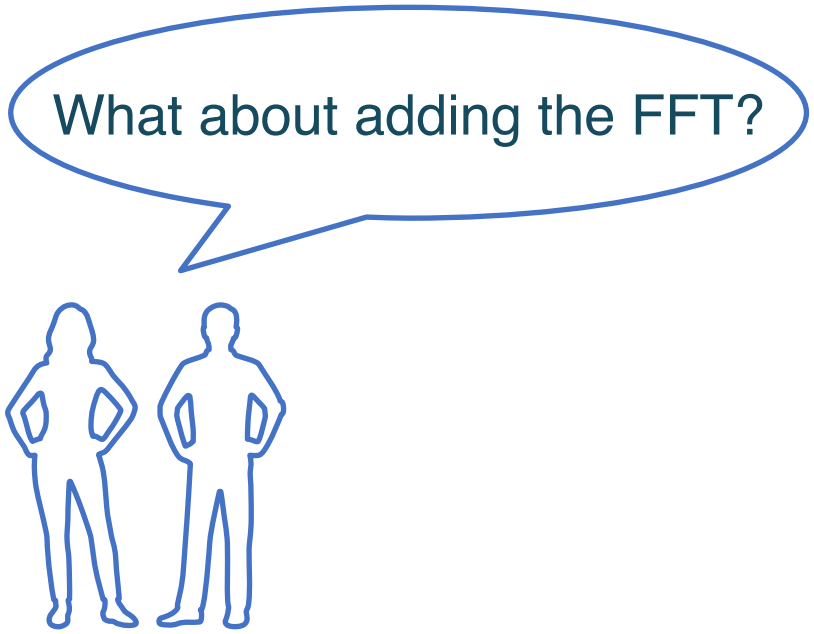


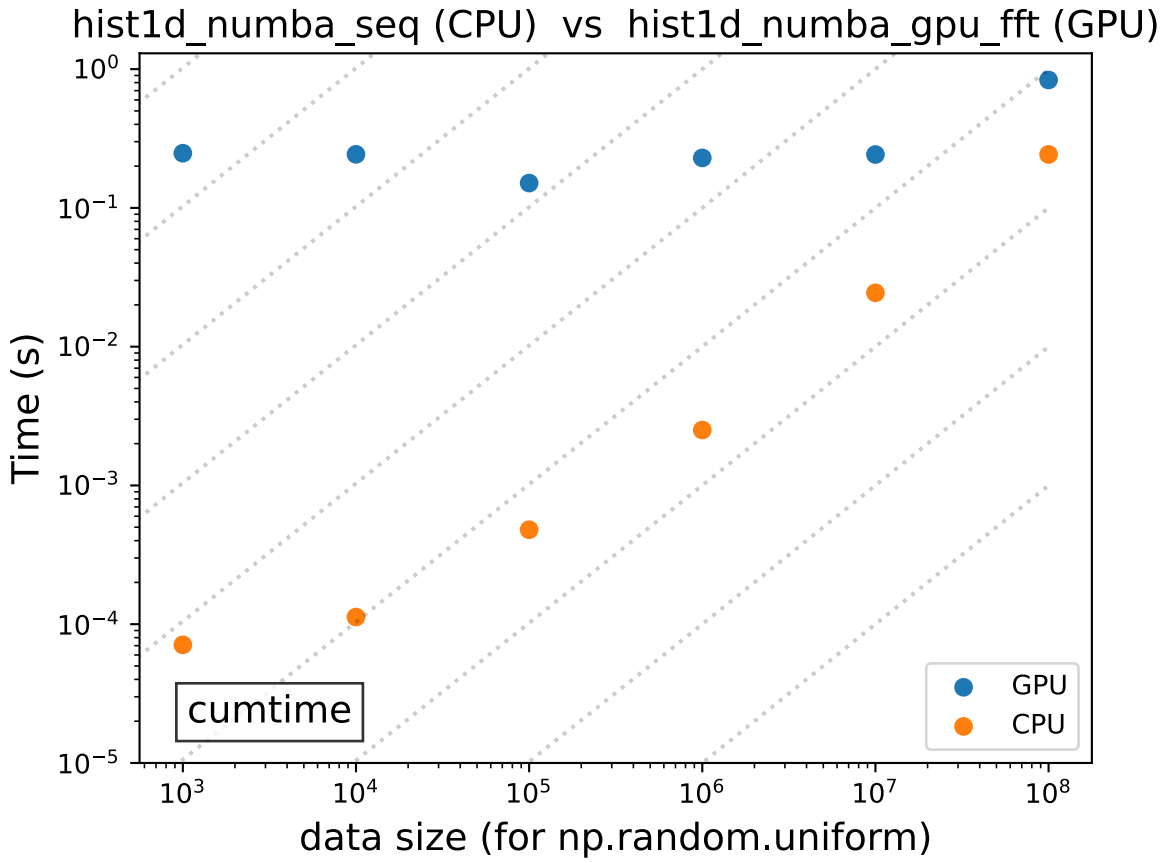
Porting in GPU?

Not convenient :(

Main Results: SMALL data set (< 32GB)

PDP 2025
33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2025)
12-14 March 2025, Torino (Italy)





Conclusions and final steps

LARGE data sets (> 32GB)

1. Data loading 

2. Parallelisation
   - Implementation and BM 
   - Include it in the codebase NEXT

Conclusions and final steps

LARGE data sets (> 32GB)

1. Data loading ✓
2. Parallelisation
   - Implementation and BM ✓
   - Include it in the codebase — NEXT

SMALL data sets (< 32GB)

1. GPU porting seems to be not suitable

Conclusions and final steps: future for Stingray

Matteo Bachetti, Fergus Baker and Kashish Shrivastav

GSoC 2025 with OpenAstronomy

Spectral timing in Julia

REQUIREMENTS
- Understanding Python code
- Julia knowledge

INITIATIVES
GSOC

COLLABORATING PROJECTS
stingray juliaAstro

TAGS
python Julia time series analysis

DIFFICULTY
medium

PROJECT SIZE
350 h

MENTORS
@matteobachetti @stefanocovino @fjebaker

Spectral timing in Julia

DESCRIPTION

The analysis of time series from astronomical observations in the X-rays is an excellent tool to test advanced physical theories. Of particular interest are the periodicities that are often observed in the X-ray signals coming from the surroundings of accreting black holes, and the evolution of the rotation period of neutron stars. The essential toolbox for X-ray timing analysis includes different kinds of periodograms, cross spectra, and a number of "variability vs energy spectra", that allow to understand the variability at different energies. This project is about the implementation of a basic set of X-ray timing analysis operations in Julia, continuing the porting of the core operations from the `stingray` Python package [initiated during Google Summer of Code 2022] (https://github.com/StingraySoftware/Stingray.jl)

MILESTONES

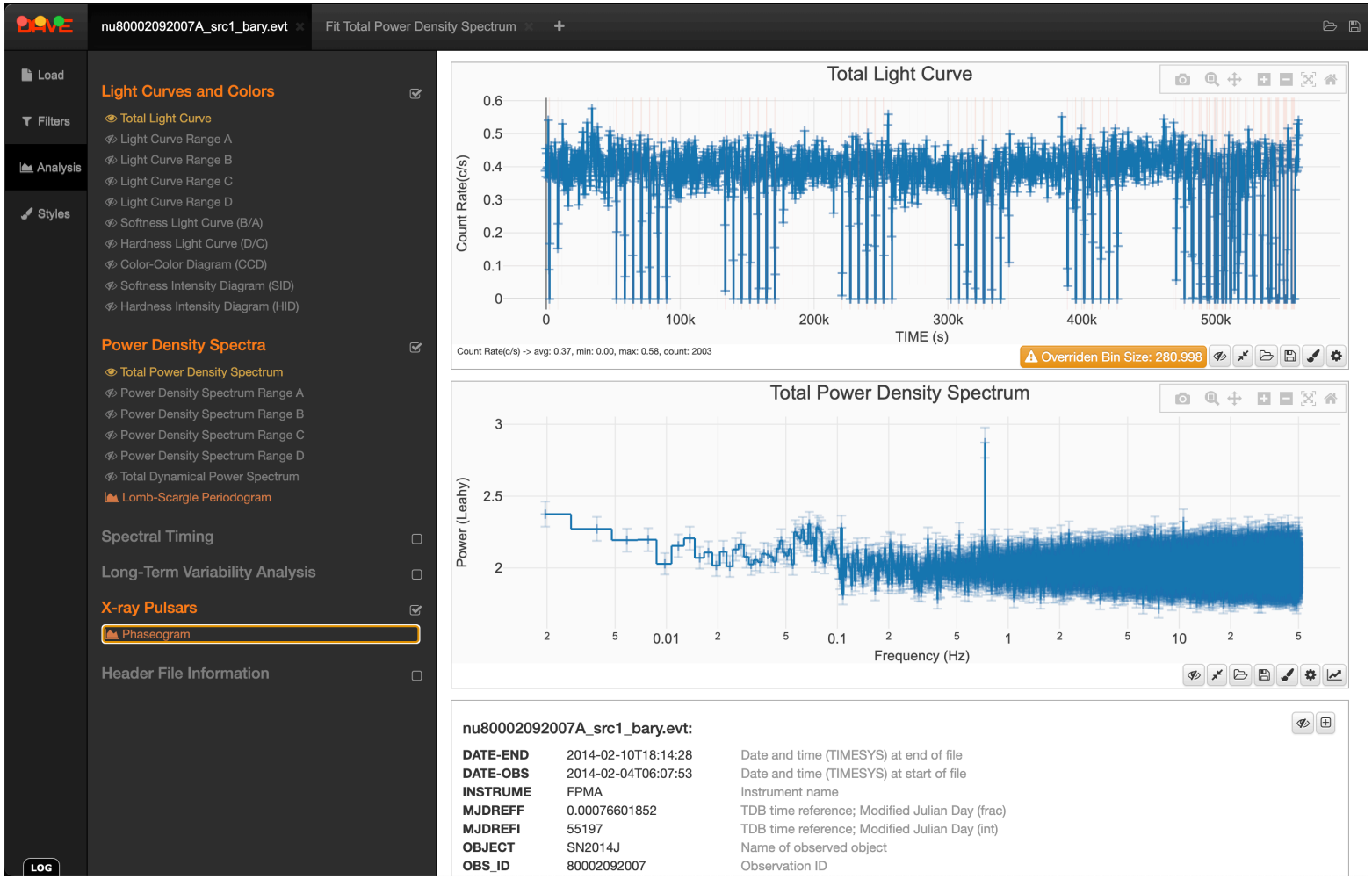
Coding starts
- Gain familiarity with the codebase
- Apply existing analysis to simulated datasets
- Implement I/O operation on FITS files

1st evaluation
- Implement a series of tests in Julia that the new code will have to pass
- Extend basic operations (periodograms and cross spectra) to event lists and light curves
- Time lags and coherence spectra

Very latest results: Dave resurrection!

DAVE (Data Analysis of Variable Events) is a GUI built on top of Stingray.

Its goal is to enable scientists to explore astronomical X-Ray observations and to analyse the data in a graphical environment.



Very latest results: Dave resurrection!

Modernization 2025 #79

New issue

Open **kartikmandar** wants to merge 17 commits into `StingraySoftware:master` from `kartikmandar:modernization-2025`

Conversation 0 · Commits 17 · Checks 0 · Files changed 180 · +46,322 −2,018

kartikmandar commented last week — Member

DAVE Modernization Project - Complete Technology Stack Upgrade

Overview

This pull request completes a comprehensive modernization of DAVE, updating the application from 2017 technology to 2025 standards. The project addresses 8 years of technical debt while maintaining complete scientific accuracy and improving performance significantly.

Key Accomplishments

Technology Stack Updates:

- Python 3.5.1 → 3.13
- Flask 0.10.1 → 3.1+
- NumPy 1.11.0 → 2.2+ (2-3x performance improvement)
- Electron 1.7.10 → 36.3.1 (35 major versions)
- Node.js 8.x → 22.16.0 LTS
- Plotly.js 1.30.1 → 2.35.2

Performance Improvements:

- FFT operations: 3.4x faster (34M+ points/second)
- Statistical calculations: 3x faster (107M+ points/second)
- Memory efficiency: 3x better (<0.1 KB per data point)

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
Successfully merging this pull request may close these issues.
None yet

2 participants

Google Summer of Code 2024

Kartik Mandar

&

Matteo Bachetti

https://github.com/StingraySoftware/dave/pull/79


Thank you!