How Can Profiling help us identify Ways to Optimise Science Modules?

Massinissa MACHTER / IJCLab - CNRS

Fink@World (8-12 June 2026)

What is Code Profiling ?

Measure execution time

- Profiling shows how much time is spent in each function or line of code

Find optimisation opportunities

- Profiling helps identify bottlenecks and areas for improvement

Profiling in Python

- Many scientific modules are developed in Python

Tool: **line_profiler** , simple and easy to use

- Installation: **pip install line-profiler**
- Add the **profiler decorator**
- Profile the script: **kernprof -l -v path/to/code_to_profile.py**

```
@profile
def my_function():
    ...
```

Example of Profiling in Python

```
Total time: 0.78459 s
File: /home/machter-l/fink-science-perf/profiling_examples/slow_example.py
Function: count_even at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def count_even(numbers):
     6         1          1.9      1.9      0.0      count = 0
     7   1000001     308531.1      0.3     39.3      for i in range(len(numbers)):
     8   1000000     319856.1      0.3     40.8          if numbers[i] % 2 == 0:
     9    500000     156198.0      0.3     19.9              count += 1
    10         1          2.4      2.4      0.0      return count
```

- **% Time** shows the percentage of total execution time spent on each operation
- **Hits** represents how many times each line of code is executed.

Profiling shows that this function spends most of its execution time iterating through the list

This leads us to replace the loop with a more efficient approach using sum (Python's built-in optimized iteration)

```
Total time: 0.199192 s
File: /home/machter-l/fink-science-perf/profiling_examples/quick_example.py
Function: count_even at line 4

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     4                                           @profile
     5                                           def count_even(numbers):
     6         1     199192.2 199192.2    100.0      return sum (1 for e in numbers if e % 2 ==0 )
```

Speedup ≈ 4×

Prerequisites for Profiling Fink Science Modules (1)

Docker: Installed on your machine

If you are not familiar with Docker: **https://www.docker.com/**

Docker is used to package an application together with all its dependencies into a container (image)

One essential Docker command that we will use later:

- **docker pull <image>**

This command downloads the image from a registry (Docker Hub, GitLab registry, etc.) where the image is hosted.

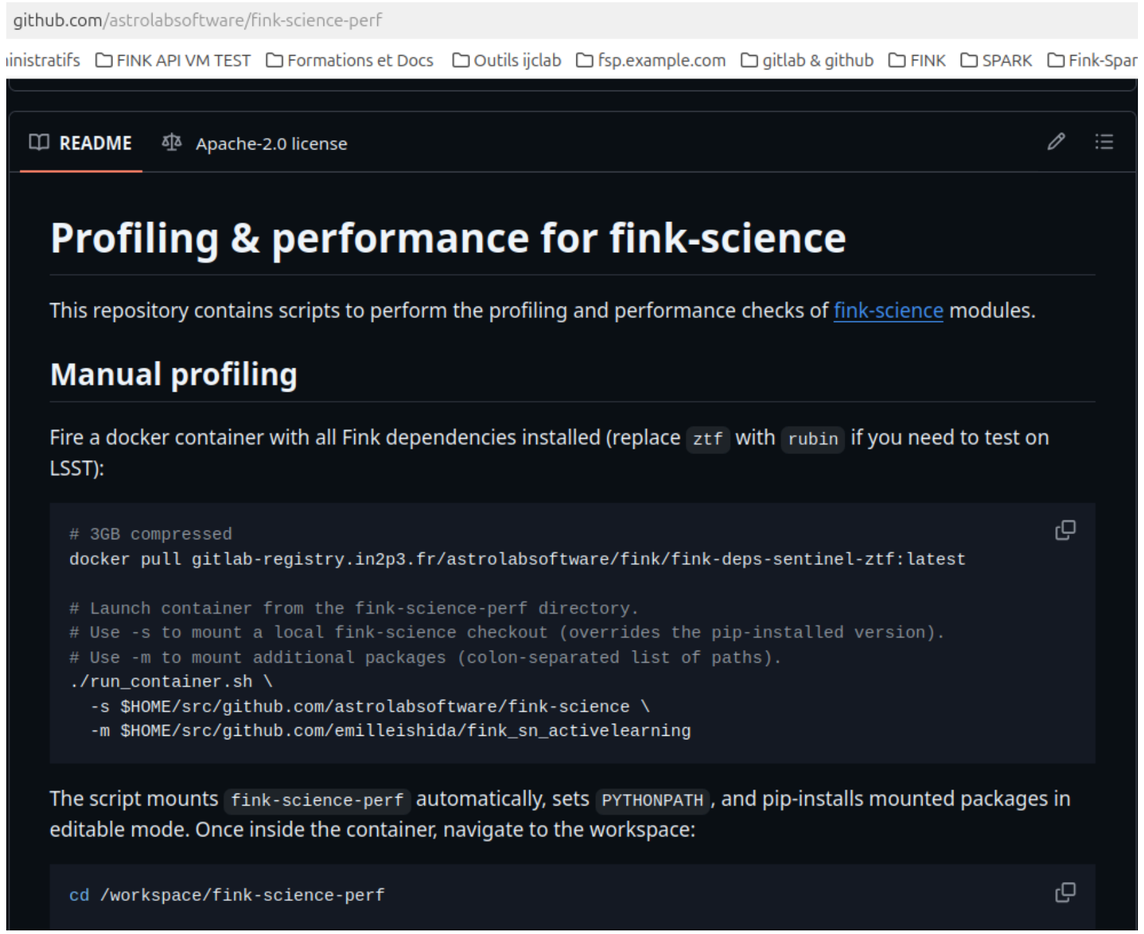
Prerequisites for Profiling Fink Science Modules (2)

Access to the **fink-science-perf** repository

Contains tools to profile Fink science modules, based on **line_profiler** described earlier.

Clone the repository in your local machine:

- git clone https://github.com/astrolabsoftware/fink-science-perf



Prerequisites for Profiling Fink Science Modules (3)

Fink Data Access as Input for Science Module Profiling

Before running the Fink data transfer, you first need to retrieve a topic by launching a data retrieval job.

For example, for ZTF data, go to: **https://ztf.fink-portal.org/download**

There, you can submit a data transfer job by selecting your query parameters.

After this, you can proceed as follows, Inside the **fink_science_perf** directory/ :


```
# Install fink-client
pip install fink-client

# Register with Fink broker (replace with your credentials)
fink_client_register \
    -survey ztf \
    -username <your_username> \
    -group_id <your_group_id> \
    -servers kafka-ztf.fink-broker.org:24499
```


```
# Download sample data
export TOPIC = <your_topic_name_data_transfert>
fink_datatransfer \
  -topic $TOPIC \
  -outdir $TOPIC \
  -partitionby finkclass \
  -survey ztf \
  --verbosew
```

... Final Preparation Step: Pull the Docker Image

Pull the image according to survey

This step only requires an internet connection and a bit of patience (may take ~ 3-4 minutes).

- image for **ZTF** dependencies

  $ docker pull gitlab-registry.in2p3.fr/astrolabsoftware/fink/fink-deps-sentinel-ztf:latest

- image for **Rubin/LSST** dependencies

  $ docker pull gitlab-registry.in2p3.fr/astrolabsoftware/fink/fink-deps-sentinel-rubin:latest

Let’s Profile a Fink Science Module

Live Demo

Installing Your Optimised Version of Fink-Science

Testing and profiling Your Optimised Version

After identifying and implementing optimisations, we need to test and validate them.

1. **Restart a container with your modified version of fink-science** (Specify your local version using the -s option):
   - $ run_container.sh -s /path/to/fink-science
2. Inside the container: **reinstall your package**
   - $ pip uninstall fink-science
   - $ cd /workspace/fink-science/
   - $ pip install .
3. **Run the module tests** (doctests must pass):
   - $ ./run_tests.sh -s <survey> --single_module path/to/processor.py
4. Go back and **re-run profiling**:
   - $ cd /workspace/fink-science-perf/
   - $ ./bench_module.sh -s ztf -n "ModuleName" -d <data_path> --profile

These steps help us ensure that our optimisations are correct, pass all tests, and improve performance before opening a pull request.

Conclusion

You can take inspiration from my ztf/Early SN Ia optimisation work :

- EarlySN classification optimisation : **https://github.com/emilleishida/fink_sn_activelearning/pull/30**
- Integration into Fink Science : **https://github.com/astrolabsoftware/fink-science/pull/676**

Thank You for Your Attention