JuliaHEP 2024 Workshop

Monday 30 September 2024 - Friday 4 October 2024

CERN

Book of Abstracts

Contents

Talks / 1

Using Julia to perform Physics on time critical systems

Author: Evangelos Paradas[1]

[1] *ASML*

Corresponding Author: evangelos.paradas@proton.me

In High Energy Physics, very demanding algorithms are written to represent the physics that takes place either in the accelerators or in the detectors and to analyze the measured signals. The cost of each algorithm can be broken down into two categories: the development and the execution. Starting from the latter, the target platforms set stringent runtime constraints, where the algorithm must behave in a very predicable way, with no anomalous behaviour under expected conditions, including rare occurrences. Because of these needs, HEP has favoured writing code in C++ and, after validation, deploying this directly to target systems.
This choice has a profound impact, adding yet another complexity layer to the development phase. Physicists (and mathematicians), often with limited software background, must learn a complex programming language with significant hurdles, such as raw pointers and memory alignment, "copy by value" vs. "copy by reference" and many other topics that require significant study.
This is exactly the point that High Level programming languages are trying to solve, by bringing expressiveness, modularity, speed and reproducibility to the code development process. At this point, development and execution seem to be pretty far away, for the use-cases that we are interested in. What if Julia could make it possible to develop in an easy way and execute in a deterministic way?
In this talk we will go through the aspects of programming using Julia, the ease of handling the Garbage Collector and the ability to interoperate with C/C++, resulting in smooth deployment and significantly reduced maintenance.

Talks / 2

Enabling Julia code to run at scale with artefact caching

Author: Elvis Alexander Aguero Vera[1]

Co-authors: Pere Mato Vila [2]; Stewart Graeme

[1] *Brown University*

[2] *CERN*

Corresponding Authors: elvis_vera@brown.edu, graeme.stewart@cern.ch, pere.mato@cern.ch

The Julia programming language has evolved into a mature tool for scientific computing over the past decade, offering high-level capabilities with just-in-time (JIT) compilation and efficient garbage collection. Its performance, comparable to that of C/C++, makes Julia an attractive option for the high-energy physics community. However, Julia's use of precompiled files to achieve this performance introduces significant startup delays on the first program execution ("time to first plot"), which would be a severe drawback in distributed systems where each node would have to compile dependencies locally. This project proposes a workflow to obtain a ready-to-use directory with all necessary precompiled files for selected applications, while also leveraging the shared CernVM-FS (CVMFS) file system to share files across nodes in a distributed setup. We developed and tested a framework for automating the publication of precompiled Julia files to CVMFS and evaluated its impact on performance using two sample applications: the Julia Jet Reconstruction package and the Geant4 wrapper package. Our findings demonstrate that caching precompiled files that are stored in CVMFS significantly reduces startup times, with reductions of up to 97% for the Geant4 package. Furthermore, we examined the effects of cross-compilation for various microarchitectures and found that nodes benefit from shared cache files without notable performance degradation due to microarchitecture differences.

Talks / **3**

Simulation for the Tau Air-Shower Mountain-Based Observatory

Authors: Jeffrey Lazar[1]; Pavel Zhelnin[None]

[1] *University of Wisconsin-Madison*

Corresponding Authors: jlazar@icecube.wisc.edu, pzhelnin@g.harvard.edu

While IceCube's detection astrophysical neutrinos at energies up to a few PeV has opened a new window to our Universe, much remains to be discovered regarding these neutrinos'origin and nature. In particular, the difficulty differentiating ⊠⊠ and ⊠⊠ charged-current (CC) events in the energy limits our ability to measure this flux's flavor ratio precisely. The Tau Air-Shower Mountain-Based Observatory (TAMBO) is a next-generation neutrino observatory capable of producing a high-purity sample of ⊠⊠ CC events in the energy range from 1-100 PeV, i.e. just above the IceCube measurements. An array of water Cherenkov tanks and plastic scintillators deployed on one face of the Colca Canyon will observe the air shower produced when a ⊠ lepton, produced in a ⊠⊠ CC interaction, emerges from the opposite face and decays in the air. In this contribution, I will present the current status of the TAMBO simulation, including preliminary sensitivities to various flux models.

Talks / **4**

Demonstrator for HEP event-processing framework in Julia

Authors: Benedikt Hegner[1]; Josh Ott[2]; Mateusz Jakub Fila[1]; Oleksandr Shchur[3]

[1] *CERN*

[2] *North Carolina State University*

[3] *Ukrainian Catholic University (UA)*

Corresponding Authors: mateusz.jakub.fila@cern.ch, joshua.kennith.ott@cern.ch, benedikt.hegner@cern.ch, oleksandr.shchur@cern.ch

Event processing frameworks are important software components of High Energy Physics (HEP) experiments, playing a critical role in building applications for HEP-specific workflows such as trigger or event reconstruction. A key aspect of these frameworks is their ability to efficiently orchestrate the parallel processing of algorithms for multiple events simultaneously. As heterogeneous setups, including GPUs and other accelerators, become increasingly accessible, there is a growing need to incorporate these resources into the frameworks effectively.

Our demonstrator project investigates the viability of developing an event-processing framework that utilizes heterogeneous resources in the Julia language. For the first demonstrator, we chose Dagger.jl as a library supporting parallel and heterogeneous computing. We will present the general assumptions and requirements for a framework, address the current state of the demonstrator project and its methodology, as well as discuss our experience with Julia's ecosystem and Dagger.jl, highlighting the strengths and challenges we encountered.

Talks / **5**

Generating Feynman Diagrams for QED in Julia

Author: Anton Reinhard[None]

Co-authors: Simeon Ehrig ; Uwe Hernandez Acosta [1]

[1] *Helmholtz-Zentrum Dresden-Rossendorf*

Corresponding Authors: u.hernandez@hzdr.de, a.reinhard@hzdr.de, s.ehrig@hzdr.de

Calculating differential cross-sections of scattering processes is a crucial observable in high-energy physics, used to predict experimental outcomes and test theoretical models. For perturbative quantum field theories, this involves generating all possible Feynman diagrams for a given scattering process and translating them into computable functions. This becomes cumbersome very rapidly, especially for high-multiplicity processes. In this talk, we introduce a method implemented in Julia for generating these functions for arbitrary scattering processes in perturbative QED, utilizing the GraphComputing.jl library. Our approach incorporates novel results and reuse optimizations, which could be extended to other theories or even the entire Standard Model and beyond.

Talks / 6

IntegrationTests.jl: a framework for the automatic generation of integration tests for Julia projects and eco systems

Author: Simeon Ehrig[1]

Co-authors: Anton Reinhard ; Uwe Hernandez Acosta [2]

[1] *CASUS -center for advanced understanding*

[2] *Helmholtz-Zentrum Dresden-Rossendorf*

Corresponding Authors: s.ehrig@hzdr.de, u.hernandez@hzdr.de, a.reinhard@hzdr.de

To be successor, every larger software project needs to be tested to verify the correct functionality and to enable its functionality to be extended flawlessly. The type of tests can be very different and depends on the kind of software project. Software projects, that are divided into several sub-projects require integration tests to verify that the individual parts work together correctly.
At JuliaHEP 2023, I gave the talk "Unit and Integration testing in modularized julia package ecosystems"and talked about the problems that need to be solved when developing integration tests for a Julia package ecosystem. With the feedback from the talk, I developed IntegrationTests.jl [1], a framework to dynamically generate GitHub Action or GitLab CI integration jobs for a given Julia Project.toml. The talk explains the different problems to solve when adding integration tests in a Julia project and how IntegrationTests.jl solves them.

[1] https://github.com/QEDjl-project/IntegrationTests.jl

Talks / 7

Hadron Physics with Julia

Author: Mikhail Mikhasenko[1]

[1] *Ruhr Univeristy Bochum*

Corresponding Author: mikhail.mikhasenko@cern.ch

This talk will explore two areas. First, case studies will demonstrate how Julia has been effectively used for complex analyses in resonance physics and computationally demanding partial-wave analysis across several projects. Second, I will introduce a recent initiative aimed at standardizing hadronic-decay model serialization. In this context, the HadronicLineshapes.jl and ThreeBodyDecays.jl packages facilitate accurate and reproducible amplitude modeling, supporting broader adoption within the hadron physics community and advancing the research frontier.

Talks / 8

Machine Learning in Julia for Calorimeter Showers

Author: Daniel Assuncao Regado[None]

Co-authors: Graeme A Stewart [1]; Pere Mato Vila [1]; Piyush Raikwar [1]

[1] *CERN*

Corresponding Authors: piyush.raikwar@cern.ch, pere.mato@cern.ch, danielregado@gmail.com, graeme.andrew.stewart@cern.ch

The calorimeter in Large Hadron Collider (LHC) experiments measures particle energy by tracking showers from collisions. Describing these processes requires precise simulation methods, such as the Geant4 toolkit. Recently, generative models have emerged as a faster alternative based on different Machine Learning (ML) architectures, such as Diffusion and Variational Autoencoders.
The training of ML models is predominantly carried out using Python frameworks, primarily PyTorch and TensorFlow. In order to determine how mature ML development is using Julia, a denoising diffusion model, CaloDiffusion, was chosen to be implemented and trained with Flux.jl. On top of technical details, this talk also covers benchmarks of both implementations and analysis of performance using GPU profiling.

Talks / 9

EDM4hep.jl: Analysing EDM4hep files with Julia

Author: Pere Mato Vila[1]

[1] *CERN*

Corresponding Author: pere.mato@cern.ch

EDM4hep aims to establish a standard event data model for the store and exchange of event data in HEP experiments. The Julia package EDM4hep.jl is capable of generating Julia-friendly structures for the EDM4hep data model and reading event data files in ROOT format (either TTree or RNTuple) that are written by C++ programs, utilising the UnROOT.jl package. This contribution explores the motivations behind the primary design choices of this package, such as the exclusive use of structure of arrays (SoA) to access the stored collections, which then empower users to develop ergonomic data analyses using Julia's high-level concepts and functionality, while maintaining performance comparable to C++ programs. Several examples are given to illustrate how efficient data analysis can be achieved using high-level objects, eliminating the need to resort to flat n-tuples.

Talks / 10

Power of Python and Julia for Advanced Data Analysis

Author: Ianna Osborne[1]

[1] *Princeton University*

Corresponding Author: ianna.osborne@cern.ch

Python and Julia are two powerful languages that are transforming data analysis in high-energy physics (HEP).

We'll start by exploring why Python remains a go-to language for data analysis, and then pivot to Julia, which is gaining recognition for its impressive speed and suitability for scientific applications.

I'll show you how to create a dynamic workflow that combines the strengths of both languages. We' ll explore PythonCall for integrating Python's vast ecosystem into Julia projects and JuliaCall for embedding high-performance Julia code into Python scripts. You'll see how easy it is to blend these languages and why it's worth the effort.

Through hands-on examples, I'll demonstrate real-world applications where combining Python and Julia leads to faster, more efficient data processing. We'll tackle scenarios common in HEP, showing the practical benefits of this hybrid approach.

We'll also discuss the challenges you might face, like dependency management and ensuring compatibility between the two languages. I'll share strategies to overcome these hurdles and keep your projects running smoothly.

Looking ahead, we'll consider the exciting future possibilities of deeper Python-Julia integration and the role of the developer community in driving innovation. I hope to encourage you to experiment with this approach and contribute to the evolving ecosystem.

Talks / 11

Fast Jet Reconstruction in Julia

Author: Graeme A Stewart[1]

[1] *CERN*

Corresponding Author: graeme.andrew.stewart@cern.ch

Jet reconstruction remains a critical task in the analysis of data from HEP colliders. We describe in this paper a new, highly performant, Julia package for jet reconstruction, JetReconstruction.jl, which integrates into the growing ecosystem of Julia packages for HEP. With this package users can run sequential reconstruction algoritms for jets, In particular, for LHC events, the Anti-, Cambridge/Aachen and Inclusive algorithms can be used. For FCCee studies the use of alternative algorithms such as the generalised ee- and Durham are also supported.

The full reconstruction history is made available, allowing inclusive and exclusive jets to be retrieved. The package also provides the means to visualise the reconstruction.

The implementation of the package in Julia is discussed, with an emphasis on the features of the language that allow for an easy to work with, ergonomic, code implementation, that achieves high-performance. Julia's ecosystem offers the possibility to vectorise code, using single-instruction-multiple-data processing, in way that is transparent for the developer and more flexible than optimization done via C and C++ compilers. Thanks to this feature, the performance of JetReconstuction.jl is better than the current Fastjet C++ implementation in jet clustering for p-p events produced at the LHC.

Talks / 13

Unveiling the Jet Substructure using Julia

Authors: Sanmay Ganguly[1]; Sattwamo Ghosh[2]

[1] *IIT Kanpur*

[2] *IISER Kolkata*

Corresponding Authors: sg21ms204@iiserkol.ac.in, sanmay.ganguly@cern.ch

High energetic quarks and gluons, produced in a scattering phenomena, undergoes fragmentation and hadronization, leading to a spray of collimated particles, which are collectively clustered to form jets. In the ultra-high momentum regime, it may often happen that multiple energetic partons are within a geometric vicinity, and the jet thus formed has multiple sub-jets within it. This led to the formulation and study of the jet-substructure paradigm, which has evolved as a sub-branch of QCD studies. Thus, jet substructure has emerged as a powerful framework for studying the Standard Model at particle colliders. The FastJet C++ package provides several modules for jet and jet substructure analysis. This work presents the translation of a few of the functionalities of FastJet, including some of the taggers, groomers, jet filtering, and trimming algorithms, into Julia, highlighting the solutions and challenges we encountered. Additionally, the performance of the Julia implementation is measured with respect to the original FastJet code, and even though a significant improvement is not observed, it holds the potential for further optimizations that could lead to better performance in the future.

Talks / 14

Bayesian and general statistics in Julia

Author: Oliver Schulz[1]

[1] *Max Planck Society (DE)*

Corresponding Author: oliver.schulz@cern.ch

This tutorial will introduce statistical tooling in Julia, with a special focus on the Bayesian Analysis Toolkit BAT.jl. We'll show how to deal with probability distributions, build models and likelihood functions, and run parameter inference.

Talks / 15

Julia in the lab

Author: Oliver Schulz[1]

[1] *Max Planck Society (DE)*

Corresponding Author: oliver.schulz@cern.ch

From controlling vacuum, high voltage and motors, running data acquisition, performing data analysis, to publication, all in Julia.

Talks / 16

Julia in Trigger Level Analysis of Z' to bb

Author: Michael Steven Farrington[1]

[1] *Harvard University (US)*

Corresponding Author: michael.steven.farrington@cern.ch

During Run 3 of the LHC, trigger level analysis (TLA) offers the possibility of targetting new signals which other analyses are less sensitive to. I will showcase how Julia has been used as a part of the analysis workflow for a TLA search of a new heavy vector boson, Z', decaying to two bottom quarks. I will discuss the advantages and experience of using Julia in this analysis.

Talks / 17

The JuLeAna Software: How to run an entire experiment in Julia

Authors: Florian Henkes[1]; Oliver Schulz[2]

[1] *Tecnical University of Munich*

[2] *Max Planck Society (DE)*

Corresponding Authors: oliver.schulz@cern.ch, florian.henkes@tum.de

The Large Enriched Germanium Experiment for Neutrinoless ββ Decay (LEGEND) experimental program is dedicated to the search for the neutrinoless double-beta (0νββ) decay of ^{76}Ge with isotopically enriched high-purity germanium (HPGe) detectors and a discovery sensitivity beyond a half-life of 10^{28} years. The project's first phase, LEGEND-200, has stably accumulated physics data at the Laboratori Nazionali del Gran Sasso (LNGS) for over a year with 142 kg of HPGe detectors and plans to install more in the coming months. The experiment uses two software stacks with two independent analysis teams. This talk will highlight the status and development of the JuLeAna (Julia LEGEND Analysis) software stack and its application to current LEGEND data. It will focus on the performance and data handling for the Digital Signal Processing (DSP), the calibration and fitting routines, event level building, metadata, and IO handling. Furthermore, a quick showcase will highlight the dataflow adaption within a custom SLURM-based parallel processing environment.
This work is supported by the U.S. DOE and the NSF, the LANL, ORNL and LBNL LDRD programs; the European ERC and Horizon programs; the German DFG, BMBF, and MPG; the Italian INFN; the Polish NCN and MNiSW; the Czech MEYS; the Slovak RDA; the Swiss SNF; the UK STFC; the Russian RFBR; the Canadian NSERC and CFI; the LNGS and SURF facilities.

Talks / 18

RNTuple writing in Julia

Authors: Jerry ⊠ Ling[1]; Tamas Gal[2]

[1] *Harvard University (US)*

[2] *ECAP, FAU Erlangen-Nürnberg*

Corresponding Authors: jerry.ling@cern.ch, tamas.gal@fau.de

We briefly share insights gained from implementing RNTuple Reader twice: first in Python, and then in Julia. We discuss the composability of the RNTuple type system and demonstrate how Julia's multiple dispatch feature has been effectively employed to realize this concisely.

Regarding the implementation of RNTuple Writer, we outline the current capabilities and illustrate how they support end-user analyses. Furthermore, we present a roadmap for future development aimed at achieving seamless data I/O interoperability across various programming languages and libraries, including C++, Python, and Julia.

Lastly, we showcase the capabilities and performance of our Julia implementation with real examples. We highlight how our solution facilitates interactive analysis for end-users utilizing RNTuple.

Talks / **19**

FHist.jl – status of v0.11

Author: Jerry □ Ling[1]

[1] *Harvard University (US)*

Corresponding Author: jerry.ling@cern.ch

We share a status update on the histogramming package FHist.jl, which started out as a course final project but grew to be production-grade and is used in real ATLAS analysis.

We briefly go over the feature and performance of FHist.jl, including the core features as seen in ROOT's TH* classes, as well as axillary features such as integration with Plots.jl and Makie.jl.

Finally, we briefly discuss possible future directions of the project and potential nice-to-have enhancements (named axis, write to disk, GPU) and integrations (statistical fitting, distribution/pdf).

Talks / **20**

Training Implicit Generative Models via an Invariant Statistical Loss

Author: Jose de Frutos[1]

[1] *Universidad Carlos III*

Corresponding Author: josemanuel.defrutos22@gmail.com

Implicit generative models have the capability to learn arbitrary complex data distributions. On the downside, training requires telling apart real data from artificially-generated ones using adversarial discriminators, leading to unstable training and mode-dropping issues. As reported by Zahee et al. (2017), even in the one-dimensional (1D) case, training a generative adversarial network (GAN) is challenging and often suboptimal. In this work, we develop a discriminator-free approach to training 1-dimensional (1D) generative implicit models. Our loss function is a discrepancy measure between a suitably chosen transformation of the model samples and a uniform distribution; hence, it is invariant with respect to the true distribution of the data. We first formulate our method for 1D random variables, providing an effective solution for approximate reparameterization of arbitrary complex distributions. Then, we consider a temporal setting (both univariate and multivariate), in which we model the conditional distribution of each sample given the history of the process. We demonstrate through numerical simulations that this new method yields promising results, successfully learning true distributions in a variety of scenarios and mitigating some of the well-known problems that state-of-the-art implicit methods present.

Talks / **21**

Open-Source Simulation of Semiconductor Detectors

Author: Felix Hagemann[1]

Co-author: Oliver Schulz [2]

[1] *Max Planck Institut für Physik*

[2] *Max Planck Society (DE)*

Corresponding Authors: oliver.schulz@cern.ch, hagemann@mpp.mpg.de

SolidStateDetectors.jl is a novel open-source software solution used to simulate the behavior of solid state detectors, e.g. germanium and silicon detectors. The package calculates the electric fields and weighting potentials, as well as the charge drift in the detectors and detector output signals.
Users can define arbitrary detector geometries via simple configuration files using constructive solid geometry (CSG). Detectors may also be segmented/pixelized and have more than two electrical contacts. The environment of the detector can be included in the geometry and the field calculation to simulate the effect of nearby objects on the field in detectors with large passivated surfaces.
SolidStateDetectors.jl features fully multi-threaded high-performance 3D field calculation in both cylindrical and Cartesian coordinates. Recent feature additions include simulation of the charge-cloud self-interactions, automatic detector capacitance calculation, GPU-support for accelerated field calculations, and an extension to the Julia wrapper Geant4.jl, which allows for the simulation of realistic event distributions.

Talks / **22**

Porting the CMS pixel reconstruction to Julia: preliminary results

Authors: Andrea Bocci[1]; Maya Ali[2]; Mohamad Ayman Charaf[2]; Mohamad Khaled Charaf[2]; Philippe Gras[3]; Ruba El Houssami[2]

[1] *CERN*

[2] *American University of Beirut (LB)*

[3] *Université Paris-Saclay (FR)*

Corresponding Authors: andrea.bocci@cern.ch, philippe.gras@cern.ch, maa351@mail.aub.edu, ruba.el.houssami@cern.ch, mohamad.khaled.charaf@cern.ch, mohamad.ayman.charaf@cern.ch

The Patatrack pixel track reconstruction is a stand-alone project that has been extracted from the CMS reconstruction software. Over the years it has been used to test and evaluate different CPU and GPU technologies, like OpenMP, TBB, CUDA, HIP, SYCL, Kokkos, and Alpaka.

In order to evaluate the Julia programming language in the context of a realistic High Energy Physics software project, the Patatrack pixel track reconstruction is now being rewritten in Julia.

The project is under active development, and about 30% of the reconstruction algorithms have been documented, rewritten in Julia, and validated. The first results are very encouraging: the Julia version produces correct results, and has a single-threaded performance very close to that of the original C++ version.

This contribution will give an overview of the project and its long-term prospects, describe the challenges encountered during the work, along with the solutions chosen to address them, and present the preliminary results in terms of correctness and performance of the Julia implementation.

Talks / **23**

Empowering Underrepresented Communities Through Julia

Author: Shahzaib Abbas[1]

Co-author: Syed Ali Asghar [1]

[1] *University of Karachi*

Corresponding Authors: syedaliazgher2001@gmail.com, shahzaib2you@gmail.com

This presentation will explore the socio-economic impact of Julia. In light of the significant social inequalities in large cities like Karachi, mastering Julia opens up new opportunities for underrepresented communities. We will showcase how integrating Julia with CMS Open data sparks high school students' interest in STEM, and how we continue to nurture this enthusiasm through undergraduate projects as they advance in their academic careers.

Talks / 24

A Julia interface to the ROOT framework

Author: Philippe Gras[1]

[1] *Université Paris-Saclay (FR)*

Corresponding Author: philippe.gras@cern.ch

A new implementation of the ROOT framework Julia interface, ROOT.jl, has been performed. Previous implementation could not run with recent Julia releases due to the stop of support of the Cxx.jl library it was based on to interface with the ROOT C++ libraries. The new implementation is based on CxxWrap.jl and WrapIt!. CxxWrap.jl is a package to interface C++ libraries with Julia, while WrapIt! is an application to generate the C++ code needed by CxxWrap.jl. Many ROOT classes are supported, including classes for histogramming, plotting, the complete Geom libraries, I/O classes including TTrees. More classes will be added in the future. It provides ROOT file read and write support. The features of the packages and the challenge of its implementation will be presented.

Talks / 25

RootIO.jl: a Julia I/O library for the ROOT framework

Author: Yash Solanki[1]

Co-authors: Pere Mato Vila [2]; Philippe Gras [3]

[1] *Indian Institute of Technology, Delhi*

[2] *CERN*

[3] *Université Paris-Saclay (FR)*

Corresponding Authors: pere.mato@cern.ch, philippe.gras@cern.ch, 252yash@gmail.com

RootIO.jl is a package that provides a high-level abstraction for I/O to the ROOT files. It provides a streamlined interface for creating, writing, and filling ROOT TTrees. With this new package, the user can easily write all primitive types, arrays, vectors and dataframes to a TTree in a row-wise manner, without worrying about references and pointers to the objects. The package implementation uses ROOT.jl. The presentation will focus on implementation details, usage and I/O examples. Plans for extending the current features will also be presented.

Talks / 26

Welcome

Corresponding Authors: graeme.andrew.stewart@cern.ch, pere.mato@cern.ch

Computing Seminar / 27

Julia in high-energy physics: a paradigm shift or just another tool?

Corresponding Author: u.hernandez@hzdr.de

The Julia programming language was designed for scientific computing and with its claimed usability („walks like Python") and speed („runs like C"), it seems to be a scientists'software dream come true. Julia appears to be particularly well-suited for high-energy physics (HEP), where reliable software tools and rapid development cycles are crucial for everyday work. Whether it's data processing, or the simulation of the whole experiment, or the final data analysis and interactive visualization, the Julia ecosystem —with over ten thousand packages —might be a modern and high-performance software solution and the right set of tools to easily build any missing pieces.
In this talk, we will discuss, if the Julia programming language meets these requirements and can withstands testing on the workbenches of HEP. Additionally, we give an overview of current contributions in Julia to the HEP-related software stack and its potential trajectory. Moreover, we explore how the software development process itself can benefit from Julia, as it strikes an ideal balance between high-performance technology and student-friendly training —an especially valuable combination for the rapidly moving high-energy physics community.

Computing Seminar / 28

Discussion

Talks / 29

3D Neutrino Event Display RainbowAlga.jl

Author: Tamas Gal[1]

[1] *ECAP, FAU Erlangen-Nürnberg*

Corresponding Author: tamas.gal@fau.de

RainbowAlga.jl is a 3D neutrino event display based on GLMakie. The package was already introduced and demonstrated in 2013 in its early development phase at the very first JuliaHEP. This talk shows the current status of the package which has evolved to a helpful utility that is customisable and offers both interactive and programmatic ways to display neutrino events in Cherenkov neutrino detectors KM3NeT, IceCube, Baikal BDUNT/GVDl, P-One or Trident.

30

Workshop Social Diner

Hackathon / 31

Julia for physics (and physicists)

Hackathon / 32

Hands-on Julia

Corresponding Author: u.hernandez@hzdr.de

This session will provide a step-by-step guide on building a Julia package, implementing physics models, and applying best practices for the everyday workflow of a physicist.

Hackathon / 33

Hands-on Julia II

Corresponding Author: u.hernandez@hzdr.de

This session will build on the morning's hands-on Julia work, introducing more advanced topics based on earlier progress.

Hackathon / 34

Introduction to GPU programming

Corresponding Author: tim@juliahub.com

Hackathon / 35

Final Discussion

Corresponding Author: philippe.gras@cern.ch

Hackathon / 36

Closeout

Hackathon / 37

Hackathon

Computing Seminar / 38

Julia as a Statically-Compiled Language

Corresponding Author: jeff.bezanson@gmail.com

Though first intended as an interactive, productivity language, Julia has seen a surprising amount of interest as an alternative to C and C++. In this talk I will discuss why that is, and what we are doing to encourage and enable those use cases. Providing more compile-time safety and making it easier to deploy Julia programs are major areas of focus for the project and JuliaHub currently.

Hackathon / 39

Q&A Juila for physics

Hackathon / 40

Q&A GPU programming