EPJ Web of Conferences **251**, 03038 (2021) https://doi.org/10.1051/epjconf/202125103038
CHEP 2021

CORSIKA 8

A novel high-performance computing tool for particle cascade Monte Carlo simulations

Antonio Augusto Alves Jr.[1], *Maximilian* Reininghaus[1,2], *André* Schmidt[1], *Remy* Prechelt[3], and *Ralf* Ulrich[1,*] for the CORSIKA 8 collaboration

[1]Karlsruher Institut für Technologie, Institut für Astroteilchenphysik, Karlsruhe, Germany
[2]Instituto de Tecnologías en Detección y Astropartículas, Buenos Aires, Argentina
[3]Department of Physics & Astronomy, University of Hawai'i at Manoa, Honolulu, USA

Abstract. The CORSIKA 8 project is an international collaboration of scientists working together to deliver the most modern, flexible, robust and efficient framework for the simulation of ultra-high energy secondary particle cascades in matter. The main application is for cosmic ray air shower simulations, but it can also be applied to other problems in astro(particle)-physics, particle physics and nuclear physics. Besides a comprehensive and state-of-the-art collection of physics models as well as algorithms relevant for the field, also all possible interfaces to hardware acceleration (e.g. GPU) and parallelization (vectorization, multi-threading, multi-core) will be provided. We present the status and roadmap of this project. This code will soon be available for novel explorative studies and phenomonological research, and at the same time for massive productions runs for experiments.

1 Introduction

One of the supporting pillars of current astroparticle physics is the existence of a common reference frame for a vast class of measurements and studies: the legacy CORSIKA air shower simulation program [1] that is written in FORTRAN 77 and whose early stages date back some 30 years [2]. This package made it possible to perform meaningful comparative, incremental and progressing research in the field for several decades. With CORSIKA 8 we follow up on this approach and consequently transform it to a modern computing landscape. While solidly founded on the success of the previous CORSIKA versions, CORSIKA 8 [3] is fundamentally community-driven[1] and bundles modern concepts for computing with the goal for the most comprehensive and flexible physics description for the simulation of particle cascades. It will bring novel modularity to enable optimal and fine-tuned modelling as well as future developments. This will also make it possible to better relate the importance of model parameter uncertainties on final physics observables. At the same time modern high-performance computing concepts will be made available throughout the framework to users.

*e-mail: ralf.ulrich@kit.edu
[1]development and discussions happen at https://gitlab.ikp.kit.edu/AirShowerPhysics/corsika

For reasons of maximum performance and modularity we chose C++ as basic language of the framework. Currently we use the version 17 of the language because of its wide availability. We will incorporate new features (C++20, C++23) when they become sufficiently available and provide major benefits to the project. Important updates in the C++ language are expected, for example, in the field of parallel computing algorithms (cf. e.g. [4]) and also in general modularity (cf. e.g. [5]).

A very basic comparison of simulation results is shown in this proceedings, but a detailed analysis is beyond the scope here and will be published appropriately.

2 Scientific context and open problems

It is a common problem in astroparticle physics to have high or ultra-high energy particles interacting with matter to form secondary particle cascades. This can happen in astrophysical context and produce gamma-ray and neutrino emission, but most notable also in our atmosphere where huge cosmic-ray-induced extensive air shower cascades (EAS) are produced and observed by experiments. The precise interpretation of such experimental data in terms of fundamental properties of the primary cosmic ray particles is a key to understanding many aspects of the nature of the ultra-high energy universe. At the highest energies major open problems are to get a more precise measurement of the primary mass composition of cosmic rays and also to better understand how such particles interact.

The precise description of EAS is a very challenging endeavour. Hadron collisions must be described with good quality from hundreds of TeV of center-of-mass energy down to GeV. This, most importantly, includes the relatively poorly known forward phase space, thus, even requires dedicated input from accelerators. Cascades develop in air, which is a mixture of light nuclei, making the modelling of nuclear effects very important. Electromagnetic interactions must consider the density-dependent high-energy Landau–Pomeranchuk–Migdal (LPM) effect [6–8] which suppresses cross sections at high energies as well as density-corrections at low energies.

Astroparticle physics observatories depend on the modelling of EAS for their design, operation and also for their data analyses. This is most obvious for indirect cosmic ray experiments, regardless if they measure particles at ground, fluorescence light, Cherenkov light, radio emission, or something else (e.g. [9]). But also neutrino observatories require a very precise understanding of their atmospheric backgrounds, which are immense and extremely difficult to simulate. Finally, also the PeV gamma ray observatories need simulations for signal and background studies, sometimes even in a time-dependent and per-source setting.

Thus, basically all emission produced in EAS are used by experiments. CORSIKA 8 is the tool to link microscopic models to the final experimental observables. One of the big challenges is the better understanding of how uncertainties in input models affect observables.

In fig. 1 the 3D output of one EAS simulation performed with CORSIKA 8 is shown.

3 Design overview and infrastructure

CORSIKA 8 is a framework. It provides all the infrastructure to write concrete physics applications. The main goal driving the development are

- **Modularity:** It is a fundamental requirement that all algorithms and physics models are interfaced in a transparent, simple and modular way. Physics processes are provided in *modules* with a minimal and obvious scope. It should be as simple as possible to understand what belongs to specific models and how they work. All assumptions and parameters are transparent and can be changed. New models can be designed to extend

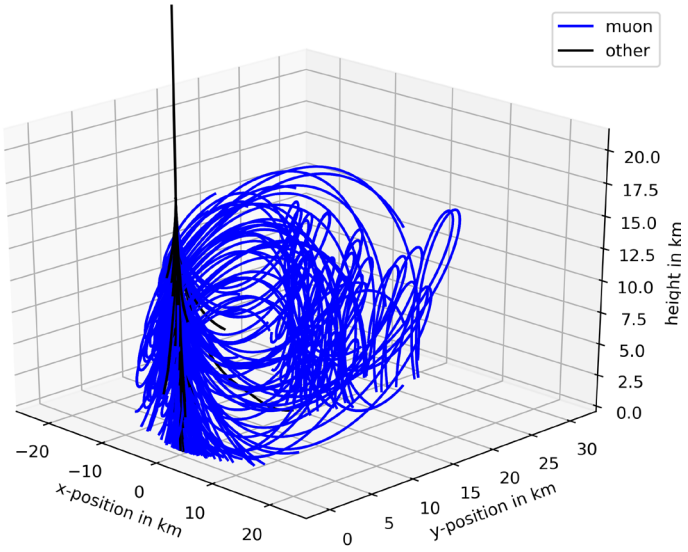

Figure 1. Example of one simulated proton shower at $E_0 = 10^{14}$eV with CORSIKA 8, with propagation and interaction of hadrons and muons including magnetic deflection. For illustrative purpose the magnetic field in this example is very high (15 mT with 45° inclination). Only particles above 20 GeV are followed. Black are hadrons, blue are muons.

or replace other modules. There are different classes of modules to describe the physics of cascades, depending on what data they need to access at what time and how they are allowed to modify it: `ContinuousProcess`, `InteractionProcess`, `DecayProcess`, `SecondariesProcess`, `StackProcess`, `BoundaryCrossingProcess` [10].

- **Completeness:** It is one purpose of CORSIKA 8 to not just offer a single reference implementation but at best all solutions available on the market. This will make it possible to study systematic effects and find grounds for further model improvements. Also it will create new links and synergies over the boundaries of previous projects. It was shown in the past that a platform for the sharing and inclusion of algorithms can be very important for an entire community – CORSIKA 8 takes this principle seriously and offers such a platform.
- **Robustness:** For a project that is designed to work over decades, long-term stability is a real challenge from the very beginning and several measures are taken to assure it. First, we employ strong types widely, e.g. "`ContinuousProcessIndex`" instead of just "`int`", or "`Code code=Electron;`" and not "`int code = 11; //pdg`". This avoids simple mixup and type mistakes. Second, we use compile-time syntax (statically typed units) to express physics properties, for example "`HEPEnergyType E0 = 10_GeV;`" and not "`double E0 = 10; //GeV`", or also "`Vector<HEPMomentumType>`" [11]. This way the compiler makes the first and strongest check on the correctness of physics expressions – and not the end-user in tedious reverse engineering. For physics units we use a template-based solution that can also make compile-time unit calculations. Third, unit testing is mandatory. At the moment of writing our code coverage is 94.4 % with a clear target value of 100 %. Fourth, physics algorithm validation is needed on top of unit testing. Models must be evaluated against other models, parameter changes and most importantly *data*.

This is still work to be done but is a fundamental requirement for all physics algorithms in CORSIKA 8.

- **Performance:** The variety of algorithms possibly be used by CORSIKA 8 and their configuration do not allow for a universal way to benefit from hardware acceleration. However, the framework should support efficient vectorization as well as being cache-friendly, and in addition should allow users to easily deploy and integrate specific hardware accelerators like GPUs, possibly even FPGAs. Besides this, all classes for storage and computation must be written in a thread-safe manner to allow running in parallel context. Besides hardware aspects of performance optimization, also novel routes in algorithms and software technology are fully considered: this includes statistical thinning methods [12], numerical cascade equations, and also the potential of artificial neural networks.

The CORSIKA 8 framework was first designed in a standard source-code/header-file C++ design with a full CMake build system. It was realized in 2020 that due to its extensive use of templates and meta-programming, a header-only setup with a minimal CMake system is feasible and can have beneficial properties and a major refactoring was conducted to achieve this goal. The header-only approach was found to fit the extensive use of templates and meta-programming much better. Only the external modules and the unit tests require compilation. For all applications of the framework, including the delivered examples, a standard exported CMake config-file setup is used. Since the CMake targets are automatically generated and can be imported with a simple `find_package(corsika REQUIRED)` the complexity for writing, maintaining and also using the build system is greatly reduced.

As part of this refactoring we also moved from a standard system-package dependency management to a conan [13] setup. Conan is automatically run by CMake and handles all dependencies without further user intervention in a system-agnostic way. Of course, full guarantee for success and expected results can only be given for the systems actually tested on the continuous integration. For this reason we run unit-tests and examples for a minimum of two typical compilers (gcc and clang) for each commit via gitlab-ci. On CI we use docker containers to provide different, well-defined environments. If further requirements on system stability arise, this system can be easily extended with new hardware (gitlab Runners) and new configuration (docker containers).

In table 1 some characteristics of the code base are compared pre- and post-refactoring. The functionality of the code is identical in both versions. The CMake build system was reduced by almost 27 %, the C++ code increased in size by 17 % (8 %) when counting lines (words). However, the words per line of C++ code was reduced by 9 % reflecting the fact of a simplified more readable code. Furthermore, parts of the increase of the code base are due to extended documentation which could not be disentangled here. The user experience is clearly improved post-refactor since structure is 100 % consistent, namespaces were reduced to (almost) two: `corsika` and `corsika/detail` where the latter is not of concern for users/physicists. There are a few extra namespaces mainly in the modules to distinguish different physics models from each other. The build time of the post-refactor framework, without the external modules, is 37 % faster.

4 Parallelization and HPC-readiness

Different aspects of CORSIKA 8 may benefit in various ways from available optimization strategies. There is no general one-fits-all solution. A central aspect will remain the single-core performance and the related intelligent use of SIMD [14] and caching, which is of paramount importance for minimizing the resources needed to solve a fixed-size problem.

Table 1. Comparison of code base pre- and post-refactoring. While pre-refactoring is a standard source/header-file layout, the post-refactoring version is header only with much improved coding guidelines as well as more uniform documentation. Another major improvement is the use of spdlog.

	pre refactoring	post refactoring
lines of C++	27 931	32 759
words of C++	95 196	103 283
lines of CMake	5368	3910
build time (w/o external modules)	$\approx$ 5 min 30 s	$\approx$ 3 min 30 s

For example the massive production of huge shower libraries for experimental collaborations depends on this.

Some aspect of the computing, for example in the generation of radio and Cherenkov photon emissions, have high potential for GPU acceleration [15] and possibly FPGA optimization. It will remain an interesting research question to what extent such hardware accelerators can also be used for other aspects of the simulations.

However, at the highest energies and at ultimate precision, the mere computing power required to solve even simple questions grows prohibitively fast. Only with a fully multi-threaded and multi-node approach such problems can be solved on existing hardware in reasonable time.

5 Storage of particles in memory, flow of data

In a particle cascade code the typical working principle is to convert the total energy in the initial state into the mass/momentum of low energy particles mostly below their critical energy. Thus, the number of secondaries can become extremely large at high energies. In general, the number of particles that need to be stored in memory at the same time can be kept minimal if always the lowest energy particle is processed until it drops out of the simulation. For these reasons, in CORSIKA 8 particles are stored on a last-in-first-out stack, leading to a depth-first traversal of the shower.

The memory layout of the stack is arbitrary, and is at the moment by default arranged as structure-of-arrays (columns). It can be extended flexibly by adding new property columns. Also array-of-structs (row) storage are possible, or even a mixture. However, the default CORSIKA 8 C++ stack interface can easily be adapted to read out data stored in FORTRAN common blocks, which is in fact used to communicate with the FORTRAN event generators (e.g. SIBYLL, QGSJet, EPOS, etc.). This allows for a very uniform programming, regardless of whether particles are stored in C++ (CORSIKA 8) or in fortran (event generators).

The access to particle data is performed with iterators in a way compatible with standard C++17 standard library containers. Since the original data is stored in columns the iterator merely functions as a reference and as an interface to access the data inside the stack – data stored in the stack is never copied. There is no compact "particle" object representing the properties of one single particle, only the interface iterator to access such data stored on the stack. Copying a particle, copies data on the stack. Deleting a particle, deletes the data on the stack. Etc. This allows to deal with particle properties using a clear interface, but entirely avoiding to copy any data or to allocate any memory. The column-wise storage in memory also has the advantage of being SIMD friendly, if suited access patterns are used.

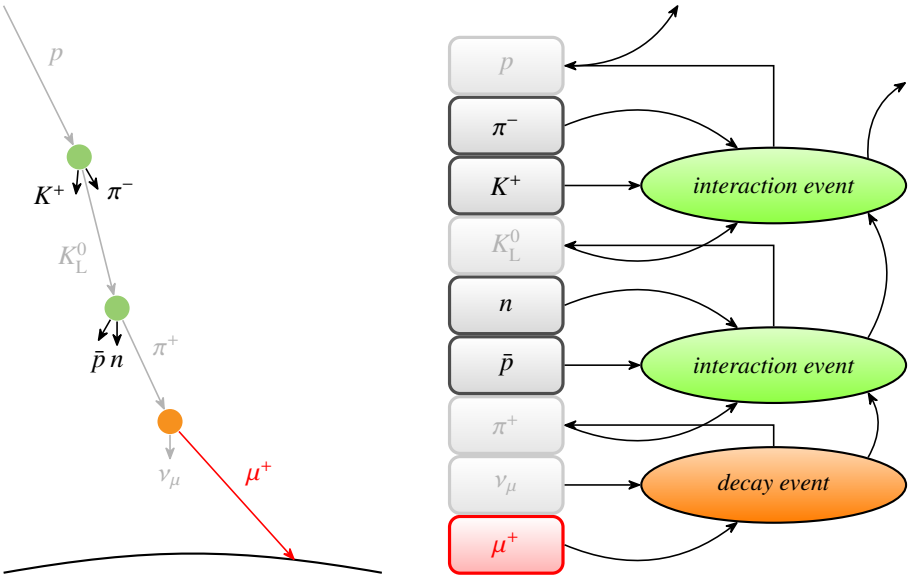

Figure 2. Snapshot of a stack with history information. Red color indicates the particle currently being propagated (muon here). Greyed-out particles represent dead particles which still reside in memory but are not active any more. The muon can still refer to those dead particles since these represent its chain of ancestors.

6 Cascade history

A novel feature for air shower simulations is to retain the entire history of the secondary particle cascade over all generations reaching from the primary particle to the final physics observable. Similar features are available in legacy CORSIKA [16] and also in AIRES [17], however limited to the immediate parent and grandparent generations. This tool has proven itself to be valuable in studying the production of muons in air showers [18]. Here, the muon's parent most often corresponds to a low-energy decaying meson, while the grandparent represents the projectile of the hadronic interaction in which that meson had been produced. Nevertheless, it is desirable to extend this genealogical information further since the only reason why only two preceding generations have been available is of purely technical nature: In the implementation in legacy CORSIKA, a record on the stack is extended to comprise not only the state of a single particle but also the states of its parent and grandparent.

In CORSIKA 8 we follow a different approach. Fundamentally, the particle cascade is equivalent to a tree, with the primary particle being its root. The standard way of simulating these cascades corresponds to a depth-first search with a LIFO stack of particles as data structure. The particle on top gets deleted as soon as it undergoes an interaction (for the sake of simplicity we will only speak of interactions here, although decays are treated in the same way) and newly produced secondaries are pushed onto the stack.

To preserve the history we modify this procedure: When undergoing an interaction the projectile particle is not immediately removed from the stack but only marked as *dead* before the secondaries get pushed onto the stack. It is only factually removed as soon as the subcascade it initiates is completely processed, which is exactly the case when the dead particle is the top particle of the stack for the second time – of course prohibiting any reordering on the stack.

In order to link particles with their parent generation, we introduce *event* objects and the particle object hold shared pointers to their event of creation, which in the case of the primary particle is just a null pointer. The event objects themselves keep a reference to the (dead) projectile to establish the link to the parent particle. Furthermore each event maintains a copy of its secondaries that is filled at the time of their production and not modified afterwards. A particle on the stack is modified during propagation as its position and 4-momentum change. By keeping a constant copy, the state at production is preserved and allows for relating this state to the state at reinteraction. It is foreseen that the event class can be subclassed so that specific types of events can store additional information, e.g. hadronic events store data like impact parameter or number of wounded nucleons, that can be correlated with particle distributions at ground. An illustration of the design is given in fig. 2. It is relevant to note that dead particles are physically deleted from the stack if no further descendant exists. In this example here, when the muon is absorbed at ground it has no descendants by itself and is removed from the stack; at this moment also the following neutrino has no further active descendant and will be removed; the same for the subsequent pion.

Theoretical considerations and first experience show that the stack size in this history approach typically grows only by less than a factor of two compared to the standard approach, which does not pose any restrictions given the anyway small memory footprint of the stack.

7 Output generation and output format

One design requirements for the output data format are high-performance writing since particle output typically exceeds available memory. In most scenarios the data writing in CORSIKA 8 is no bottleneck since computing time in the physics part is significant and output is relatively simple. However, we must consider that in some cases output file of many GB up to TB can be generated. So both, writing performance and also compression are relevant parameters.

We also note that the typical workflow for air shower simulations is "simulate once, read often". Big Monte Carlo libraries are often generated by experimental collaboration at significant cost and are then used for many different purposes. Again, storage compression is important and reading performance at least as important as writing performance.

Due to the differing characteristics of different properties of particle data, it is an advantage to chose a column-wise storage also on disk. This will support better compression and higher performance.

It is also interesting to note that there are two orthogonal extreme scenarios for the output of CORSIKA 8: First, at ultra-high energies there will be huge event sizes with TB of data. Second, at <TeV energies where gamma-ray observatories are operating, the particle content per event becomes negligible (few particles per event, with Poisson fluctuations). Here a typical output scenarios is a library of million of events in a single run. It was found in our tests that e.g. HDF5 perform excellent in the former case, but struggles in the latter. ROOT files also perform quite well, but have disadvantages mostly in form of a visible overhead when saving a huge number of very small events. Furthermore, parquet files can also be used within ROOT.

We performed a comprehensive study of different output options in realistic conditions. Factoring in the fact that multi-language, multi-platform support is very much desired and simplicity is favoured, we decided to store output in a structured way using the normal filesystem, where we create a simple directory structure populated with data files. For data where the I/O performance is irrelevant we save human readable yaml files, for the other data we chose parquet [19] as binary column-wise solution.

CORSIKA 8 also ships a Python data analysis library to provide a suite of high-level analysis toolkit. The standard analysis workflow in CORSIKA 8 is to produce data during runtime. These data are either directly saved to disk, or histogrammed, and then saved to disk. CORSIKA 8 makes use of the `Boost.Histogram` [20] library to accumulate data during runtime like for example the number of interactions depending on particle species and energy. In `Boost.Histogram` histograms of any dimension and with a wide variety of axis types can be created. As `Boost.Histogram` does not offer any built-in means of saving histograms to disk, we have developed a simple custom data format based on NumPy [21] npz-files that allows to serialize basic histograms.

The high-level analysis is best done by reading and processing the output in Python.

8 Physics modules

The physics content of CORSIKA 8 is not yet complete but is steadily extended and already useful for some studies. Currently available event generators for hadron collisions are SIBYLL2.3d [22], QGSJetII.04 [23], PYTHIA 8 [24] and UrQMD [25]. The latter is used at "low" energies (typically below around 100 GeV), while the former two can go up to the highest energies. The electromagnetic interactions and continuous/radiative energy losses of leptons are simulated with PROPOSAL [26–28]. Decays are treated with PYTHIA 8 or SIBYLL. Particles are tracked in magnetic fields on curved trajectories using a leap-frog algorithm. The geometry of the environment and media of the tracking is defined in a custom geometry/media package. The purpose of the geometry package is to allow the definition of different materials on large scales, e.g. showers in the atmosphere which then penetrate the ice shield at the south pole. The geometry package is not designed to support a microscopic description of, for example, a detector like in GEANT4 [29]. Nevertheless, a comparison with, or alternative use of existing tracking codes, as e.g. VecGeom [30], is possible and planned in the future.

Further physics models are added continuously as they become needed, and as quickly as manpower allows.

9 First results, examples

While the full potential has not yet been reached, a wide range of studies can already be done. This includes the first validation studies but provides also insight new types of research that was not feasible before.

In particular, physics modules for the hadronic and muonic shower components are available and allow us to perform cross-validation with other air shower simulation codes. In fig. 3 we compare the energy spectra of secondary particles reaching ground as obtained from simulations with CORSIKA 8, CORSIKA 7.74 and CONEX [31] using the same parameters and interaction models in all three codes (vertical proton primary of 10^{17} eV, SIBYLL 2.3d and UrQMD 1.3 for hadronic interactions above and below 63.1 GeV, respectively). We observe good agreement in the high energy regime. In the low energy regime we find significant differences in the spectra of nucleons, which we attribute to differences in the interfaces to the event generator. The impact on the muon spectra, however, is small. First detailed results of such comparisons have been published elsewhere [32] and further studies are ongoing.

As a first application using the cascade history described in section 6 we display the ancestry of projectiles that finally lead to a muon on ground in fig. 4. At energies between the primary energy (10^{18} eV) and down to about a decade less, nucleon interactions dominate, with only a small fraction being antinucleons. Especially the impact of the primary proton

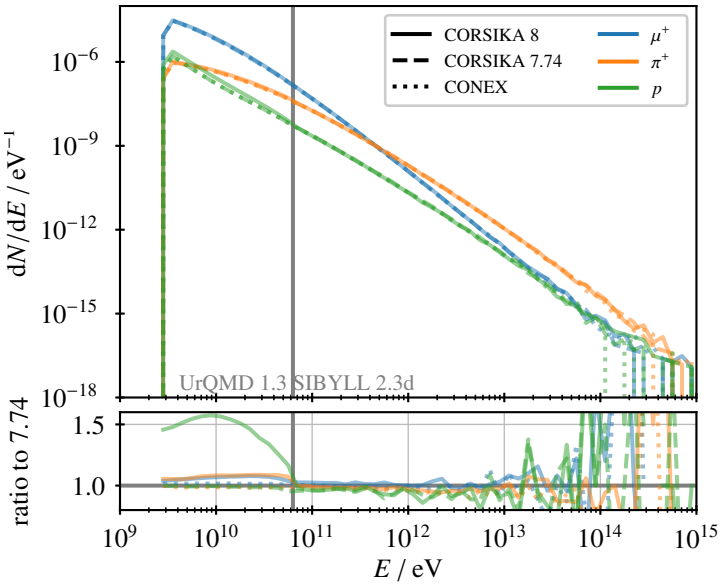

Figure 3. Energy spectra of μ^+, π^+ and p of 1×10^{17} eV vertical proton shower, averaged over 400 events

is visible as it is the unique particle in the "zeroth generation" of all shower particles and therefore contributes 1 to $\langle N_{\mathrm{gen}} \rangle$ in the highest energy bin. At lower energies ($\lesssim 10^{17}$ eV), pion interaction are the main driver of the hadronic cascade and each decade in energy contributes roughly equally to the number of generations until around several 10 GeV they decay into muons and almost no pion interacts again below 10 GeV.

10 Summary and roadmap

The novel CORSIKA 8 framework is designed as a modern high-performance computing project with modularity, completeness, robustness, and performance in mind. As an open community project the goal is to provide the best possible environment for all simulations of secondary particle cascades in astroparticle physics and beyond. Ideally, all state-of-the art physics models and algorithms are implemented inside CORSIKA 8 to serve as a platform for the exchange of modular algorithms for the community.

The framework is written in C++17 with potential future updates also in C++20/23. The software will support to run suited sub-problems on hardware accelerators, and will make intelligent use of modern and future computer hardware. Modern software engineering techniques are employed. For example conan for packet management, and python bindings for data analysis. The lessons learned from a recent code base refactor to a header-only approach have been presented.

A first pre-release version is available for tests and full validation [33]. It is the goal to release a version of the framework within the year 2021 that has a complete interface design for all typical applications. This includes physics models, geometry, environment/media, output and configuration. Such a version can be used by the community for more extensive tests also on larger production scale. With the feedback collected during this period, a first full physics production release will emerge on a short timescale. It is foreseen that the setup

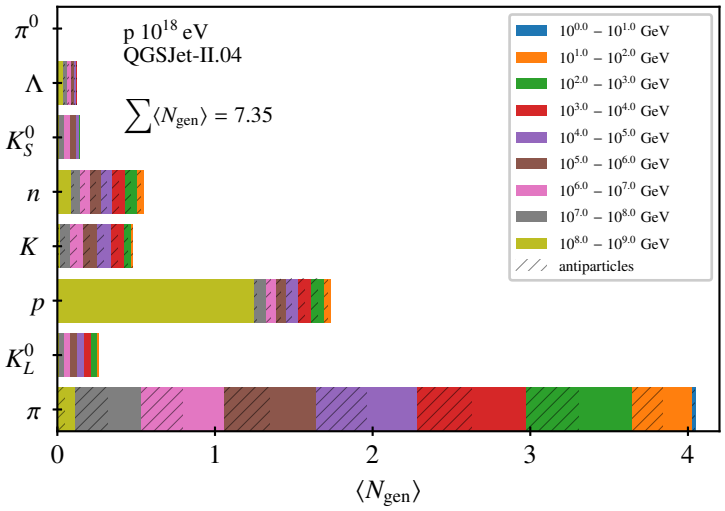

Figure 4. Average number of muon ancestor projectile generations by particle species and energy.

as an open-source community project will support the goal for continued development over a very extended period of time.

Acknowledgments

The authors acknowledge support by the High Performance and Cloud Computing Group at the Zentrum für Datenverarbeitung of the University of Tübingen, the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no. INST 37/935-1 FUGG.

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).
[2] H. J. Gils, D. Heck, J. Oehlschlaeger, G. Schatz, T. Thouw, and A. Merkel, Comput. Phys. Commun. **56**, 105 (1989).
[3] R. Engel, D. Heck, T. Huege, T. Pierog, M. Reininghaus, F. Riehn, R. Ulrich, M. Unger, and D. Veberič, Comput. Softw. Big Sci. **3**, 2 (2019), arXiv:1808.08226 [astro-ph.IM] .
[4] J. Hoberock *et al.*, *A Unified Executors Proposal for C++*, Tech. Rep. P0443R11 (2019).
[5] G. Dos Reis, M. Hall, and G. Nishano, *A Module System for C++ (Revision 2)*, Tech. Rep. N4214 (2014).
[6] A. B. Migdal, Phys. Rev. **103**, 1811 (1956).
[7] L. D. Landau and I. Pomeranchuk, Dokl. Akad. Nauk Ser. Fiz. **92**, 735 (1953).
[8] L. D. Landau and I. Pomeranchuk, Dokl. Akad. Nauk Ser. Fiz. **92**, 535 (1953).
[9] R. Smida *et al.*, EPJ Web Conf. **53**, 08010 (2013).

[10] M. Reininghaus and R. Ulrich, EPJ Web Conf. **210**, 02011 (2019), arXiv:1902.02822 [astro-ph.IM] .
[11] H. P. Dembinski, L. Nellen, M. Reininghaus, and R. Ulrich (CORSIKA 8), PoS **ICRC2019**, 236 (2020).
[12] A. M. Hillas, in *Proc. 17th Int. Cosmic Ray Conf.* (1981) p. 193.
[13] https://conan.io/.
[14] L. Arrabito, K. Bernlöhr, J. Bregeon, M. Carrère, A. Khattabi, P. Langlois, D. Parello, and G. Revy, Comput. Softw. Big Sci. **4**, 9 (2020), arXiv:2006.14927 [astro-ph.IM] .
[15] D. Baack and W. Rhode, J. Phys. Conf. Ser. **1690**, 012073 (2020).
[16] D. Heck and R. Engel, *The EHISTORY Option of the Air-Shower Simulation Program CORSIKA*, Tech. Rep. FZKA-7495 (Forschungszentrum Karlsruhe, 2009).
[17] S. J. Sciutto, (1999), 10.13140/RG.2.2.12566.40002, arXiv:astro-ph/9911331 .
[18] C. Meurer, J. Bluemer, R. Engel, A. Haungs, and M. Roth, Czech. J. Phys. **56**, A211 (2006), arXiv:astro-ph/0512536 [astro-ph] .
[19] https://arrow.apache.org/docs/cpp/parquet.html.
[20] H. P. Dembinski, J. Pivarski, and H. Schreiner, EPJ Web Conf. **245**, 05014 (2020).
[21] C. R. Harris *et al.*, Nature **585**, 357 (2020), arXiv:2006.10256 [cs.MS] .
[22] F. Riehn, R. Engel, A. Fedynitch, T. K. Gaisser, and T. Stanev, Phys. Rev. D **102**, 063002 (2020), arXiv:1912.03300 [hep-ph] .
[23] S. Ostapchenko, Phys. Rev. D **83**, 014018 (2011), arXiv:1010.1869 [hep-ph] .
[24] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, Comput. Phys. Commun. **191**, 159 (2015), arXiv:1410.3012 [hep-ph] .
[25] M. Bleicher *et al.*, J. Phys. G **25**, 1859 (1999), arXiv:hep-ph/9909407 .
[26] J.-M. Alameddine, J. Soedingrekso, A. Sandrock, M. Sackel, and W. Rhode, J. Phys. Conf. Ser. **1690**, 012021 (2020).
[27] M. Dunsch, J. Soedingrekso, A. Sandrock, M. Meier, T. Menne, and W. Rhode, Comput. Phys. Commun. **242**, 132 (2019), arXiv:1809.07740 [hep-ph] .
[28] J. H. Koehne, K. Frantzen, M. Schmitz, T. Fuchs, W. Rhode, D. Chirkin, and J. Becker Tjus, Comput. Phys. Commun. **184**, 2070 (2013).
[29] S. Agostinelli *et al.* (GEANT4), Nucl. Instrum. Meth. A **506**, 250 (2003).
[30] S. Wenzel, J. Apostolakis, and G. Cosmo, EPJ Web Conf. **245**, 02024 (2020).
[31] T. Bergmann, R. Engel, D. Heck, N. N. Kalmykov, S. Ostapchenko, T. Pierog, T. Thouw, and K. Werner, Astropart. Phys. **26**, 420 (2007), arXiv:astro-ph/0606564 [astro-ph] .
[32] D. Melo, M. Reininghaus, F. Riehn, and R. Ulrich (CORSIKA 8), PoS **ICRC2019**, 399 (2020).
[33] https://gitlab.ikp.kit.edu/AirShowerPhysics/corsika.