Towards the next generation of CORSIKA: A framework for the simulation of particle cascades in astroparticle physics

Ralph Engel[1], Dieter Heck[1], Tim Huege[1], Tanguy Pierog[1], Maximilian Reininghaus[1], Felix Riehn[2], Ralf Ulrich[1], Michael Unger[1], and Darko Veberič[1]

[1]Institute for Nuclear Physics, Karlsruhe Institute of Technology, Germany
[2]Laboratório de Instrumentação e Física Experimental de Partículas, Lisboa, Portugal

June 2018

Abstract

A large scientific community depends on precise modelling of complex particle-cascading processes in various types of matter. The most obviously related fields are cosmic-ray physics, astrophysical-neutrino physics, and gamma-ray astronomy. In this white paper we summarize the steps needed to ensure the evolution and availability of optimal simulation tools in the future. The purpose of this document is not to act as a strict outline of the software, but merely to provide guidance for the vital aspects of its design. The main topics considered here are driven by physics and scientific applications, furthermore, the main consequences on implementation and performance are given an outline. We highlight the computational performance as an important aspect guiding the design since future science applications will heavily depend on an efficient use of computational resources.

1 Introduction, History, and Context

Simulations of air showers are an essential instrument for successful analysis of cosmic-ray data. The air-shower simulation program CORSIKA [1] is the leading tool for the research in this field. It has found use in many applications, from calculating inclusive particle fluxes to simulating ultra-high energy extensive air showers, and has been in the last decades employed by most of the experiments (see e.g. [2] and references therein). It has supported and helped shape the research during the last 25 years with great success. Originally designed as a FORTRAN 77 program and as a part of the detector simulation for the KASCADE experiment (the name itself comes from "COsmic Ray SImulations for KAscade"), it was soon adapted by other collaborations to their uses. The first were the MACRO [3] and HEGRA [4] experiments in 1993. As a consequence, over the time it has evolved enormously and is nowadays used by essentially all cosmic-ray, gamma-ray, and neutrino astronomy experiments. Furthermore, it helped to create a universal common reference for the worldwide interpretation and comparison of cosmic-ray air-shower data. Before CORSIKA, it was very difficult to assess the physics content of the data, and almost impossible to qualify the compatibility of different measurements. In general the simulation of extensive air showers was recognized as a fundamental requirement for astroparticle physics [5], and other tools have also been proposed for this purpose, of which the most well known are MOCCA [6], AIRES [7], and SENECA [8].

In general, the simulation of extensive air showers was recognized as one of the fundamental prerequisites for succesful research in astroparticle physics [5]. In the past, some other tools have also been developed for these purposes, of which the most well known are MOCCA [6], AIRES [7], and SENECA [8].

Over all the years CORSIKA evolved into a large and hard to maintain example of highly complex software, mostly due to the language features and restrictions inherent to FORTRAN 77. While the performance is still excellent and the mainstream use-cases are frequently tested as well as verified, it is increasingly difficult to keep the development up-to-date with requests and requirements. It is becoming obvious that the limited features of the FORTRAN language and the evident complexity of the new developments are getting into a conflict. Furthermore, in the future, the expertise needed to maintain such a large FORTRAN codebase will be more-and-more difficult to provide. Therefore, it is important to make CORSIKA competitive for the challenges we are facing in the future, requiring us to make a major step in terms of used software technology. This will ensure that CORSIKA will evolve further and become the most comprehensive and useful tool for simulating extensive particle cascades in all required environments.

2 Purpose and Aim

The purpose of CORSIKA is to perform a "particle transport with stochastic and continuous processes". A *next-generation CORSIKA* (ngC) will implement this core task in the most direct, flexible, and efficient way. In this document we will refer to this project as ngC, but just as a simplification and to clearly distinguish it from the existing CORSIKA program. The ngC will provide a framework where users can implement plugins and extensions for an unspecified number of scientific problems to come. CORSIKA will take a step from being an air-shower simulation program, to becoming the most versatile framework for particle-cascade simulations available.

The ngC must support particle tracking, cascade equations (CE), thinning, various particle interaction models, output options, (massively) parallel computations including GPU support, various possibilities for user routines to interact with the simulation process, and full exposure of particles while they are tracked/simulated. Furthermore, production of Cherenkov photons, radio signals, and similar non-cascade extensions should be fully supported. As usual, the cascades could be simulated in the atmosphere, but options for other media or a combination of them will be added.

In the future we expect that millions, if not billions, of CPU hours of high performance computing will be spent on air-shower simulations for experiments like CTA [9], IceCube [10], HESS [11], MAGIC [12], the Pierre Auger Observatory [13], the Telescope Array [14], LOFAR [15], and other next-generation experiments. It is up to ngC to make sure this is done as efficiently and accurately as possible, while maximising the resulting physics output. In this respect, ngC plays an important role in spending valuable and sparse resources while it is at the same time a fundamental cornerstone supporting the physics output of many large experiments.

3 Main design considerations

Some of the goals to achieve with ngC are extensibility, flexbility, modularity, scalability, and efficiency. The main outline of the steps of a typical particle transport code with processes is illustrated in Fig. 1. The central loop involves a stack used for temporary storage of particles, a geometric transport code, and a list of processes that can lead to secondary-particle production or absorption.

3.1 Overcoming limitations of current CORSIKA

The current CORSIKA implementation has a number of limitations, originating mostly from optimization to specific use-cases, most of which should be eliminated in ngC. These limitations and our anticipated improvements include the following items:

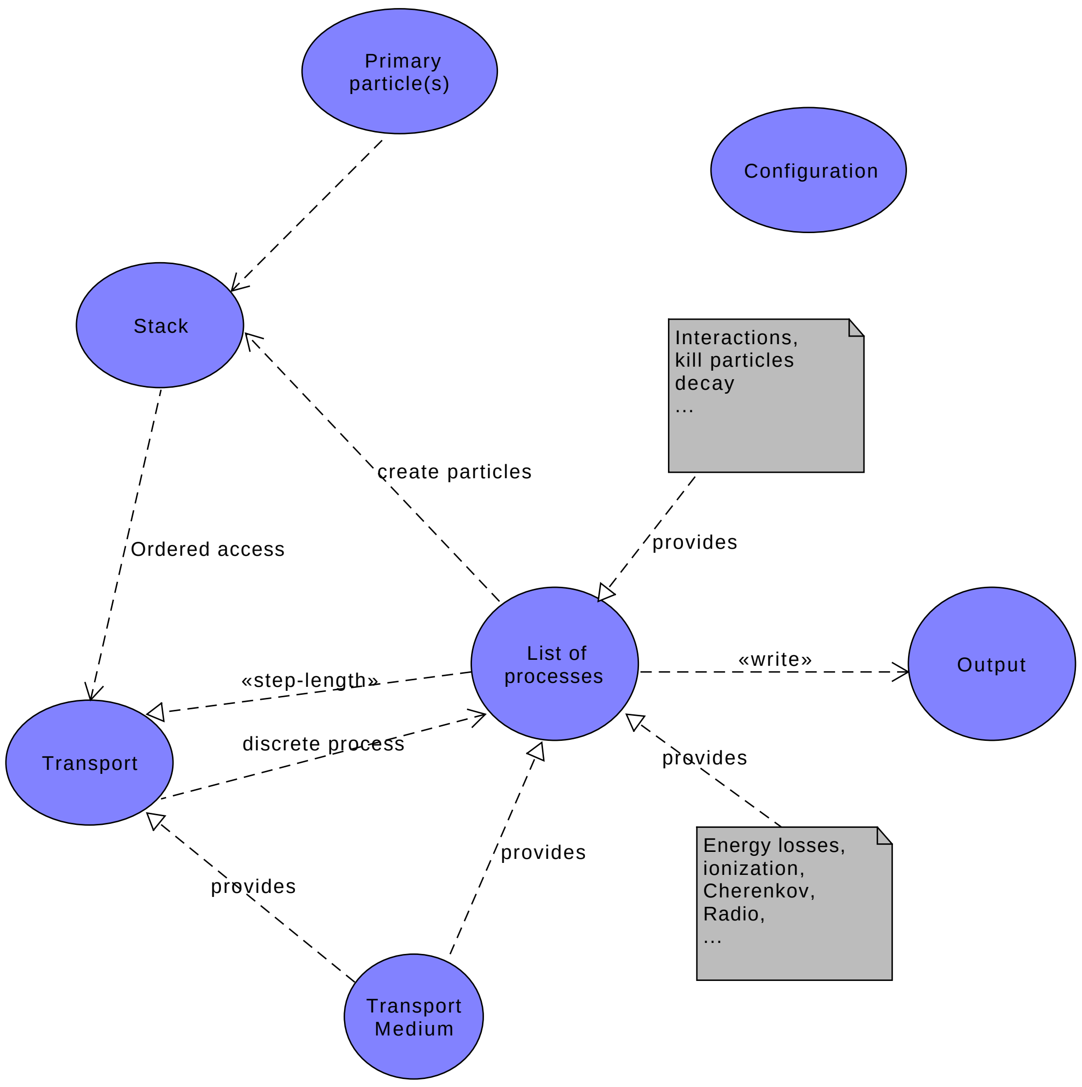

Figure 1: Scheme of particle transport code with processes.

1. The interaction medium is air. Its density is analytically described by five piecewise-exponential layers. NGC should support arbitrary media (air, ice, lunar regolith, rock salt etc.) and also transitions between them. In addition to density, the medium should provide refractive index, humidity, temperature, and possibly other information. Medium properties might also need to be fed back to the cascade simulation, e.g. by influencing various energy cutoffs.

2. In CORSIKA, processes taking the cascade simulation as input, e.g. radio or Cherenkov-light calculations, cannot feed back information to the cascade simulation. In NGC, any process should be able to give a useful feed-back, e.g. by requiring a change of simulation step-size.

3. In NGC, an interface should be provided for easy addition of new interaction models which treat particles or energy ranges not covered by any other interaction model.

4. CORSIKA does not allow for particle oscillations (e.g. neutrinos, K^0_L/K^0_S). A discussion whether or not oscillations should be incorporated in NGC should be started.

5. Support for inspecting and storing the history of ground-reaching particles (with the EHISTORY option [16]) is very limited due to rigid memory layout.

6. No upward-going Cherenkov photons can be handled.

3.2 Related projects and previous work

ngC will heavily depend on expertise gained with the original CORSIKA program. In addition, experience gained in other projects will be taken into account:

- MCeq [17] is a recent tool dedicated to the numerical solution of the CE. It already offers GPU support and very high computational efficiency. CONEX [18] is a CE air-shower simulation program that has been integrated in CORSIKA and provides enormous increase in computation speed.
- dynstack [19] is a recent extension of CORSIKA. Its basic functionality should be adopted for the stack of ngC.
- COAST [20] has been developed for CORSIKA with the aim of offering scientists a plugin-like extensibility. The fundamental functionality of COAST will be available in ngC.
- Offline [21], the offline analysis framework of the Pierre Auger Observatory, offers a versatile interface to and implementation of concepts related to geometry and coordinate systems.
- Other programs also combine tracking and physics processes, but with emphasis on different aspects than what is needed in air-shower simulations; examples are GEANT4 [22] and CRPropa [23].

3.3 Output

Physics processes of any type can produce various pieces of output information. These can be particle lists, profiles, histograms, text, etc. Therefore, when this output is written to a disk, it has to be stored in a similarly structured file format, since we require that all the relevant output ends up in one single file only. However, output can also be written to other places than disk, for example directly as input for subsequent workflow steps, network sockets, or other programs – but this is subject to requirements from the user community. The user should be able to decide what kind of output is optimal for his specific case.

The old binary output format "DATxxxxxx" file can still remain as a legacy option with known limitations. The new standard output, however, will have an internal directory structure. Different processes will produce their output in specific places in this structure. The content of the output file will change with the choice of the processes and their configurations.

HDF5 [24] is an obvious choice to be considered, while still having the potential disadvantage of being an external dependence. ROOT [25] could be another possible option.

3.4 Computational efficiency

Computational efficiency is not optional for ngC. The efficient use of expensive large-scale resources is a crucial requirement, and must be planned and considered from the early on. The priority is given to performance over run-time flexibility. The most fundamental settings of the simulation must be defined at compile time in a static way: the type of stack, including particle-level data content, physics models, environmental models, etc. Of course, all models can have additional parameters that can be defined and modified at run-time.

In general, the use of run-time dynamic design patterns like virtual classes or dynamic libraries should be minimized (i.e. no virtual methods). Static design patterns are preferred.

Data copy operations must be minimized, or performed as late as possible. The use of "lazy" functionality, which is executed only delayed and when actually needed should be promoted.

Compiler and CPU optimization should be fully considered for NGC. Production versions of the code should claim full benefits from all the available optimizations. The execution of particular code on GPUs or other hardware accelerators (or maybe even more custom hardware) must be transparently possible.

Parallel and multi-core computations are standard, and are built into the core of NGC.

4 Tools and infrastructure

The main development infrastructure for NGC will be provided by KIT. This is mostly the organization, discussion platform, scientific coordination, steering, and maintenance of the core functionality. The most useful tool for collaborative development available today is git. Git allows having a very dynamic and large base of contributors, and at the same time a well controlled access to the main code-base via *pull requests* (PRs). The code review, discussion, testing and validation of PRs will be an important task of the project steering. Code will be peer-reviewed, with an emphasis on clearness and readability, and inline documentation (doxygen). Furthermore, automatic unit-testing and validation will be performed. Unit tests must yield a very high coverage of the NGC code. Unit tests are executed automatically by a jenkins (or equivalent) service to perform low-level code and PR validation. Additional automatic validation and high-level tests must accompany the regular testing, cover all the important functionality and, in particular, all physics.

Automatic testing will provide a well defined list of supported environments, combined with a control over a specified set of different selections of simulation options.

We will use a gitlab server for the hosting, the default choice for this is `https://gitlab.ikp.kit.edu`. This gitlab server provides also an issue tracking functionality that will be linked to defined milestones. A wiki page service is also provided.

5 Main challenges

While there are many challenges to overcome, a list of topics that require particularly dedicated attention is given in the following. These topics are more-or-less directly linked to the underlying/internal physics of the cascade process and require very intelligent and likely highly-complex solution.

1. Efficient integration of electron-gamma cascades (previously EGS4).
2. Random number generation in an inherently multi-core and parallel environment while ensuring the full reproducibility of simulations.
3. Investigating the limits of equivalence between CE-solving and detailed transport methods ($\mathrm{d}E/\mathrm{d}X$, Cherenkov, lateral structure, radio production etc.).
4. GPU optimization.
5. Scalability in supercomputing environments.

6 Details

Taking the aforementioned considerations and requirements into account, a more detailed scheme of the simulation workflow becomes necessary, as outlined in Fig. 2. Some of the aspects of this diagram still need to be optimized or determined precisely. Nevertheless, with the basic design as given here, the modular functionality and building blocks can be developed in parallel. Rudimentary definitions of interfaces needed for these purposes are given below.

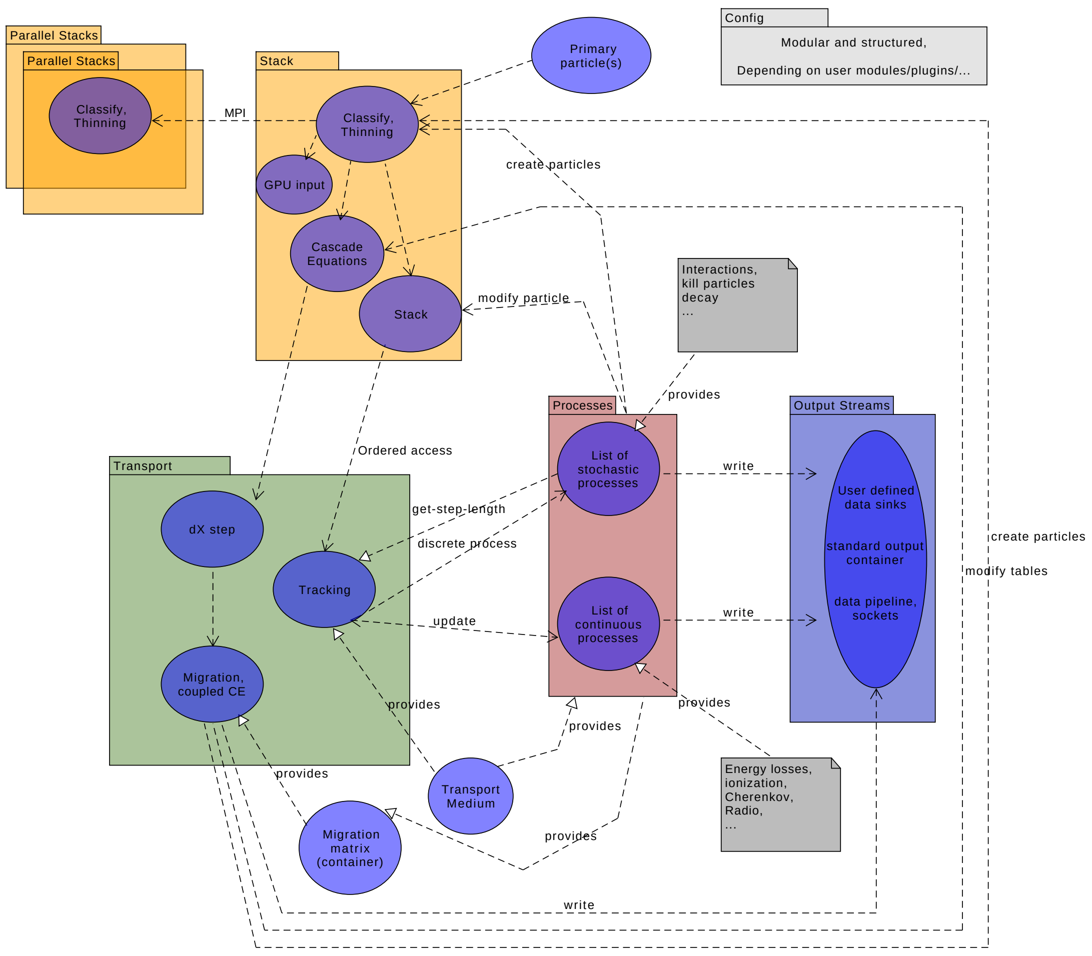

Figure 2: Main building blocks and workflow steps of ɴɢC which already highlight the fundamental functionality and flexibility.

Note that the code fragments given as examples here are, first of all, not in any specific language and do not follow any specific syntax. This is a pure pseudo-code used to illustrate the basic functionality and employed patterns, and only vaguely resembles C++.

6.1 Conventions and coding

A computer language offering high level of design flexibility and at the same time excellent compiler and optimization support is required. It is an advantage to chose a language that also has non-science relevance and thus assures long-term support, development, and expertise. For this purpose, we chose to use C++. At the beginning NGC will be based on the C++14 standard, a choice that will most probably evolve in the future.

General guidelines for contributing of the code will be well defined and must be strictly enforced [26]. These guidelines will be distributed via the documentation section on the gitlab server mentioned above and/or the wiki pages. The guidelines can be discussed, agreed upon, and also improved in discussions between the developers and the project steering. One of the most important things in such a project is communication – and the code will be the prime means of communication between the team members [27], since, let us not forget, most of the time people will spend on this project will be dedicated to reading other people's code [28]. A more exhaustive list of core guidelines for C++ can be found in Ref. [29]. Those items are also relevant in this respect:

- Code must be accompanied with inline comments. Note that a well chosen naming of identifiers and functions can greatly reduce the burden of documenting the code. A well written code is self-explanatory to a large extend. In addition, for systematic documentation doxygen commands must be used where possible.
- One aspect of choices of the style should be to minimize the probability of programming errors. For example pointers should be used only where absolutely necessary, and that should never be exposed to the user.
- We will favour static over dynamic polymorphism. On a low level of the code, this will lead to the abundant use of templates. However, high-level users and physicists should not be exposed to templates, unless absolutely required.
- Test-driven development is encouraged. Therefore, from early on, a useful setup of unit tests should be supported by the build system. The unit testing will be an essential part of NGC. A high coverage of code by tests will be a prime criterion for acceptance.

6.2 Dependencies

The use of external code and libraries must be minimized to the absolute minimum in order to stay conflict-free and operational over a very extended period of time. Individual exceptions might be possible, but must be well motivated and discussed before use. For each functionality we should evaluate whether a basic re-implementation is more feasible than inclusion of an external dependency. In any case, whenever possible, appropriate wrappers in NGC should hide the implementation details of external packages in order to keep replacement or re-implementation option open without a need for interface break. Likely packages and options for external libraries are (excluding packages that will be distributed together with NGC):

- C++14 compiler.
- CMake build system.
- git.
- doxygen.

- presumably `boost` for `yaml` and `xml`, histograms, file system access, command-line options, light-weight configuration parsers (property tree), random numbers etc.
- One package for unit testing, could also be from `boost`.
- HDF5 and/or ROOT for data storage.
- Eigen for linear algebra.
- PyBind11 in case a desire for Python bindings arises.
- HEPMC as generic interface, also for exotics.
- In order to generate random numbers, we will use standardized interfaces and established methods. For testing purposes, the possibility to exchange the random-number engine should be relatively easy. No homegrown generators and only well established, checked, and vetted methods for generating random numbers should be used, likely provided by `boost` as well.

6.3 Configuration

The framework has to support extensive run-time (from configuration files or on command line) as well as compile-time configuration. The latter involves conditional compilation, static polymorphism, switching between policies in policy-driven classes.

The run-time configuration will support structured yaml or xml as input, either in a single file, or multiple files located in a directory. Modules of NGC can retrieve the required configuration via a global object in a structured way. Command-line options are parsed and provided via the same mechanism. By default, the complete configuration will be saved into the output file, and will thus, if needed, allow identical reproduction of a simulation at a later time. Physics modules can access configuration via a *section name* and a *parameter name*, for example

```
primaryEnergy = Config.Get("PrimaryParticle/Energy");
```

where `PrimaryParticle` is the name of the configuration section, and `Energy` the parameter. The data can be obtained from files, or provided via the command line, for example via `--set PrimaryParticle/Energy=1e18_eV`.

For more intricate situations where a simple configuration file might not be sufficient, or when a dynamic change of parameters during runtime is needed, the simulation process can be more conveniently steered by means of a script. The library *PyBind11* allows us to provide bindings to Python with minimal efforts. It is up to the opinion of the community to decide whether this is a feature worth having, at the cost of additional external dependency.

6.4 Units

NGC will utilize `boost.units` or the header-only library *PhysUnits* [30] to allow for convenient attachment of units to the numerical literals (e.g. `auto criticalEnergy = 87_MeV;`), avoiding other, hard to enforce explicit conventions. In this way an otherwise silent error of mismatched units is converted to a compile-time error. During compilation the conversion of quantities to common base units (which the developer does not need to know and are internally chosen to minimize numerical errors) is performed. In addition, it will be possible in the future to change the base units in case this will be desired.

Furthermore, as information about the unit of a quantity is encoded in its type, the compiler performs a dimensional analysis of computations involving dimensionful quantities. This will not only improve the code readability but also make development less error-prone due to the compile-time check of unit commensurability. Thus, e.g. a typing error like in

```
auto distance = 35.2_cm;
auto time = 35.9_ns;
auto speed = distance + time; // compiler error!
```

will result in a compiler error instead of staying unnoticed. Because of this functionality, *PhysUnits* is superior to more simplistic implementations like e.g. provided in Geant4/CLHEP [22, 31], where units are provided only as a set of self-consistent multiplication constants.

These features introduce no runtime overhead when compiler optimizations are enabled since, after all, such a dimensionful quantity will be in memory just the usual floating-point number.

6.5 Lorentz transformation and geometry

Each time a Lorentz transformation is created, an inverse should be also stored. The most common use case will namely be when projectiles will require a transformation to enter the center-of-mass frame and the resulting secondary particles will need its inverse transformation to enter back into the shower frame. The idea is to be able to write code like this for energy-momentum vectors:

```
auto cms = utl::CreateCenterOfMassFrame(particle1, particle2);
auto energy1 = particle1.GetEnergy(cms);
auto momentum1 = particle1.GetMomentum(cms);
...
auto rest = Atmosphere::GetRestFrame();
auto energySec = secondary.GetEnergy(rest);
auto momentumSec = secondary.GetMomentum(rest);
```

Equivalent transformations will work also on space-time vectors but since interaction models do not require this information, it can be optimized away. The naive implementations of the transformations become numerically problematic at ultra-high energies (e.g. ROOT). The code offered in NGC must provide numerically stable transformations, preferably in the whole range between the very small kinetic energies and the ultra-high energies encountered in cosmic rays.

The coordinate-system implementation of the Offline framework can be a suitable starting point for the implementation of requirements in NGC.

6.6 Particle representation

The typical minimal set of information to describe a particle is: type, mass, energy-momentum, and space-time position. In certain use cases this can be extended for example with (multiple) weights, history information (unique ID, generation, grandparents, interaction ID), or further information.

Interaction models typically do not care about the space-time part since once the model is invoked according to the total cross-section, the impact parameter is decided internally by the model in a small Monte-Carlo procedure (and not e.g. from the microscopic positions of air nuclei in the atmosphere). Nevertheless, the propagation and the continuous losses will eventually need the space-time parts of the particle information.

Particle properties like mass and lifetime are extracted from the `ParticleData.xml`, file provided by PYTHIA 8 [32], together with their PDG code [33]. To allow for efficient conversions of particle properties as used in different interaction models, the NGC-internal particle code is chosen to be different than the PDG code. Since the PDG code only very sparsely covers a large integer range, they are not very useful as indices in a lookup table. NGC therefore uses a contiguous range of integers which is automatically generated from the union of all particles known by the user-enabled interaction models. Rather than using these integers directly in the NGC code, `enum` declarations will be provided for convenience and improved code readability. In contrast to their corresponding numerical values, the `enum` identifiers (e.g. `pDStarPlus`) are guaranteed to be unique.

For this purpose, the needed code is generated by a provided script before the actual compilation of NGC. This script will depend on the particle data `xml` file from PYTHIA. The output is C++ code that will allow to write expressions like these:

```
auto mElectron = ParticleData::GetMass(pEMinus); // is a static expression
auto tauPi = ParticleData::GetLifetime(pPiPlus); // also a static expression
...
auto particleType = stack.GetTopParticle().GetType();
auto charge = ParticleData::GetCharge(particleType);
```

The internal numeric particle-ID is just an index, the representation of particles in NGC code and `enums` is obtained from the particle names in the `xml` file. When specific interaction models internally use different schemes for particle identification, extra code is provided in the interface part to those models, where the conversion between the external and internal codes is performed.

For binary output purposes, however, NGC-internal codes are converted to the well known, standardized PDG codes to ensure seamless interoperability with other software packages used within the HEP community. In any text output, e.g. log files, the output is always converted to a human-readable identifiers. For example, `cout << someParticleCode << endl;` might, depending on the value of `someParticleCode`, printout "`e-`" or "`D+`" unless a numerical output (in NGC or PDG scheme) is explicitly requested.

6.7 Framework

The NGC consists of an inner core and associated modules that can also be entirely external. Thus, there can be – and generally is – a distinction between code in the "core" of NGC and "outside" of this, defining a "frontier" where conventions, units, and all kind of reference frames have to be adapted and converted in a consistent way. Most obviously is this the case for all existing hadronic event-generators and input/output facilities. Nevertheless, this can occur also in other components, and the frontier can thus occur at different places. The code needed for the conversions in the frontier must be provided together with the NGC framework. Special care must be taken in cases where different models, for example, use different constants for the mass of particles, which can lead to numerically unreasonable results like negative kinetic energies or invalid transformations. The details of such effects must be investigated and a comprehensive solution has to be found at a later time.

6.7.1 Particle processing and stacks

A core concept of NGC is that particles are stored on a dedicated stack. This is needed since in cascade processes an enormous number of particles can be accumulated, requiring careful handling of such data. The stack can automatically swap to disk when memory is exhausted. The access and handling of particles on the stack has an important impact on the performance of the simulation process. In typical applications it is optimal to process the lowest-energy particles first, but there can be situations where completely different strategy becomes relevant. The stack should be flexible enough to allow various user-specific interventions, while the simulation is writing and reading from it.

In NGC there is no need to have a dedicated persistent object describing an individual particle. Particles are always represented by a reference/proxy to the data on the stack. On a fundamental level, such stacks can be a FORTRAN common block, dynamically-allocated C++ data, a swap file, or any other source/storage of particle data.

6.7.2 Main loop, simulation steps, processes

A central part of NGC is the loop over all particles on the stack. These particles are transported and processed in interactions with the medium, and as part of this also cascade-equation tables can be filled. All these processes can produce new particles or modify existing particles on the stack. Furthermore, the processes can produce various output data of the simulation process. Cascade-equation migration-matrices are either computed at program start or read from pre-calculated files. When the stack is empty (or any other trigger), the cascade equations are solved numerically, which can, once more, also fill the particle stack. Thus, a double-loop is required here in order to process the full particle cascade:

```
while (!stack.Empty()) {
  while (!stack.Empty()) {
    auto particle = stack.Get();
    Step(particle);
  }
```

```
    cascadeEquations.Solve();
  }
```

The transport procedure needs to handle geometric propagation of neutral and charged particles, thus, magnetic and electric deflections are important. The transport step-length is used to distinguish two type of processes:

- *Continuous* processes occur on a scale much below the transport step-length, e.g. ionization, and thus an effective treatment can be used.
- *Discrete* processes typically lead to the disappearance of a particle and to production of new particles (typically in, but not limited to, collisions or decays).

The optimal size of the simulation step is determined from the list of all processes considered. The discrete process with the highest cross-section limits the maximum step size. However, also a continuous process can limit the step size, for example by the requirement that ionization energy-loss, the multiple-scattering angle, or the number of emitted Cherenkov photons cannot exceed specific limits. Furthermore, even particle transport is just a specific type of a continuous process which propagates particles. Since the propagation can lead a particle from one medium (e.g. the atmosphere) into another (e.g. ice), the particle transport can also have a limiting effect on the maximum step length allowed. An individual step cannot cross from one medium to another, but for correct treatment must terminate at the boundary between the two media. Furthermore, the particle transport in magnetic fields leads to deflections, where step size has to be adjusted according to the curvature of the deflection.

Thus, the geometric particle-transport must be the first process to be executed. The information about the particle trajectory is important input for the calculation of subsequent continuous processes. Finally, the type and probability of one single discrete process is last to be determined for each simulated transport step. The discrete process simulated is randomly selected, typically according to its cross section or lifetime. The structure of the code to execute in one simulation step is thus:

```
Step(auto particle)
{
  auto stepLength = MinimalStepLength(tracking, continuousProcesses, stochasticProcesses);
  auto trajectory = tracking.Propagate(particle, stepLength)
  for (auto cp : continuousProcesses) {
    cp.Propagate(particle, trajectory, stepLength);
  }
  // randomly select ONE or NONE stachastic process
  if (discreteProcess dp = SelectStochasticProcess(stepLength)) {
    dp.Interact(particle);
  }
}
```

The numerical solution of the cascade equations is performed as being functionally fully equivalent to a normal propagation. While some of the processes can easily be formulated using migration matrices, our aim is, though, to scientifically evaluate and exploit the concept as extensively as possible, covering the production of Cherenkov photons, radio emission, etc. The data for the cascade equations is stored in a *table* (which in general will cover multiple dimensions) representing histograms, for example of the number of particles of specific type versus energy. The *migration* of particles to different bins in energy *and* to different particle types is described by pre-computed migration-matrices. The matrices implicitly already encode the information on the geometric length of simulation steps. In some aspects the cascade-equation approach corresponds to the approximation where the discrete processes are handled like continuous processes. This is reflected in the structure of the corresponding code:

```
CascadeEquations::Solve()
{
  while (!table.Empty()) {
    for (auto cp : continuousProcesses) {
      cp.CascadeEquationPropagate(table)
    }
    for (auto dp : discreteProcesses) {
      dp.CascadeEquationPropagate(table)
    }
  }
}
```

The limits of the application of cascade equations to specific processes are not known precisely at this moment and certainly there are various challenges facing us ahead. Particularly difficult processes are those which depend significantly on geometry, like Cherenkov or radio emission. It is up to the detailed studies to evaluate their performance and adapt the methods to potential (limited) use cases. This will be subject of research as part of the project.

6.8 Radio

Radio-emission calculations, which were in the original CORSIKA provided by the CoREAS extension [34], rely on the position and timing information of charged particles to calculate the electromagnetic radiation associated with a particle shower. With its increased flexibility, ngC will enable radio-emission calculations for a much larger range of problems. In particular, simulation of the radiation, associated with showers penetrating from air into a dense medium or vice-versa, will become possible due to the more generic configuration of the interaction media. Feedback of the radio calculation to the cascade simulation (e.g., modifying simulation step-sizes or possibly thinning levels) might increase performance and/or simulation accuracy. GPU parallelization has the potential to greatly reduce computation times, which are currently the main bottleneck for simulations of signals in dense antenna arrays. Simulations in media with a sizable refractive-index gradient will require certain ray-tracing functionalities, possibly even finite-difference time-domain calculations. The modular approach of ngC will allow the implementation of different radio-emission calculation-formalisms and enable systematic studies of their differences.

6.9 Environment

Traditionally the medium of transport for CORSIKA was the Earth's atmosphere. It is one of the purposes of ngC to allow for much more flexible combination of environments. This includes water, ice, mountains, the moon, planets, stars, space, etc. In this case also the interface between different media becomes a matter of significance for the simulation. Showers can start in one medium and subsequently traverse into different media. The environment will be a dedicated object to configure for every physics application. The structure of the environment will be defined before compilation, the properties of the environment can be configured via configuration files in any way needed for the application. This can be either static, or time-dependent.

The global reference frame is specified by the user and depends on the chosen environmental model. For a standard curved Earth this is the center-of-the-earth frame. With double floating-point precision this yields a precision better than a nanometer over more than 10 000 km distance.

Particles are tracked in the global reference frame. The secondary particles produced by discrete processes occurring at specific locations in the cascade are transformed and boosted back into the global coordinate frame.

For specific purposes, like tabulations and some approximations, the *shower coordinate-system*, in which the z-axis points along the primary-particle momentum, can also be relevant. The transformation matrix from global to shower coordinate-system is provided for this purpose.

The initial randomization of primary-particle locations and directions is performed by dedicated modules, which can be changed and configured by the users to get on the detector level the desired distributions. The environment object provides all of the required access to the environmental parameters, e.g. roughly in the following form:

```
Environment::GetVolumeId(point)
Environment::GetVolumeBoundary(trajectory)
Environment::GetTargetParticle(point)
```

```
Environment::GetDensity(point)
Environment::GetIntegratedDensity(trajectory)
Environment::GetRefractiveIndex(point)
Environment::GetTemperature(point)
Environment::GetHumidity(point)
Environment::GetMagneticField(point)
Environment::GetElectricField(point)
```

This interface is sufficient since, for example a concept like altitude, defined as distance from a point to a surface on a direct line to the origin (center of the Earth), is needed only internally within the environment object.

The environment object will use a C++ policy concept to provide access to the underlying models. This requires re-compilation after changes in the model setup. However, individual models can still be configured at the runtime.

6.9.1 Geometric objects

We will keep the geometry description as simple as possible and to the level needed to achieve the physics goals. At the moment, this goals include being able to define different (typically very large) environment regions with distinct properties. Initially, it is sufficient to provide only the most simple forms and shapes, e.g. sphere, cuboid, cylinder, and maybe trapezoid as well as pyramid. The geometry package must be structured in a generic way, so that it can be extended, if needed, to include more complex and fine-grained objects at a later time. We are not planning to support general-purpose geometry as, for example in Geant4 [22]. When in a specific volume of the simulation a very complex geometry is required, it is probably the best choice to allow seamless integration of ngC with Geant4, where particles can be passed-on from one package to the other.

7 Summary

The steps towards creation of ngC outlined here are optimized to best support scientific research in fields where the simulation involves particle transport and particle cascades with stochastic and continuous processes. The targeted goals of the resulting framework will be far beyond the capabilities of the original CORSIKA program. It is up to the scientific community to decide in which concrete applications ngC will be used in the future. It is our aim to offer long-term support for the ngC program over a period of more than 20 years.

The modularity of the proposed code and the magnitude of the project offers the opportunity for the scientific community to participate in a collaborative manner. Specific functionality and modules can be provided and maintained by different groups. The core of the project, the integration, and the steering are provided by KIT. This can be also a suitable model for a scenario where different communities have different requirements, but the overall collaborative approach is the one we want to promote and foster. This will require dedicated and strict commitment to the project from all the participating parties in order to assure the stability and functionality with no compromises needed.

A better access to the air-shower physics-simulation process will be one of the keys to resolve the main open questions of cosmic-ray physics, the universe at the highest energies, and related scientific problems.

8 Acknowledgements

T.H. acknowledges very fruitful discussions within the radio-detection community. M.R. acknowledges support by the DFG-funded Doctoral School "Karlsruhe School of Elementary and Astroparticle Physics: Science and Technology".

References

[1] D. Heck, J. Knapp, J. N. Capdevielle, G. Schatz and T. Thouw, *CORSIKA: A Monte Carlo code to simulate extensive air showers*, Tech. Rep. FZKA-6019, Forschungszentrum Karlsruhe, 1998. https://publikationen.bibliothek.kit.edu/270043064.

[2] J. R. Hörandel, *A Review of experimental results at the knee*, J. Phys. Conf. Ser. **47** (2006) 41 [astro-ph/0508014].

[3] MACRO collaboration, M. Ambrosio et al., *The MACRO detector at Gran Sasso*, Nucl. Instrum. Meth. A **486** (2002) 663.

[4] A. Daum et al., *First results on the performance of the HEGRA IACT array*, Astropart. Phys. **8** (1997) 1.

[5] J. Knapp, D. Heck, S. J. Sciutto, M. T. Dova and M. Risse, *Extensive air shower simulations at the highest energies*, Astropart. Phys. **19** (2003) 77 [astro-ph/0206414].

[6] A. M. Hillas, *Shower simulation: Lessons from MOCCA*, Nucl. Phys. B Proc. Suppl. **52** (1997) 29.

[7] S. J. Sciutto, *AIRES: A System for air shower simulations. User's guide and reference manual. Version 2.2.0*, astro-ph/9911331.

[8] H.-J. Drescher and G. R. Farrar, *Air shower simulations in a hybrid approach using cascade equations*, Phys. Rev. D **67** (2003) 116001 [astro-ph/0212018].

[9] CTA Consortium collaboration, M. Actis et al., *Design concepts for the Cherenkov Telescope Array CTA: An advanced facility for ground-based high-energy gamma-ray astronomy*, Exper. Astron. **32** (2011) 193 [1008.3703].

[10] IceCube collaboration, A. Achterberg et al., *First Year Performance of The IceCube Neutrino Telescope*, Astropart. Phys. **26** (2006) 155 [astro-ph/0604450].

[11] H.E.S.S. collaboration, J. A. Hinton, *The Status of the H.E.S.S. project*, New Astron. Rev. **48** (2004) 331 [astro-ph/0403052].

[12] MAGIC collaboration, J. Aleksić et al., *The major upgrade of the MAGIC telescopes, Part II: A performance study using observations of the Crab Nebula*, Astropart. Phys. **72** (2016) 76 [1409.5594].

[13] Pierre Auger collaboration, J. Abraham et al., *Properties and performance of the prototype instrument for the Pierre Auger Observatory*, Nucl. Instrum. Meth. A **523** (2004) 50.

[14] Telescope Array collaboration, T. Abu-Zayyad et al., *The surface detector array of the Telescope Array experiment*, Nucl. Instrum. Meth. A **689** (2013) 87 [1201.4964].

[15] M. P. van Haarlem et al., *LOFAR: The LOw-Frequency ARray*, Astron. Astrophys. **556** (2013) A2 [1305.3550].

[16] D. Heck and R. Engel, *The EHISTORY Option of the Air-Shower Simulation Program CORSIKA*, Tech. Rep. FZKA-7495, Forschungszentrum Karlsruhe, 2009. https://publikationen.bibliothek.kit.edu/270078292.

[17] A. Fedynitch, R. Engel, T. K. Gaisser, F. Riehn and T. Stanev, *MCE_Q - numerical code for inclusive lepton flux calculations*, PoS **ICRC2015** (2016) 1129.

[18] T. Bergmann, R. Engel, D. Heck, N. N. Kalmykov, S. Ostapchenko, T. Pierog et al., *One-dimensional Hybrid Approach to Extensive Air Shower Simulation*, Astropart. Phys. **26** (2007) 420 [astro-ph/0606564].

[19] D. Baack, *Data Reduction for CORSIKA*, Tech. Rep. Baack/2016a, TU Dortmund, SFB 876, 2016. https://sfb876.tu-dortmund.de/PublicPublicationFiles/baack_2016a.pdf.

[20] R. Ulrich, *COAST*, 2005–2013. https://web.ikp.kit.edu/rulrich/coast.html.

[21] S. Argiró, S. L. C. Barroso, J. Gonzalez, L. Nellen, T. C. Paul, T. A. Porter et al., *The Offline Software Framework of the Pierre Auger Observatory*, *Nucl. Instrum. Meth. A* **580** (2007) 1485 [0707.1652].

[22] GEANT4 collaboration, S. Agostinelli et al., *GEANT4: A Simulation toolkit*, *Nucl. Instrum. Meth. A* **506** (2003) 250.

[23] E. Armengaud, G. Sigl, T. Beau and F. Miniati, *CRPropa: a numerical tool for the propagation of UHE cosmic rays, gamma-rays and neutrinos*, *Astropart. Phys.* **28** (2007) 463 [astro-ph/0603675].

[24] The HDF Group, *Hierarchical Data Format, version 5*, 1997–2018. http://www.hdfgroup.org/HDF5/.

[25] R. Brun and F. Rademakers, *ROOT: An object oriented data analysis framework*, *Nucl. Instrum. Meth. A* **389** (1997) 81.

[26] "C++ FAQ." https://isocpp.org/faq.

[27] N. C. Zakas, *Why Coding Style Matters*, 2012. https://www.smashingmagazine.com/2012/10/why-coding-style-matters/.

[28] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2009.

[29] "C++ Core Guidelines." https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.html.

[30] M. Moene, *PhysUnits C++11*, 2018. https://github.com/martinmoene/PhysUnits-CT-Cpp11.

[31] L. Lönnblad, *CLHEP: A project for designing a C++ class library for high-energy physics*, *Comput. Phys. Commun.* **84** (1994) 307.

[32] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten et al., *An Introduction to PYTHIA 8.2*, *Comput. Phys. Commun.* **191** (2015) 159 [1410.3012].

[33] Particle Data Group collaboration, C. Patrignani et al., *Review of Particle Physics*, *Chin. Phys. C* **40** (2016) 100001.

[34] T. Huege, M. Ludwig and C. W. James, *Simulating radio emission from air showers with CoREAS*, *AIP Conf. Proc.* (2013) 128 [1301.2132].