PROCEEDINGS OF SCIENCE

Parallel processing of radio signals and detector arrays in CORSIKA 8

A. Augusto Alves Jr,[a,*] **Nikolaos Karastathis**[a] **and Tim Huege**[a,b] **for the CORSIKA 8 collaboration**

[a] *Institute for Astroparticle Physics (IAP), Karlsruhe Institute of Technology, Karlsruhe, Germany*
[b] *Astrophysical Institute, Vrije Universiteit Brussel, Belgium*

E-mail: aalvesju@gmail.com

This contribution describes some recent advances in the parallelization of the generation and processing of radio signals emitted by particle showers in CORSIKA 8. CORSIKA 8 is a Monte Carlo simulation framework for modeling ultra-high energy particle cascades in astroparticle physics. The aspects associated with the generation and processing of radio signals in antennas arrays are reviewed, focusing on the key design opportunities and constraints for deployment of multiple threads on such calculations. The audience is also introduced to Gyges, a lightweight, header-only and flexible multithread self-adaptive scheduler written compliant with C++17 and C++20, which is used to distribute and manage the worker computer threads during the parallel calculations. Finally, performance and scalability measurements are provided and the integration into CORSIKA 8 is commented.

38th International Cosmic Ray Conference (ICRC2023)
26 July - 3 August, 2023
Nagoya, Japan

ICRC2023
38th International Cosmic Ray Conference
The Astroparticle Physics Conference

*Speaker

PoS(ICRC2023)469

1. Introduction

Over the past couple of decades of research on extensive particle showers, radio detection has become a technique competitive with standard particle and fluorescence driven measurements. Due to the complexity of extensive particle showers, in air and other media, detailed particle-level simulations of the radio emissions are often needed to analyze experimental data and reconstruct the properties of the primary particles.

In this context, the two standard software tools used for radio emission simulations are CoREAS [1] as implemented in CORSIKA 7 and ZHAireS [2]. These tools implement two different formalisms for calculating the radio emission from the particle tracks in the extensive particle shower, namely the "Endpoint" [3, 4] and the "ZHS" [5] formalisms, respectively. Both algorithms have been recently implemented on CORSIKA 8 [6], which is a modern C++17 compiliant Monte Carlo simulation framework for modeling ultra-high energy particle cascades in astroparticle physics.

Additionally, proposed next-generation experiments with growing array size and channel-count pose significant challenges regarding the computational cost for calculating radio emissions, especially for ultra high-energy showers and signals propagating in media with varying properties. In order to mitigate such impacts, the radio emission module for the CORSIKA 8 (C8) framework [7] has been reimplemented in multithread friendly fashion. This contribution discusses these developments and is organized as the following. section 2 gives an overview of the radio module of CORSIKA 8. section 3, Gyges, a C++17/20 library for distribution and management of tasks on multithread systems, is presented. section 4 the parallelization strategy used of the radio module calculations is covered, including the corresponding updates on the interfaces and codes implementing the algorithms. Finally, section 5 presents the performance gains in function of the number of threads for both formalisms, measured for array detectors with different sizes. section 6, the conclusions and perspectives are drawn.

2. Overview of the radio module in CORSIKA 8

The top-level architecture of the radio process module is shown in Figure 1. All components in the module can be independently configured and combined with either the CORSIKA 8 built-in interface or custom C++ code, making possible construct multiple radio process instances for different scenarios. The components of the modules have been extensively presented in [7, 8]. In this contribution, the flow of the radio calculation is discussed and how performance is enhanced using multithreading.

Once the radio process has received a particle track, the track is being checked according to the *track filter* in order to be determined if this track is relevant for the radio calculation or not. The track is then pushed forward to the *formalism* and the track needs to be looped over all antennas existing in the antenna collection. A significant portion of the calculation happens after this step, which needs to be repeated for every antenna available and for every single particle track provided. This is precisely the part of the code we wish to accelerate with this work. Inside the loop, the particle track is fed to the propagator, which calculates the valid emission paths from the particle to the antenna. Hence, all the necessary information to calculate the electric field vector (or vector potential) is present now, and finally this information is processed and stored in the *antenna* instance.

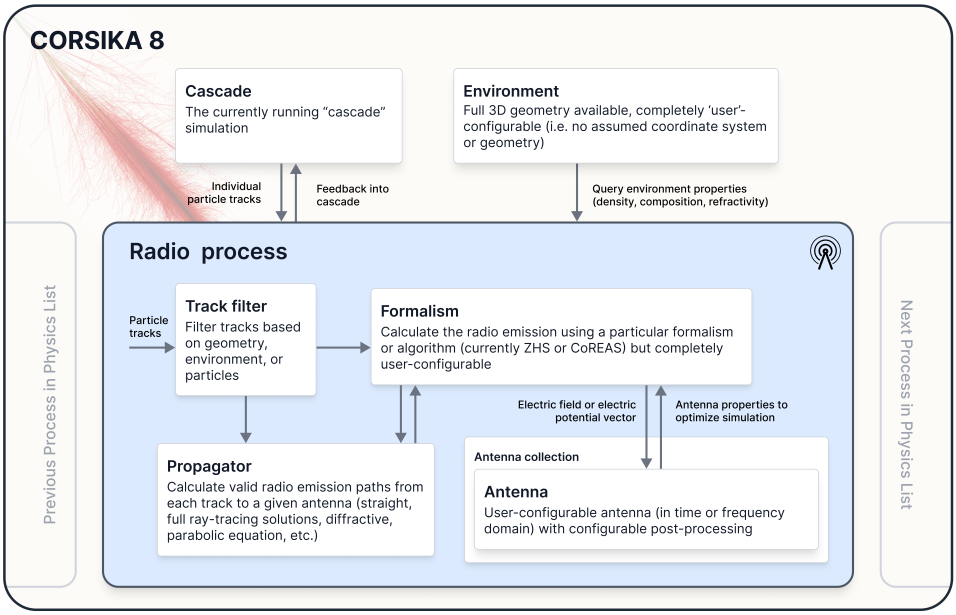

Figure 1: A schematic diagram of the radio process currently implemented in CORSIKA 8 and how it integrates with the CORSIKA 8 framework

The load of this calculation is directly affected by the underlying complexity of the *propagator* used. Naturally, the larger the number of antennas in the detector, the higher the runtime of the radio simulation will be. By assigning different bunches of antennas to available threads, we expect to observe a significant performance boost.

3. Gyges

Gyges is a lightweight C++17, or higher, header-only library to manage thread pooling, which has been developed in the context of the ongoing effort to paralellize the CORSIKA 8 framework. By deploying Gyges, the computational costs associated to creating and destroying a thread-pool, a `gyges::gang` in the library's jargon, can be paid just once in the program lifetime, with threads of the pool picking-up tasks as they become available. If there are no tasks, the threads just go sleeping. Additionally, tasks can be submitted from multiple threads, with the submitter getting a `std::future` object to monitor the task in-place. On the task implementation side, developers get access to a `std::stop_token` that can be used to interrupt the task execution, if a request to do so arrives from `gyges::gang` via the `gyges::gang::stop()`.

As default behavior, once a `gyges::gang` is created, it will promptly pick up and process any submitted task. This behavior can be changed, putting the `gyges::gang` in a "hold-on" state. In that case, the processing of the tasks will be postponed until it is put back on "unhold" status, while the threads will be put to sleep until the `gyges::gang::unhold()` command is sent. Among other features, Gyges provides two implementations of the `gyges::for_each` algorithm, with one of than able to use an already existing `gyges::gang` object.

```cpp
class gang
{
    // constructor taking the
    gang(unsigned int const thread_count=std::thread::hardware_concurrency(), bool release = true ) ;
    gang( gang const & other ) = delete;
    gang( gang && other )      = delete;

    //submit a task implementing void operator( void )
    template<typename FunctionType>
    inline std::future<void> submit_task(FunctionType f) requires gyges::Dispatchable<FunctionType>;
    //notify the running tasks (request stop), interrupt picking up new tasks and destroys the gang
    inline void stop(void);
    //put the gang on ``hold'' status
    inline void hold(void);
    //revert the gang to ``processing'' status
    inline void unhold(void);
    //checks the gang status
    inline bool on_hold(void);
    //get the gang size
    inline std::size_t size(void);
};

// for_each accepting a pre-created gang
template<typename Iterator, typename Predicate>
void for_each(Iterator begin, Iterator end, Predicate const& functor,  gang& pool);

// for_each
template<typename Iterator, typename Predicate>
void for_each(Iterator begin, Iterator end, Predicate const&  functor);
```

Listing 1: Interface of `gyges::gang` and `gyges::for_each` implementations.

Gyges is licensed under GPL version 3 and is currently in a stable release state. The code is available at https://gitlab.iap.kit.edu/AAAlvesJr/Gyges.

4. Radio module parallelization strategy

The radio module calculates the signal corresponding to each particle, and the tracks that describe its trajectory, for each antenna of the array detector, often running as one of the final operations in the particle simulation process sequence. In order to parallelize the radio module, the calculation of the signal over the array detector is processed using a `gyges::gang` containing a specifiable number of threads, in a such way that, for each particle and its tracks, the response of the antennas and the storing of information is calculated in parallel.

Since the signal processing corresponding to a single antenna is not intensive enough to occupy efficiently a thread, each submitted task computes the response corresponding to a bunch of antennas. As it will be detailed in section 5, the number of antennas in this bunch in comparison to the Gyges gang size is a critical parameter for the overall efficiency of the radio module.

This logic is implemented with the introduction of a couple of classes, one per formalism, to encapsulate the pulse calculated for each antenna in a callable object abstracting away the implementation details of CoREAS and ZHAireS. This object is called runner, and it is the one to be distributed, together with the antenna collection that describes the array detector, to the worker threads managed by the `gyges::gang` instance, which is being held by the `corsika::RadioProcess` and has the same life-time of it. These developments are complemented by changes in the user interfaces easing to instate and to deploy the radio module. These changes are summarized in Listing 2.

```
//convenience function for creating a propagator, taking as parameter an environment object
auto propagator = make_simple_radio_propagator(environment);
//convenience functions for creating CoREAS and ZHS instances, taking as parameters detector
//and propagator objects, as well as the number of threads
auto coreas     = make_radio_process_CoREAS(detector, propagator, nthreads);
auto    zhs     = make_radio_process_ZHS(detector, propagator, nthreads);
```

Listing 2: Improved interface of radio module.

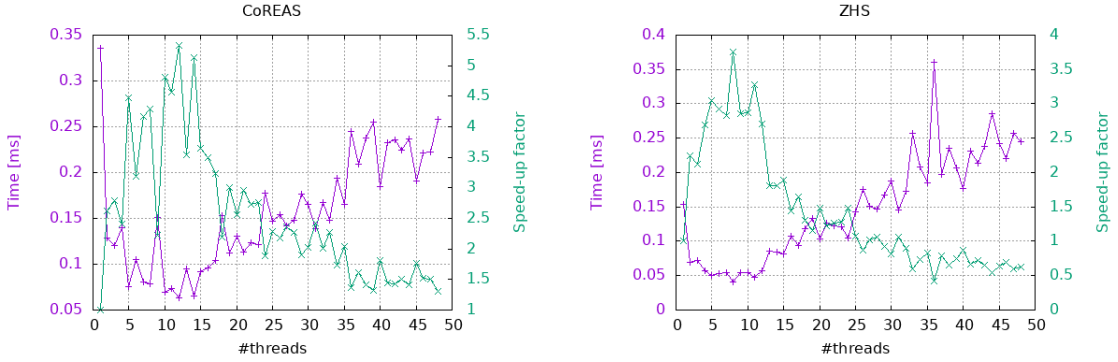

Figure 2: Performance to process a single particle as a function of number of threads for an array detector containing 200 antennas.

CORSIKA 8-wise, the expected overall speed-up depends hugely on the detector size, i.e. number of antennas in the detector. For large array detectors, or computing intensive propagators, the importance of radio module operations grows, tending to dominate the particle simulation sequence. In such situations, the speed-up is larger.

5. Performance measurements and validation

The raw performance gains from parallelization of the radio module calculations over the antennas of the array detector have been assessed measuring the time spent, and the corresponding speed-up, to process the electromagnetic pulse from a single particle as a function of the array detector size and number of threads. Array detectors with different sizes have been tested against `gyges::gang` with up to 48 worker threads. The results are summarized in Figure 2, Figure 3 and Figure 4

Figure 2 shows that for array detectors with 200 antennas, the speed-up peaks between 10 and 15 worker threads, beyond which the performance decreases due to computing tasks not being able to occupy the CPU enough to hide the latency and costs associated to management of multiple threads. As it is shown in Figure 3 and Figure 4, by increasing the number of antennas, the speed-up scales mostly as predicted by Amdahl's law. Similar results would be achieved, albeit leading to performance peaking at different number of threads, when deploying propagators performing heavier calculations.

The overall impact of the parallelization of the radio module on CORSIKA 8 has been measured running a full electromagnetic shower simulation. In that scenario, due to the Gyges design, the

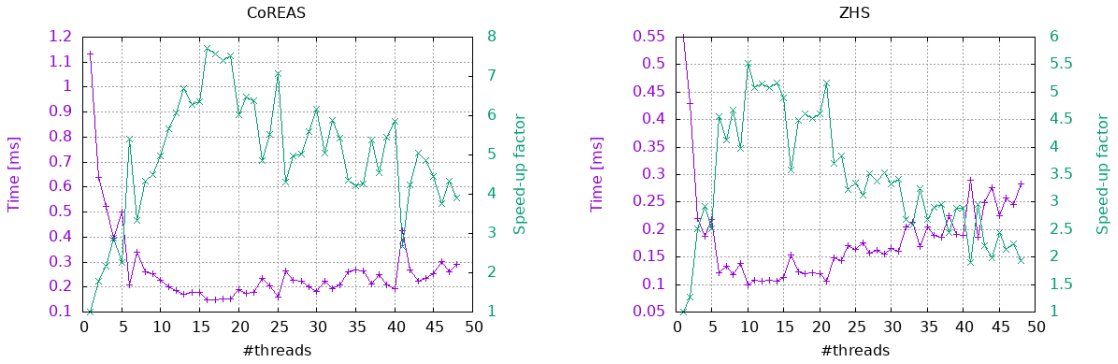

Figure 3: Performance to process a single particle as a function of number of threads for an array detector containing 1000 antennas.

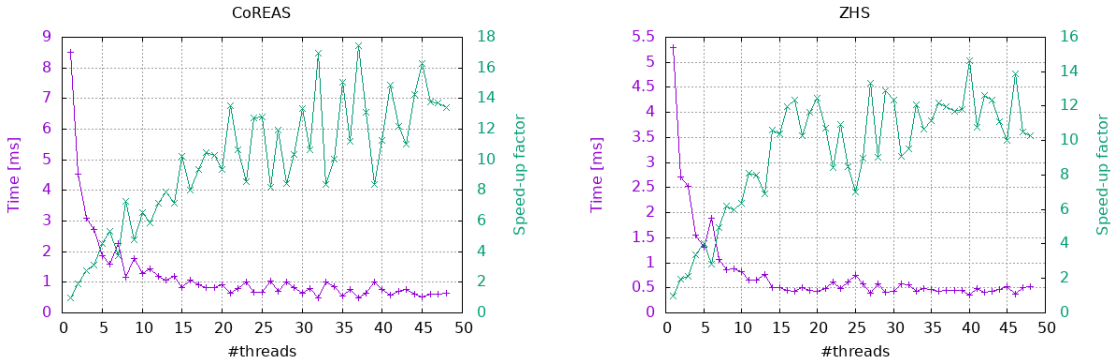

Figure 4: Performance to process a single particle as a function of number of threads for an array detector containing 10,000 antennas.

overhead for creating, managing and submitting tasks to the thread pool is negligible in comparison to the other initialization routines called up-front in the full shower simulation. The radio module is currently the only component of the CORSIKA 8 sequence capable of performing its tasks in parallel, meaning that the maximum speed-up is limited by the amount of code running sequentially, in accordance with Amdhal's law. The total time to run the full shower is limited below by not deploying the radio module at all, and above by running this module in a single thread, that is sequentially. Figure 5 summarizes the results and confirms the profiles performed for measuring the single particle performance. In the same figure, we show for reference the runtime of the same electron induced shower with the radio emission calculation turned off.

Finally, the numerical consistence of the predictions for each algorithm has been checked for different numbers of threads. Figure 6 and Figure 7 show that there is no measurable impact of the parallelism in numerical results provided by each algorithm. The signal pulses simulated with both formalisms are identical regardless the number of threads, which confirms that the physics calculations are done consistently and accurately.

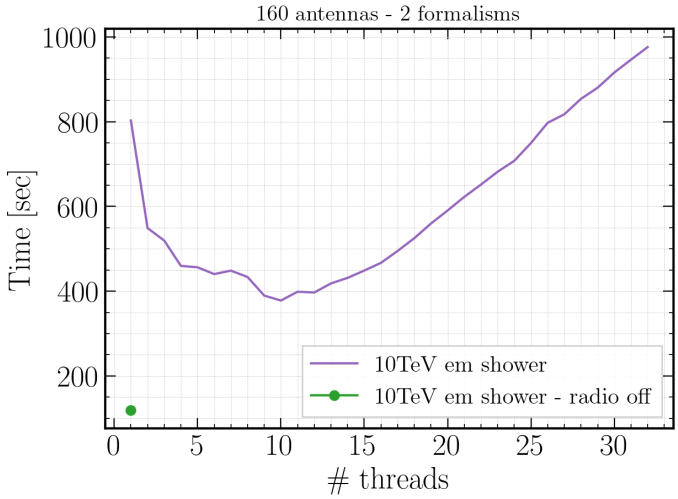

Figure 5: The parallelized radio module running on an electron induced air shower processing a detector array of 160 antennas. 2 formalism, namely CoREAS and ZHS are activated and use 160 antennas each. The performance peaks at 10 worker threads, beyond which the performance degradates.

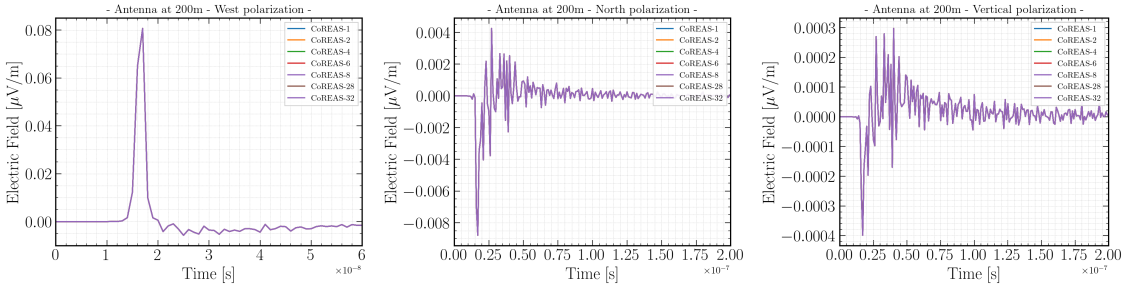

Figure 6: Pulse comparisons in all three polarizations using the CoREAS formalism. The pulses have been simulated and processed on `gyges::gangs` of different sizes. Different number of threads produce identical pulses for CoREAS, as expected.

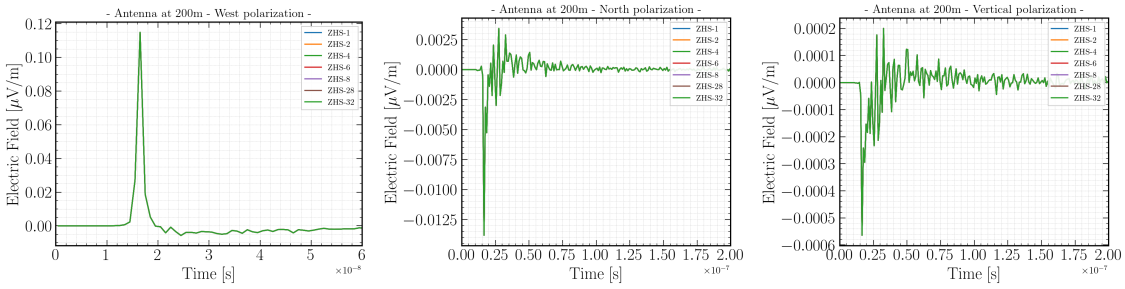

Figure 7: Pulse comparisons in all three polarizations using the ZHS formalism. The pulses have been simulated and processed on `gyges::gangs` of different sizes. Different number of threads produce identical pulses for ZHS, as expected.

6. Conclusions

The status of the effort to parallelize the calculations of the radio module implemented in CORSIKA 8 has been summarized. The implementation of the multithread dispatching mechanisms and management, which is based in Gyges, is compliant with C++17 or higher standard and allows specifying the number of worker threads without impacting any numerical result. The optimal number of threads, in which the performance peaks, depends on of the size of the antenna array. Under favorable, the performance gains are significant, with speeding-up reaching a factor 10 or superior. The code is currently under final internal review and should be integrated into CORSIKA 8 main branch in near future.

References

[1] Huege T, Ludwig M and James C W 2013 *AIP Conference Proceedings* **1535** 128–132 ISSN 0094-243X (*Preprint* https://pubs.aip.org/aip/acp/article-pdf/1535/1/128/11832917/128_1_online.pdf) URL https://doi.org/10.1063/1.4807534

[2] Alvarez-Muñiz J, Romero-Wolf A and Zas E 2010 *Phys. Rev. D* **81**(12) 123009 URL https://link.aps.org/doi/10.1103/PhysRevD.81.123009

[3] James C W, Falcke H, Huege T and Ludwig M 2011 *Phys. Rev. E* **84**(5) 056602 URL https://link.aps.org/doi/10.1103/PhysRevE.84.056602

[4] Ludwig M and Huege T 2011 *Astroparticle Physics* **34** 438–446 ISSN 0927-6505 URL https://www.sciencedirect.com/science/article/pii/S0927650510002094

[5] Zas E, Halzen F and Stanev T 1992 *Phys. Rev. D* **45**(1) 362–376 URL https://link.aps.org/doi/10.1103/PhysRevD.45.362

[6] Engel R, Heck D, Huege T, Pierog T, Reininghaus M, Riehn F, Ulrich R, Unger M and Veberič D 2018 *Computing and Software for Big Science* **3** URL https://doi.org/10.1007%2Fs41781-018-0013-0

[7] Karastathis N, Prechelt R, Huege T and Ammerman-Yebra J 2021 *PoS* **ICRC2021** 427

[8] Karastathis N, Prechelt R, Ammerman-Yebra J and Huege T 2023 *PoS* **ARENA2022** 050

PoS(ICRC2023)469

The CORSIKA 8 Collaboration

J.M. Alameddine[1], J. Albrecht[1], J. Alvarez-Muñiz[2], J. Ammerman-Yebra[2], L. Arrabito[3], J. Augscheller[4], A.A. Alves Jr.[4], D. Baack[1], K. Bernlöhr[5], M. Bleicher[6], A. Coleman[7], H. Dembinski[1], D. Elsässer[1], R. Engel[4], A. Ferrari[4], C. Gaudu[8], C. Glaser[7], D. Heck[4], F. Hu[9], T. Huege[4,10], K.H. Kampert[8], N. Karastathis[4], U.A. Latif[11], H. Mei[12], L. Nellen[13], T. Pierog[4], R. Prechelt[14], M. Reininghaus[15], W. Rhode[1], F. Riehn[16,2], M. Sackel[1], P. Sala[17], P. Sampathkumar[4], A. Sandrock[8], J. Soedingrekso[1], R. Ulrich[4], D. Xu[12], E. Zas[2]

[1] Technische Universität Dortmund (TU), Department of Physics, Dortmund, Germany
[2] Universidade de Santiago de Compostela, Instituto Galego de Física de Altas Enerxías (IGFAE), Santiago de Compostela, Spain
[3] Laboratoire Univers et Particules de Montpellier, Université de Montpellier, Montpellier, France
[4] Karlsruhe Institute of Technology (KIT), Institute for Astroparticle Physics (IAP), Karlsruhe, Germany
[5] Max Planck Institute for Nuclear Physics (MPIK), Heidelberg, Germany
[6] Goethe-Universität Frankfurt am Main, Institut für Theoretische Physik, Frankfurt am Main, Germany
[7] Uppsala University, Department of Physics and Astronomy, Uppsala, Sweden
[8] Bergische Universität Wuppertal, Department of Physics, Wuppertal, Germany
[9] Peking University (PKU), School of Physics, Beijing, China
[10] Vrije Universiteit Brussel, Astrophysical Institute, Brussels, Belgium
[11] Vrije Universiteit Brussel, Dienst ELEM, Inter-University Institute for High Energies (IIHE), Brussels, Belgium
[12] Tsung-Dao Lee Institute (TDLI), Shanghai Jiao Tong University, Shanghai, China
[13] Universidad Nacional Autónoma de México (UNAM), Instituto de Ciencias Nucleares, México, D.F., México
[14] University of Hawai'i at Manoa, Department of Physics and Astronomy, Honolulu, USA
[15] Karlsruhe Institute of Technology (KIT), Institute of Experimental Particle Physics (ETP), Karlsruhe, Germany
[16] Laboratório de Instrumentação e Física Experimental de Partículas (LIP), Lisboa, Portugal
[17] Fluka collaboration

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Projektnummer 445154105. For the simulations presented, computing resources from KIT have been used.