Technical Report

Data reduction for CORSIKA

Dominik Baack

06/2016

technische universität dortmund

SFB 876 Verfügbarkeit von Information durch Analyse unter Ressourcenbeschränkung

Part of the work on this technical report has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project C3.

Speaker: Prof. Dr. Katharina Morik
Address: TU Dortmund University
Joseph-von-Fraunhofer-Str. 23
D-44227 Dortmund
Web: http://sfb876.tu-dortmund.de

1 Introduction

For the analysis of measured data by experiments, simulated Monte Carlo data is essential. It is used to test the understanding of the experiment, for separation of signal and background and for reconstruction of real physical properties from observable parameters. With increasing size of the experiments, more and more simulated data is needed. To optimize the simulation and to reduce the huge amount of calculation time needed, two different methods exist. The first method is the low-level optimization of the source code. The second one is the reduction of the actually needed Monte Carlo data. This report focuses on the cosmic ray simulation CORSIKA [1], which simulates cosmic ray induced particle showers within the atmosphere. In case of CORSIKA, big parts of the program are already optimized. Additionally, parts of the source code are only accessible in binary form so the first method of optimization is nearly impossible. Therefore the preferred method here is the reduction of unnecessarily generated data.

Simulations with CORSIKA

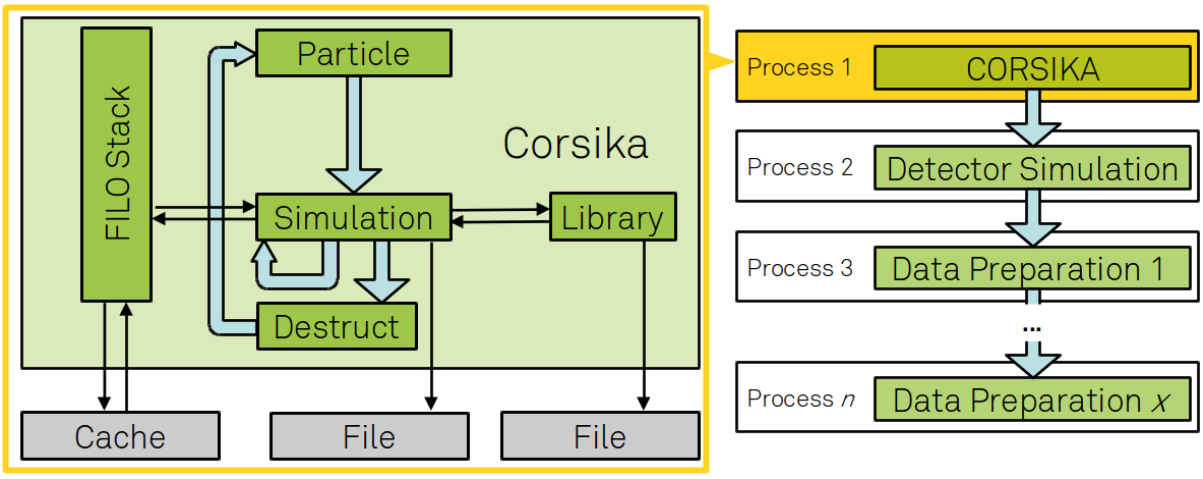

Figure 1: A typical simulation chain for an experiment using CORSIKA and the internal structure of the unmodified CORSIKA.

For many experiments using CORSIKA, the simulation is the first step in a complex and computing intensive simulation chain (Figure 1). The results of the simulation are the particles produced in the calculated air shower. These particles are used as input for the actual detector simulation. Depending on the detector those simulations can be arbitrarily complex. For a real analysis, many steps of data preparation, selection and processing follow.

The CORSIKA processing sequence starts with the sampling of a primary particle. This particle interacts with other particles in the atmosphere, which produce many additional ones. These new ones can also interact and produce additional particles, which results in a cascade of particles and interactions. This cascade fades out when the initial energy of the primary particle is consumed.

On the implementation side, CORSIKA starts with a randomly drawn particle from a user specified distribution. The primary particle is handed over to the code for the actual simulation of the physical interaction and propagation process, resulting in the generation of new particles. Those particles are written to a FILO ("First in - Last out") stack. In the next iteration, the last particle added to the stack is taken and goes through the same cycle. When a particle is below a specified energy, it is removed from the process. The whole simulation stops as soon as the stack is empty.

This report presents a modified and extended internal structure for CORSIKA, which is shown in Figure 2. The modifications can be divided in two modules: *Dynamic Stack* and *Remote Control*. Both have complementary approaches to reduce the amount of needed simulation cycles and provide an easy API for customizations without assuming any of the CORSIKA code or structure.

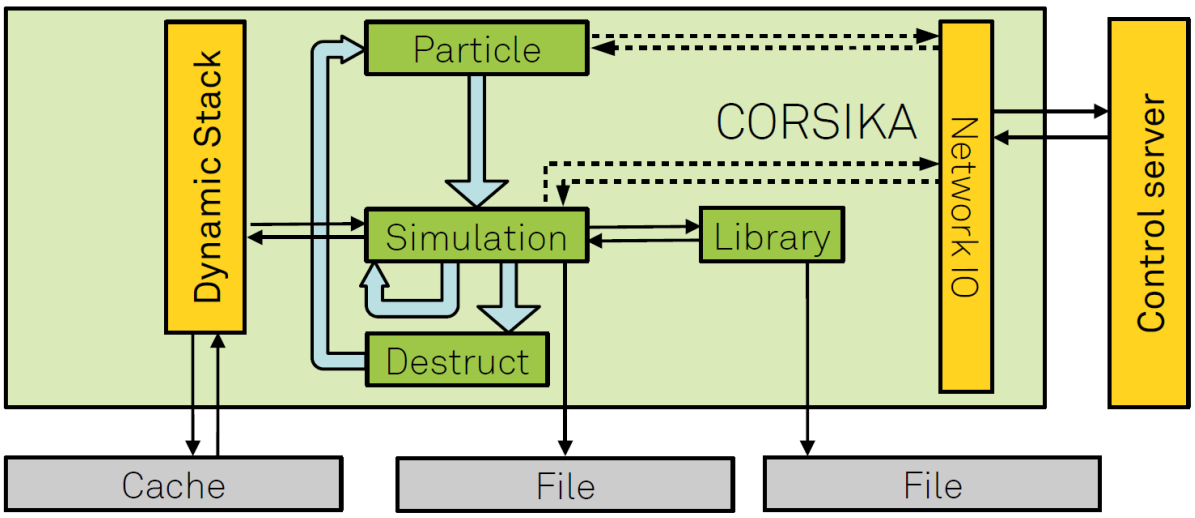

Figure 2: Schema of the internal structure of the modified CORSIKA. Every newly implemented element is highlighted in yellow.

2 Dynamic stack

The signature an air shower produces is highly dependent on the used detection mechanism. For different experiments different regions of energy, direction and particles types are measured. For example, an imaging air Cherenkov telescope like FACT [2] detects only photons in the field of view of the telescope (see Figure 3). The exact detection thresholds for the energy and direction depends on the orientation of the telescope and the trajectory of the shower. This example illustrates the reduction in computation time that can be achieved with a sophisticated selection of particles worth simulating.

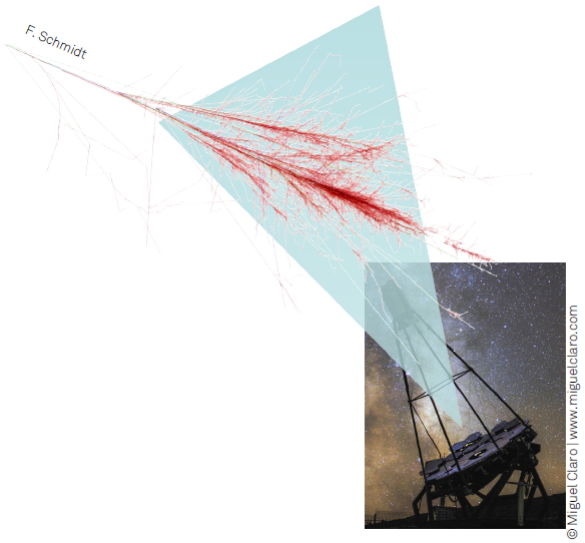

Figure 3: Schematic image of an imaging air Cherenkov telescope detecting a simulated air shower. The blue cone visualize the telescopes field of view.

Approach and Implementation

CORSIKA calculates every particle interaction within the atmosphere in a sequential order. Incoming particles are simulated and create, depending on their energy, type and possible interactions, a cluster of new particles. Every single one of them is independent and needs to be simulated separately. For this reason every produced particle with its type, energy, trajectory and additional information is stored on an intermediate stack. In the standard version of CORSIKA, the stack consists of an array used as "First in - Last out" queue and a hard disk cache that secures the stack from possible overflows. In this implementation, no complex or user defined filtering when adding or taking particles from the queue is intended.

The *Dynamic Stack* implementation is written in C++11/14 and replaces this old static storage entirely. Unlike the classic implementation, the stack is not fixed to a single implementation but easily customizable for every user. The implementation is based on the decorator design pattern with many preimplemented classes. A short overview of the most important, already implemented functionality is shown in Table 1.

To change the stack used in the simulation, only a single function needs modification. The function has access to command line arguments, so custom thresholds and settings can be set for every execution. Using combinations of the decorator classes, a complex decision making can be added to the stack in a transparent and efficient way. To replace the FILO stack with a FIFO stack, preceded by a filter, the code could look like this:

Storage	
FiFo Stack	Traditional “First in - First out” storage
FiLo Stack	Traditional “First in - Last out” storage
Null Stack	Discards every particle

Modifier	
Sort	Discrete sorting to specific stacks
Filter	Removal of particles that do not fullfil a user defined function
Callback	Calls a user defined function with particle information
Copy	Copy every particle into a second stack
Modify	Modifies every particle with a user defined function
Overflow to disk	Reduce the stack size and temporarily stores it on disc

Table 1: Implemented storages and decorators for the *Dynamic Stack* API.

```
1 auto setup(const char* const arguments = "")
2 {
3     auto stack = std::make_unique<
4             ::dynstack::wrapper::InputFilterStack<
5                 ::dynstack::storage::FIFO_Stack<Particle>,
6             decide>
7         >(100000);
8     return std::move(stack);
9 }
```

Each decorator makes heavily use of templates to reduce possible overhead. This leads to the same runtime for a stack with the same functionality as the original CORSIKA.

3 Remote Control

Even a sophisticated decision making on the level of produced particles doesn't prevent the simulation of showers which are rejected in later stages of the simulation toolchain. For example the complex trigger system of a detector can heavily favor or disfavor particles with specific physical characteristics. This often leads to an unevenly sampled parameter space. The approach pursued with *Remote Control*, is to have a control instance capable of seeding specific primary particles and cancel or prevent the simulation of the complete shower.

In Figure 4, the major parts of the FACT simulation chain and the number of events on those stages are shown. Only every thousandth event simulated survives the four steps of the simulation. For other experiments, the number of rejected events can be even many orders of magnitude higher. There are solutions to change the initial particle sampling to a not physically correct one that favors observable showers. But this leads to the necessity of additional error prone reweighting of the results.

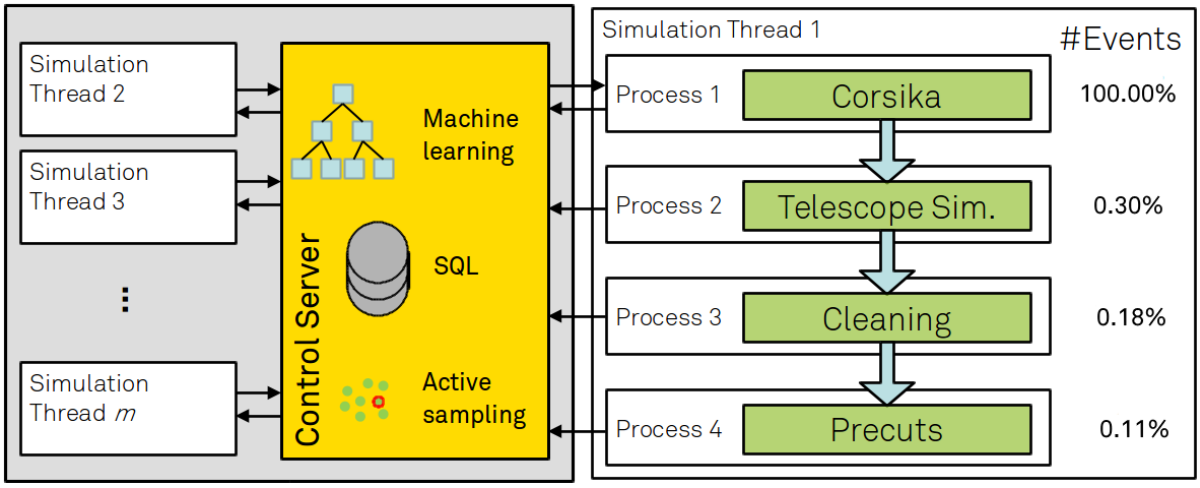

Figure 4: The figure shows a potential scheme of interactions of a control server with different simulation threads. The server can directly communicate with a running CORSIKA simulation using the *Remote Control* interface. The numbers shown for simulation thread 1 are the number of events on the different stages of the simulation. A physical analysis of the data takes place after process 4 and will implement additional selections of events and reduce the number of events even further.

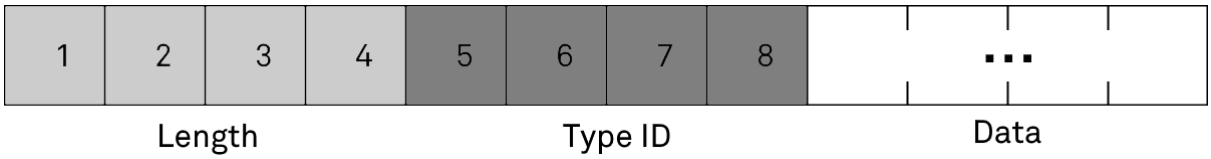

Figure 5: Scheme of the basic binary data package. The first four bytes are the length as unsigned integer (little endian). The next four bytes contain an id that maps to a specific data part.

Approach and Implementation

This second implemented API consists of an interface that allows communication with an external program. Different communication standards can be used with the main focus on TCP/IP Ethernet. For the communication itself, a slow XML-like human readable or faster binary protocol can be used. The network interface uses a threaded implementation to reduce, if not desired, any waiting for the main CORSIKA thread. Beside user initialized messages for information transmission, the remote control API is integrated into different parts of CORSIKA itself. This allows automatic information transmission to another program and receiving of information to use in the simulation itself. The information consists of normal event information (RUNHeader, EVTHeader, EVTEnd, RUNEnd, ...) on the CORSIKA sending side. The receiving part in CORSIKA can be used to replace basic functionality. It is possible to remove the local particle sampling and use the remote program to sample particles for the simulation, the same works for traditional startup parameters. Additionally the external program can stop the current event simulation or terminate CORSIKA altogether.

4 Application

Right now both implementations are used in a test setup with the FACT simulation chain. A filtering routine is implemented with *Dynamic Stack* in which every particle that is below a specific energy limit or direction gets removed. To ensure the complete shower will be detected by the telescope the particles are sorted to prioritize the simulation of high energy particles or/and particles directly pointing to the experiment. Subsequently, the *callback* decorator (see Table 1) is used to aggregate particle informations for an early rough estimation of the shower. This information is sent to a control server via the *Remote Control* API. It is possible but not feasible to send every particle to the control server, because this would lead to heavy strain on the network and the server receiving side. The external server uses the received information to estimate the telescope trigger probability. This probability is determined with a machine learning model trained on previous simulations of the full telescope.

Future Applications

The running test setup could be extended in different ways. For example the used trigger model could be trained online. The whole process would be initialized without a trigger model but the simulated events would be processed through the real trigger simulation. Those trigger informations could be used to train the model while the simulation is running. With a certain trigger model accuracy the control server could start to cancel showers in the already running setup. Another possibility is to use an active learning algorithm. For this purpose the signal-noise separation trained with the Monte Carlo data gives feedback about poorly sampled parameter regions. This feedback could be used by the control server to seed simulations in those regions. This should minimize the amount of very similar generated data and reduce the time needed to train the model.

Another way to utilize the *Remote Control* API is to implement an infrastructure to run CORSIKA simulations on very heterogeneous, distributed systems. A possible scenario is to set up a control server controlling simulations on multiple threads and machines (see Figure 4). A starting CORSIKA thread would register at the control server and request simulation tasks and startup parameters. A cancelled thread could notify the control server and push the simulated showers to the server. In such a setup it would not matter if the CORSIKA thread is running on a node of a big computing cluster, an unused workstation or any other kind of unused computing resource.

5 Conclusion

The *Dynamic Stack* implementation allows arbitrarily complex filtering and sorting routines to reduce the amount of simulated particles in a shower. It can also modify individual particles to increase the relevance of the simulated showers for the individual needs of the user. With the *callback* decorator from *Dynamic Stack* it is possible to obtain information about a running simulation of a shower. This information can be transferred

with the *Remote Control* API to an external control server. The server can run any kind of decision making and incorporate all available information to make decisions whether to cancel or to modify the running simulation. In combination the *Remote Control* and *Dynamic Stack* APIs offer many ways to utilize the computation time as well as possible and adjust the simulation for individual needs. The *Remote Control* API can also be used to set up complex infrastructures to maximize the use of available computing resources.

Both APIs can be used easily without any knowledge about the CORSIKA code or internal structure. This should give the users the ability to implement large scale production setups and small scale individual simulations in a very efficient manner.

With the plans to build bigger and more complex detectors for astrophysical measurements like CTA [3] and IceCube Gen 2[4], the need for air shower simulations will increase rapidly. Right now CORSIKA is the most widely used simulation in this area. Therefore it can be expected that the needed amount of CORSIKA simulations will increase in a similar order of magnitude as the growth of the detectors. A better utilization of the computing time and resources will be vital to satisfy this need.

References

[1] Heck, D. *et al.*. CORSIKA: A Monte Carlo code to simulate extensive air showers, No. FZKA-6019, 1998

[2] Anderhub, H., *et al.*; Design and operation of FACT - the first G-APD Cherenkov telescope., Journal of Instrumentation 8.06 (2013): P06008.

[3] Actis, M., et al. "Design concepts for the Cherenkov Telescope Array CTA: an advanced facility for ground-based high-energy gamma-ray astronomy." Experimental Astronomy 32.3 (2011): 193-316.

[4] Aartsen, M. G., et al. IceCube-Gen2: a vision for the future of neutrino astronomy in Antarctica. arXiv preprint arXiv:1412.5106 (2014).