Imperial College London

SHiP
Search for Hidden Particles

Implementation of muon shield optimisation

Oliver Lantwin
[oliver.lantwin@cern.ch]

11th SHiP Collaboration Meeting, CERN

8th June 2017

Introduction

- Give overview of implementation
- Touch on technologies used
- Outlook onto challenges and plans

Implementation: Summary

- Runs on SKYGRID at YANDEX
- Use FAIRSHIP for all simulations and geometry

olantwin/FairShip:optimisation_shield

- Replace the entire geometry:
  - Bare minimum geometry besides shield
  - Sensitive plane instead of trackers at z_{T1}

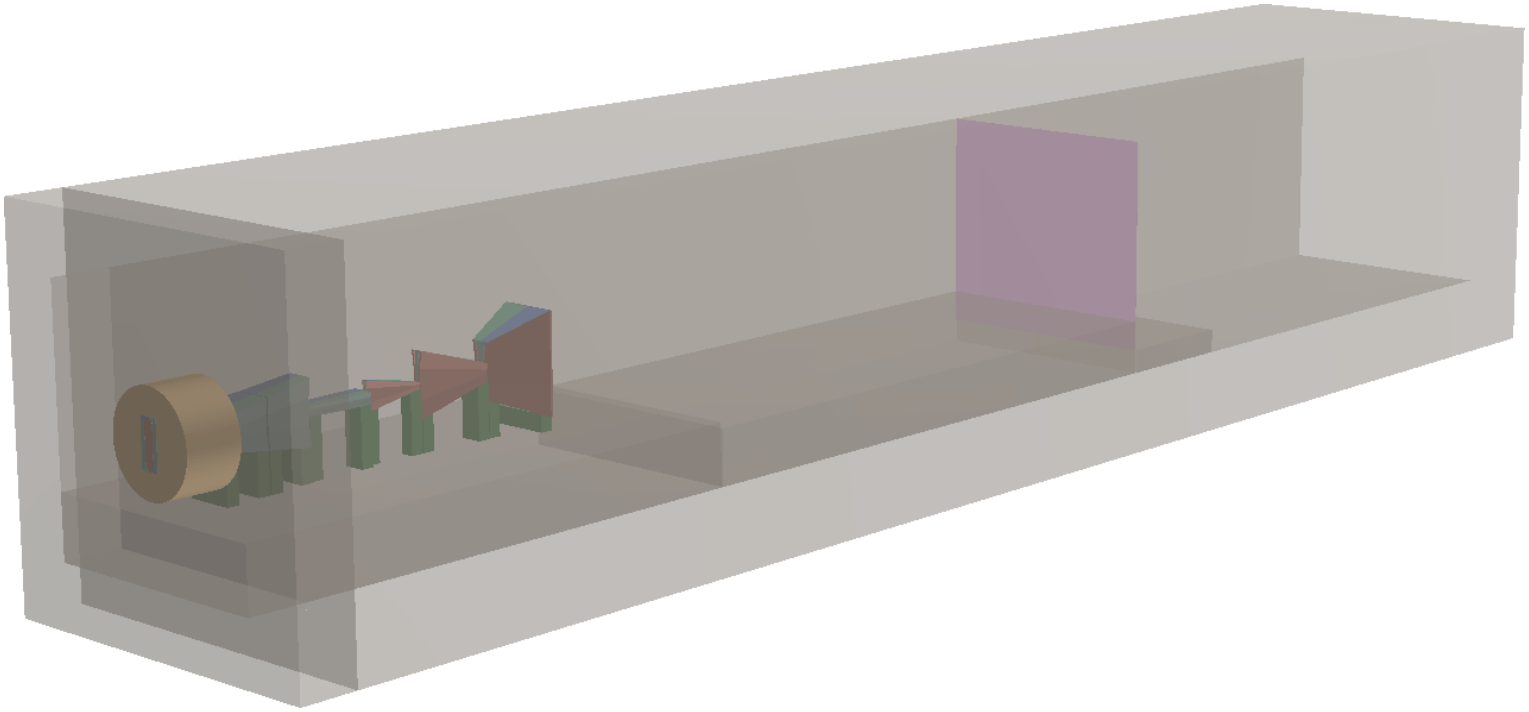

Figure: Geometry used

- Many changes to the way `ShipMuonShield` is created to make it easier to understand (for me at least), and to modify without breaking anything
  - Add logic to allow initialisation of `ShipMuonShield` from a `ROOT`-file

SKYGRID

- Cluster of 1600 CPU cores available for use by SHiP
- Jobs are run in Docker containers:
  - Idea: Job that takes a given input and reproducibly produces a given output in a functional way $\rightarrow$ easy to troubleshoot and test
  - Optimised for long-running jobs, e.g. large MC productions, as use of Docker results in non-negligible start-up time of jobs

SHiP Docker containers

- Container image based on CentOS 7
- Usable for general purpose FairShip work
- Great for reproducibly running studies
  - Can track exactly which software runs
- Special version with `olantwin/FairShip:optimisation_shield` branch maintained by me (`olantwin/ship-shield`)
- Use Docker container to ensure FairShip is set up correctly, no need to install FairShip on cluster
- We could use Docker for archiving analyses and testing software throughout SHiP

Optimisation algorithm used

`scikit-optimize`

- Bayesian optimisation library for the scientific python stack
- Considered both Gaussian-process-based and tree-based algorithms, as the latter can handle integral parameters. Return to Gaussian-process implementation for now.

Overall architecture

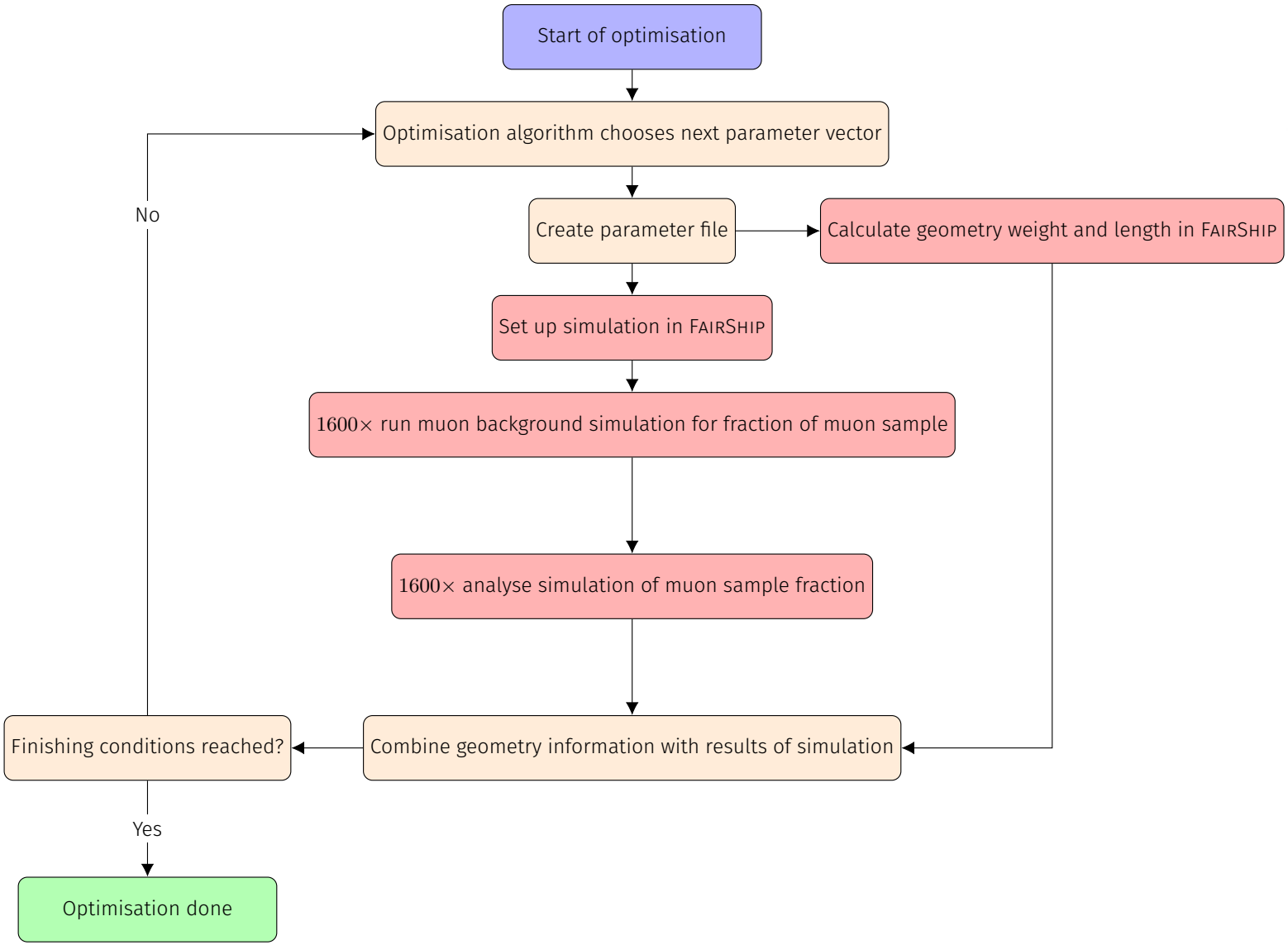

Figure: Flow chart of optimisation process; red blocks run in a short-lived Docker containers, one long-lived process for everything else

Preparation

- Split muon sample into input files for the jobs: $\frac{1}{1600} \times \#\mathrm{totalmuons}$ per job.
- Download input files from EOS and store locally on the cluster
  - Optimisation does not use EOS at all any more, as polling for files etc. proved to be unreliable
- Define physical bounds of magnet parameters (maybe too generous, as many physically allowed magnets are very large)

Translating the parameter vector to the geometry (and back)

How do 56 numbers turn into a geometry configuration?

- Create `ROOT`-file with a `TVectorD` of 56 `double`s and pass it to `shipDet_conf.py`, `geometry_config.py` and `ShipMuonShield.cxx`, which were all modified to parse information needed from this file.
- Use `pytest` to implement a closure test and algorithm to reconstruct this parameter vector from the geometry. Can initialise and reconstruct completely random geometries within bounds $\rightarrow$ within floating point arithmetic errors we achieve perfect closure
  - `pytest` a joy to use. We should use it more to prevent code breaking! `C++` equivalent?

Simulation

- Stripped down version of `run_simScript.py` in a self-contained function, as it is impossible to `import run_simScript` without side-effects
- Use `TGeant4` with the SHiP-default set of physics processes enabled
  - Investigated using more basic physics processes as well (in addition with a full simulation), but set aside for now
- Use `--FollowMuon` option in FairShip to
  - discard any non-muons, muons simulated with full fidelity
  - keep information about muon tracks in muon shield
- Annoyance/Problem: FairRoot uses some `C++` singletons (e.g. `FairRunSim`), which can not be created more than once per process and deleting/retrieving them does not seem to work in Python...
  $\rightarrow$ Workaround: Only one simulation per process, use subprocess/thread if necessary

Extracting geometry measurements

1. Initialise geometry in FairShip
2. Use TGeo to extract shape of muon shield, measure exact length as seen by simulation
3. Have TGeo calculate weight of muon shield (exclude supports)

Analysis

In the simulation process:

- Create histograms for offline checks of optimisation
- Select muons in acceptance (see next slide)
- Return sum of muon penalty weights (1 `float` per job)

Results passed back by the software managing the cluster

In the steering process:

- Combine geometry measurements and simulation results in cost function

Cost function

$$\propto \frac{W \times (1 + \sum \chi^2_\mu)}{A_{\mathrm{HNL}}}$$

$W :=$ Muon shield mass (from geometry)
$\chi^2_\mu :=$ $\chi^2$of muon hits in bending plane (from simulation)
$A_{\mathrm{HNL}} :=$
Signal acceptance (from geometry, approximated via length of magnet)

χ^2

$$\chi^2(x/\mathrm{cm}) = \sqrt{1 - (x + 300)/560}$$

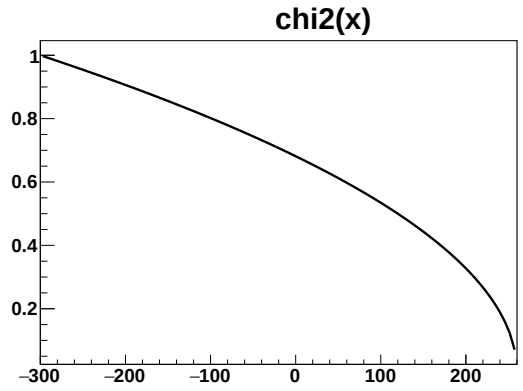

Figure: χ^2 as a function of x [cm] for μ^-

- for μ^+ flip sign of x as it's supposed to be on the other side of the magnet
- defined only for -300 < x < 260 cm (~dimensions of tracking station)
- Cut on $|y| < 5$ m

Challenges

Performance currently limited by constant costs:

- Need to initialise a new FairRoot simulation for every single simulation run (and also to initialise the geometry)
  - Partially because of the singleton issue mentioned above
  - Partially because updating the geometry in FairRoot is either complicated or impossible
- Starting up and retrieving results from Docker containers

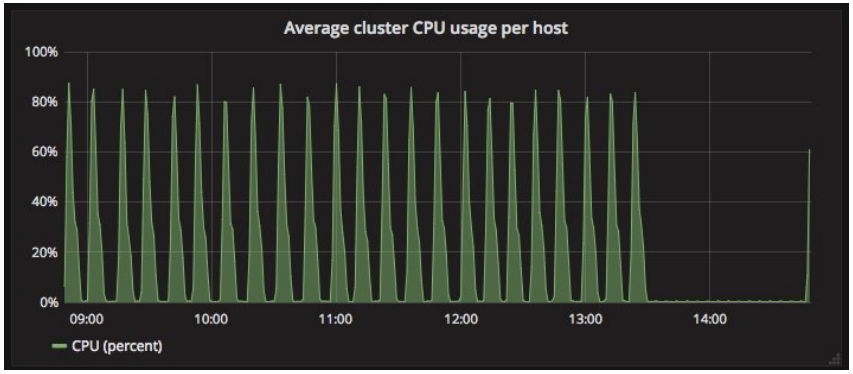

Figure: Cluster usage for optimisation iterations

Possible solutions:

- Try several configurations in parallel in order to make simulation time dominant? → needs to be supported by algorithm
- Reuse Docker containers (and ideally processes) for simulation to avoid constant costs as much as possible?

Planned changes

- Move to `scikit-optimize` ask-and-tell interface to allow for restart/replay, multiple simultaneous evaluations (maybe not possible)
- Once ready, move to Skoltech version of Bayesian Optimisation, which can
  - Predict multiple configurations
  - Handle different fidelity of observations (e.g. full statistics vs. resampled muons; full GEANT4 processes vs. subset; etc.)