ORIGINAL ARTICLE

CrossMark

Towards A Next Generation of CORSIKA: A Framework for the Simulation of Particle Cascades in Astroparticle Physics

Ralph Engel[1,2] · Dieter Heck[1] · Tim Huege[1,3] · Tanguy Pierog[1] · Maximilian Reininghaus[1] · Felix Riehn[4] · Ralf Ulrich[1] · Michael Unger[1] · Darko Veberič[1]

Abstract

A large scientific community depends on the precise modeling of complex processes in particle cascades in various types of matter. These models are used most prevalently in cosmic ray physics, astrophysical-neutrino physics, and gamma ray astronomy. In this white paper, we summarize the necessary steps to ensure the evolution and future availability of optimal simulation tools. The purpose of this document is not to act as a strict blueprint for next-generation software, but to provide guidance for the vital aspects of its design. The topics considered here are driven by physics and scientific applications. Furthermore, the main consequences of implementation decisions on performance are outlined. We highlight the computational performance as an important aspect guiding the design, since future scientific applications will heavily depend on an efficient use of computational resources.

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 [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 for many types of experiments to assess the physics content of their data, and almost impossible to qualify the compatibility with different measurements. In general, the simulation of extensive air showers was recognized as one of the fundamental prerequisites for successful 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] (with the extension TIERRAS [8] for simulations of showers below ground), and SENECA [9].

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

✉ Ralf Ulrich
ralf.ulrich@kit.edu

Ralph Engel
ralph.engel@kit.edu

1   Institut für Kernphysik, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany

2   Institut für Experimentelle Teilchenphysik, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany

3   Vrije Universiteit Brussel (VUB), Brussels, Belgium

4   Laboratório de Instrumentaço e Física Experimental de Partículas (LIP), Lisboa, Portugal

Springer

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 that 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 the required environments.

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 only, 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. In particular, the excellent performance of thinning is critical for simulations at the highest energies [10–12]. With NGC, it will be possible to study thinning very precisely with the techniques known as multi-thinning [13], in combination with a deep analysis of the cascade history. It is important to improve the thinning performance and technology with respect to the solutions available so far. 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.

We expect that millions, if not billions, of CPU hours of high-performance computing will be spent in the future on air shower simulations for experiments like CTA [14], H.E.S.S. [15], IceCube [16], LOFAR [17], MAGIC [18], the Pierre Auger Observatory [19], the Telescope Array [20], and other next-generation experiments. It is up to NGC to make sure that this is done as efficiently and accurately as possible, while maximizing 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.

Main Design Considerations

Some of the goals to achieve with NGC are extensibility, flexibility, 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. It is one aim of the NGC projects to reflect the structural simplicity of Fig. 1 to a very large degree.

Overcoming Limitations of Current CORSIKA

The current CORSIKA implementation has a number of limitations, originating mostly from optimization to specific use cases as well as the adaption to more-and-more novel use cases which were previously unconsidered in the design of CORSIKA. These limitations, most of which we intend to remedy in NGC, and our anticipated improvements include the following items:

1. The interaction medium is air with its density being analytically modeled by five piecewise-exponential layers. NGC should support arbitrary media (air, liquid and frozen water, 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 feedback, 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_L^0/K_S^0). 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 [21]) is very limited due to rigid memory layout.

Fig. 1 Scheme of particle transport code with processes
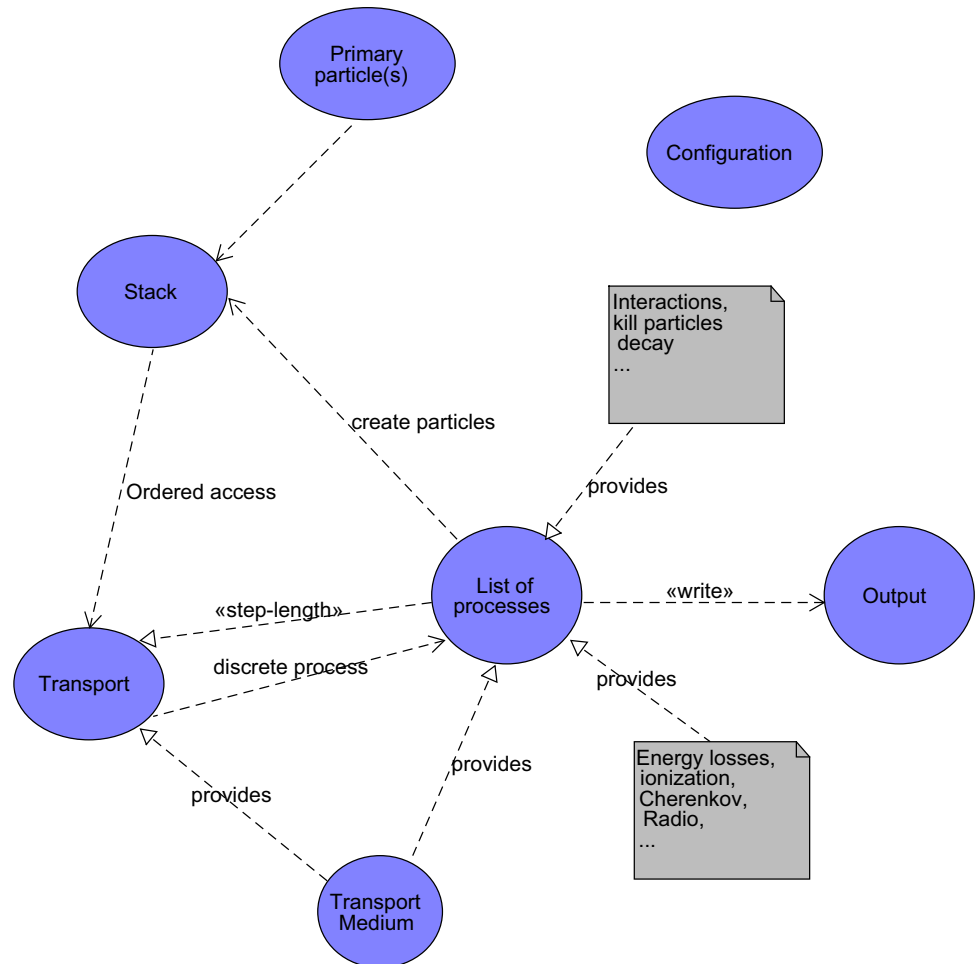
6. No upward-going Cherenkov photons can be handled.
7. It is not envisaged that a started shower simulation is *canceled* for any reason, for example when it could be flagged during the simulation process as being not relevant for a specific physics study.
8. Nuclei are supported only up to $Z = 60$ and $A = 99$.
9. No standardized visualization and validation tools for detailed inspections are provided.

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 [22] is a recent tool dedicated to the numerical solution of the CE. It already offers GPU support and very high computational efficiency. CONEX [23] is a CE air shower simulation program that has been integrated in CORSIKA and provides enormous increase in computation speed.

– Dynstack [24] is a recent extension of CORSIKA. Its basic functionality should be adopted for the stack of NGC.
– COAST [25] 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 [26], 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 an emphasis on different aspects than what is needed in air shower simulations; examples are CRPropa [27] and GEANT4 [28], although the latter is sometimes used for air showers simulations, too [29, 30].

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 the 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 [31] is an obvious choice to be considered, while still having the potential disadvantage of being an external dependence. ROOT [32] could be another possible option. In any case, we will provide a flexible output interface that physics modules can rely upon in an implementation-agnostic way.

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., avoidance of virtual methods in hot code paths). 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 the result is 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.

Tools and Infrastructure

The main development infrastructure for NGC will be provided by our group at KIT. This is mostly the organization, discussion platform, scientific coordination, steering, and maintenance of the core functionality. The most useful and widespread tool for collaborative development available today is the version control system which 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 codebase 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, and 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 use the gitlab server https://gitlab.ikp.kit.edu for the hosting. This gitlab server also provides an issue tracking functionality that is linked to defined milestones. A wiki page service is also provided. Connect to this server to see the status of the NGC project, download releases, or even get directly involved in discussions or the development.

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 Monte Carlo transport methods (dE/dX, Cherenkov, lateral structure, radio production, etc.);
4. GPU optimization;
5. scalability in supercomputing environments.

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.

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++.

Conventions and Coding

A programming 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 decided to use C++. At the beginning, NGC will be based on the C++17 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 [33]. 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 [34], since, let
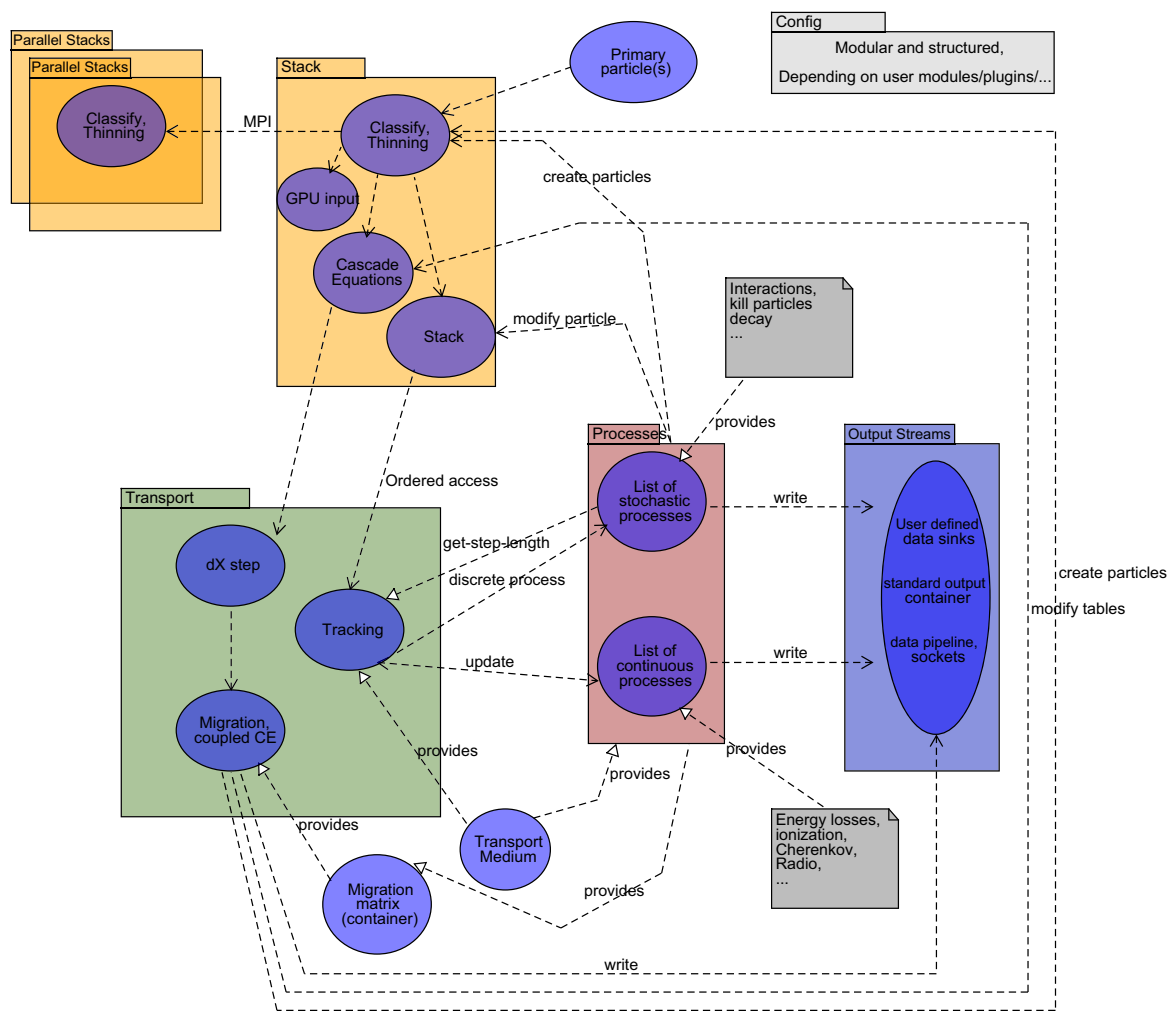

Fig. 2 Main building blocks and workflow steps of NGC which already highlight the fundamental functionality and flexibility

us not forget, most of the time people will spend on this project which will be dedicated to reading other people's code [35]. A more exhaustive list of core guidelines for C++ can be found in Ref. [36]. 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 extent. 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 favor 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.

Dependencies

The use of external code and libraries must be minimized to the absolute minimum 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 getting included into the mainline code. For each functionality, we should evaluate whether a basic reimplementation 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 to keep replacement or re-implementation option open without a need for breaking the interface. Likely packages and options for external libraries are (excluding packages that will be distributed together with NGC):

– C++17 compiler.
– CMake build system.
– git [for development].
– doxygen [for development].
– presumably `boost` for `yaml` and `xml`, histograms, file system access, command-line options, light-weight configuration parsers (property tree), random numbers, etc.
– HDF5 and/or ROOT for data storage [at least one of both required].
– PyBind11 [37] for bindings to Python.

– HepMC [38] as generic interface, also for exotics [optional].
– 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.

Light-weight packages like small header-only libraries can be distributed together with NGC. Likely candidates are:

– Eigen3 [39] for linear algebra.
– catch2 [40] for unit tests.
– PhysUnits [41] for units (see below).

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, and 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 `−−setPrimaryParticleEnergy = 1e18_eV`.

For more intricate situations where a simple configuration file might not be sufficient, or when a dynamic change of parameters during run time 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.

Units

NGC will utilize the header-only library *PhysUnits* for handling quantities having physical dimensions (i.e., "units"). First, it allows us to conveniently attach units to the numerical literals in the code (e.g., `auto criticalEnergy = 87_MeV;`),

thereby avoiding other, hard to enforce explicit conventions and improving readability, especially in a collaborative environment.

Second, as the dimensions of quantities are encoded in their respective types, a dimensional analysis is imposed upon computations involving dimensionful quantities during the compilation. This way, an otherwise silent error of mismatched units is converted to a compile-time error, as in the following example:

```
Length_t distance = 47.2_cm;
Time_t time = 35.9_ns;
Speed_t speed = distance + time; // compiler error!
Frequency_t freq = 1 / distance; // compiler error!
```

During compilation, the conversion of quantities to common base units (which the developer does not need to know and is internally chosen to minimize numerical errors) is performed.

Because of this functionality, this approach is more restrictive than more simplistic implementations like, e.g., provided in Geant4/CLHEP [28, 42], where units are provided only as a set of self-consistent multiplication constants. We believe, nevertheless, that the use of "strongly-typed units" will make development less error-prone.

At the same time, no run-time overhead is introduced when compiler optimizations are enabled since, after all, such a dimensionful quantity in memory is just the usual floating-point number.

Geometry, Coordinate Systems, and Transformations

A key ingredient to the usability of NGC is the ability to conveniently work with geometrical objects such as points, vectors, trajectories, etc., possibly defined in different coordinate systems. We will provide a geometry framework (with unit support fully integrated), to a large extent inspired by $\overline{\text{Offline}}$, in which geometrical objects are defined always with a reference to a specific coordinate system. In our case, the relevant coordinate systems mainly comprise the environmental reference frame and the shower frame, but additional systems can be defined as needed. When dealing with multiple objects at the same time, e.g., sphere.IsInside(point), is it automatically taken care of transforming the affected objects into a common reference frame. Therefore, when one can formulate his computations in a way that does not involve any specific coordinate system, the handling of potentially necessary transformations stays completely transparent.

As possible transformations that define coordinate systems with respect to each other, we restrict ourselves to the elements of the special Euclidean group $SE(3)$ (see ref. [43]), i.e., rotations and translations. Although one might favor Poincaré transformations as they include Lorentz boosts, which are certainly required for interfacing external interaction models, this would require to add a time-like coordinate to all geometric objects. This adds a significant complexity to the code in our setup that is otherwise completely static. For example, the concept of a point fixed in space in the lab frame would require to be upgraded to a world line. We currently do not envisage to support modeling of relativistic moving objects in our environment—except for the particles, of course—as this would significantly complicate and slow down our particle tracking algorithms. Due to the special properties of rotations and translations, it is not computationally expensive to perform inverse transformations, because expensive matrix inversions can be avoided.

Regarding the aforementioned Lorentz boosts, special attention must be paid to ensure numerically accurate results in all relevant regimes, comprising the range from non-relativistic ($\beta \ll 1, \gamma \simeq 1$) to ultra-relativistic ($\beta \simeq 1, \gamma \gg 1$) boosts.

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, and 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 determined internally by the model in a small Monte Carlo procedure (and not 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 [44], together with their PDG code [45]. To allow for efficient lookup of these properties, the NGC-internal particle code is chosen to be different than the PDG code. Since the PDG codes only very sparsely cover 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, however, enum declarations will be provided for convenience and improved code readability. In contrast to their corresponding numerical values, the enum identifiers (e.g., Code :: DStarMinus) are

guaranteed to be stable after recompilation with different interaction modules, as well as in future NGC releases.

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

```
// compile − time evaluated expressions :
auto constexpr mElectron = ParticleData :: GetMass(Code :: Electron);
auto constexpr tauPi = ParticleData :: GetLifetime(Code :: PiPlus);
...
// run − time evalueted expressions :
auto particleType = stack.GetNextParticle().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 by default converted to a human-readable identifiers. For example, `cout << someParticleCode << endl;` might, depending on the value of `someParticleCode`, print out "e−" or "D+" unless a numerical output (in NGC or PDG scheme) is explicitly requested.

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 kinds of reference frames have to be adapted and converted in a consistent way. Most obviously is the case for all the 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.

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 in terms of memory footprint 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 to 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 an FORTRAN common block, dynamically-allocated C++ data, a swap file, or any other source/storage of particle data.

Main Loop, Simulation Steps, and 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 CE tables can be filled. All these processes can produce new particles or modify the existing particles on the stack. Furthermore, the processes can produce various output data of the simulation process. CE migration matrices are either computed at program start or read from pre-calculated files. When the stack is empty (or any other trigger), the CE are solved numerically, which can, once more, also fill the particle stack. Thus, a double-loop is required here to process the full particle cascade:

```
while (!stack.Empty()) {
  while (!stack.Empty()) {
    auto particle = stack.GetNextParticle();
    Step(particle);
  }
  cascadeEquations.Solve();
}
```

The transport procedure needs to handle geometric propagation of neutral and charged particles, and thus, magnetic and electric deflections are important. The transport step length is used to distinguish two types 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 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 simulated discrete process 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(Particle& 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 CE 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 CE are 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 CE approach corresponds to the approximation where the discrete processes are handled like continuous processes. This is reflected in the structure of the corresponding code: The limits of the application of CE 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.

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

Radio

Radio emission calculations, which, in the original CORSIKA, are provided by the CoREAS extension [46], 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.

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 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 run time.

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, these 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 [28]. 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.

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 offer 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 that we want to promote and foster. This will require dedicated and strict commitment to the project from all the participating parties 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 address the main open questions of cosmic ray physics, the universe at the highest energies, and related scientific problems.

References

1. Heck D, Knapp J, Capdevielle JN, Schatz G, Thouw T (1998) CORSIKA: a monte carlo code to simulate extensive air showers. Technical Report FZKA-6019, Forschungszentrum Karlsruhe
2. Hörandel Jörg R (2006) A review of experimental results at the knee. J Phys Conf Ser 47:41
3. Ambrosio M et al (2002) The MACRO detector at Gran Sasso. Nucl Instrum Method A 486:663
4. Daum A et al (1997) First results on the performance of the HEGRA IACT array. Astropart Phys 8:1
5. Knapp J, Heck D, Sciutto SJ, Dova MT, Risse M (2003) Extensive air shower simulations at the highest energies. Astropart Phys 19:77
6. Hillas AM (1997) Shower simulation: lessons from MOCCA. Nucl Phys B Proc Suppl 52:29
7. Sciutto SJ (1999) AIRES: a system for air shower simulations. User's guide and reference manual. Version 2.2.0
8. Tueros Matias, Sciutto Sergio (2010) TIERRAS: a package to simulate high energy cosmic ray showers underground, underwater and under-ice. Comput Phys Commun 181:380
9. Drescher Hans-Joachim, Farrar Glennys R (2003) Air shower simulations in a hybrid approach using cascade equations. Phys Rev D 67:116001
10. Risse M, Heck D, Ostapchenko S, Knapp J (2002) EAS simulations at Auger energies with CORSIKA
11. Kobal M (2001) A thinning method using weight limitation for air-shower simulations. Astropart Phys 15:259
12. Billoir Pierre (2008) A sampling procedure to regenerate particles in a ground detector from a 'thinned' air shower simulation output. Astropart Phys 30:270
13. Bruijn R, Knapp J, Valino I (2011) Study of statistical thinning with fully-simulated air showers at ultra-high energies. Proceedings of the ICRC2011, p 39
14. Actis M et al (2011) Design concepts for the Cherenkov Telescope Array CTA: an advanced facility for ground-based high-energy gamma-ray astronomy. Exper Astron 32:193
15. Hinton JA (2004) The status of the H.E.S.S. project. New Astron Rev 48:331
16. Achterberg A et al (2006) First year performance of the IceCube neutrino telescope. Astropart Phys 26:155

17. van Haarlem MP et al (2013) LOFAR: the low-frequency array. Astron Astrophys 556:A2

18. Aleksić J et al (2016) The major upgrade of the MAGIC telescopes, Part II: a performance study using observations of the Crab Nebula. Astropart Phys 72:76

19. Abraham J et al (2004) Properties and performance of the prototype instrument for the Pierre Auger Observatory. Nucl Instrum Method A 523:50

20. Abu-Zayyad T et al (2013) The surface detector array of the telescope array experiment. Nucl Instrum Method A 689:87

21. Heck D, Engel R (2009) The EHISTORY option of the air-shower simulation program CORSIKA. Technical report FZKA-7495, Forschungszentrum Karlsruhe

22. Fedynitch A, Engel R, Gaisser TK, Riehn F, Stanev T (2016) MCE_Q-numerical code for inclusive lepton flux calculations. PoS, ICRC2015:1129

23. Bergmann Till, Engel R, Heck D, Kalmykov NN, Ostapchenko Sergey, Pierog T, Thouw T, Werner K (2007) One-dimensional hybrid approach to extensive air shower simulation. Astropart Phys 26:420

24. Baack Dominik (2016) Data reduction for CORSIKA. Technical report Baack/2016a, TU Dortmund, SFB 876

25. Ulrich Ralf COAST. https://web.ikp.kit.edu/rulrich/coast.html

26. Argiró S, Barroso SLC, Gonzalez J, Nellen L, Paul TC, Porter TA, Prado L Jr, Roth M, Ulrich R, Veberič D (2007) The offline software framework of the Pierre Auger observatory. Nucl Instrum Method. A 580:1485

27. Armengaud Eric, Sigl Gunter, Beau Tristan, Miniati Francesco (2007) CRPropa: a numerical tool for the propagation of UHE cosmic rays, gamma-rays and neutrinos. Astropart Phys 28:463

28. Agostinelli S et al (2003) GEANT4: a simulation toolkit. Nucl Instrum Method A 506:250

29. Anchordoqui LA, Cooperman G, Grinberg V, McCauley TP, Paul Thomas Cantzon, Reucroft S, Swain JD, Alverson G (2000) Air shower simulation using GEANT4 and commodity parallel computing. In Proceedings of 11th International Symposium on Very High Energy Cosmic Ray Interactions

30. Sanjeewa Hakmana, He Xiaochun, Cleven Christopher (2007) Air shower development simulation program for the cosmic ray study. Nucl Instrum Method B 261(1):918

31. The HDF group. Hierarchical data format, version 5. http://www.hdfgroup.org/HDF5/

32. Brun R, Rademakers F (1997) ROOT: an object oriented data analysis framework. Nucl Instrum Method A 389:81

33. C++ FAQ. https://isocpp.org/faq

34. Zakas Nicholas C (2012) Why Coding Style Matters.

35. Martin CR (2009) Clean Code: a handbook of Agile Software Craftsmanship. Prentice Hall, Upper Saddle River

36. Stroustrup Bjarne, Sutter Herb C++ Core Guidelines. https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.html

37. Jakob Wenzel et al PyBind11. https://pybind11.readthedocs.io

38. Dobbs Matt, Hansen Jorgen Beck (2001) The HepMC C++ Monte Carlo event record for high energy physics. Comput Phys Commun 134:41

39. Guennebaud Gaël, Jacob Benoît et al Eigen v3. https://eigen.tuxfamily.org

40. Catch2. https://github.com/catchorg/Catch2

41. Moene Martin PhysUnits C++11. https://github.com/martinmoene/PhysUnits-CT-Cpp11

42. Lönnblad Leif (1994) CLHEP: a project for designing a C++ class library for high-energy physics. Comput Phys Commun 84:307

43. Ivancevic Vladimir G, Ivancevic Tijana T (2011) Lecture Notes in Lie Groups.

44. Sjöstrand Torbjörn, Ask Stefan, Christiansen Jesper R, Corke Richard, Desai Nishita, Ilten Philip, Mrenna Stephen, Prestel Stefan, Rasmussen Christine O, Skands Peter Z (2015) An introduction to PYTHIA 8.2. Comput Phys Commun 191:159

45. Tanabashi M et al (2018) Review of particle physics. Phys Rev D 98(3):030001

46. Huege T, Ludwig M, James CW (2013) Simulating radio emission from air showers with CoREAS. AIP Conf Proc 1535:128