ISBN: 978-3-95450-247-9 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2024-WEPR52

Bmad-Julia: A Julia ENVIRONMENT FOR ACCELERATOR SIMULATIONS INCLUDING MACHINE LEARNING

D. Sagan[1,*,†], M. G. Signorelli[1], A. Coxe[2], G. H. Hoffstaetter[1]
[1]CLASSE, Cornell University, Ithaca, NY, USA
[2]Physics, Old Dominion University, Norfolk, VA, USA

Abstract

Bmad-Julia is a new, open-source project to develop, in the Julia language, a set of modular packages providing the fundamental tools and methods commonly needed for accelerator simulations. By avoiding the necessity of ”reinventing the wheel”, simulation programs can be developed in less time and with fewer bugs than developing from scratch. A modern simulation framework such as Bmad-Julia is greatly needed since the ever increasing demands placed upon machine performance require ever more comprehensive accelerator modeling.

In addition to the toolkit packages, the Bmad-Julia project will develop accelerator simulation programs for various simulation tasks. Initial program development will focus on Machine Learning / Artificial Intelligence (ML/AI) applications. However, Bmad-Julia will be applicable to accelerator simulations in general. Discussed is the present state of the project as well as future plans.

INTRODUCTION

The disorganized state of accelerator physics simulation development has been recognized by the latest High-Energy Physics community Snowmass white paper [1]:[1]

> The development of beam and accelerator physics codes has often been largely uncoordinated. This comes at a great cost, is not desirable and may not be tenable. Due to developers retiring or moving on to other projects, numerous simulation programs have been completely abandoned or are seldom used. This has resulted in a collection of codes that are not interoperable, use different I/O formats and quite often duplicate some physics functionalities using the exact same underlying algorithms. Frequently there is a huge impediment to maintaining these programs due to poorly-written code and lack of documentation. Additionally, many of the programs that are available tend to be “rigid”. That is, it is generally difficult to modify a program to simulate something it is not designed to simulate *a priori*. Adding a new type of lattice element that a particle can be tracked through is one such example.

As a consequence, the Snowmass white paper makes a recommendation on how to ameliorate the situation:

> **Recommendation on community ecosystems & data repositories** Organize the beam and accelerator modeling tools and community through the development of (a) ecosystems of codes, libraries and frameworks that are interoperable via open community data standards, (b) open access data repositories for reuse and community surrogate model training, (c) dedicated Centers and distributed consortia with open community governance models and dedicated personnel to engage in cross-organization and -industry development, standardization, application and evaluation of accelerator and beam modeling software and data.

The Bmad-Julia project is structured to be part of this ecosystem. The Bmad-Julia project was started last year (2023) and aims to develop a set of open source packages (code modules), written in the Julia programming language, that can serve as the basis for future accelerator simulation programs. This will enable simulation programs to be developed in less time and with fewer bugs (due to code reuse) than can be done for a program developed from scratch. Along with the packages, programs will be developed for various simulation tasks. The first applications will involve Machine Learning / Artificial Intelligence (ML/AI) but Bmad-Julia will not be limited to ML/AI applications.

BASED ON Bmad

Bmad-Julia is inspired by the current Bmad ecosystem of toolkits and programs for the simulation of relativistic charged particles and X-rays in accelerators and storage rings [2]. Bmad has a wide range of capabilities including tracking of polarized beams and X-rays, while simulating coherent synchrotron radiation (CSR), wakefields, Touschek scattering, higher order mode (HOM) resonances, space-charge dominated beams, weak-strong beam-beam interactions, and much more.

Although the Bmad-Julia code will be completely separate from the existing Bmad code, the development of Bmad-Julia will rely heavily on the many years of experience the developers have developing Bmad. Concepts developed in Bmad will serve as a paradigm for the present effort.

It should be noted that, due to Bmad’s wide adoption and breadth of capabilities, for the foreseeable future, Bmad-Julia will not be a replacement for Bmad, and both will co-exist side by side.

[*] david.sagan@cornell.edu
[†] Work supported in part by Department of Energy grant DE-SC0018370 and National Science Foundation award DMR-1829070.
[1] DOE-sponsored Snowmass reports outline the major funding priorities for the HEP community

WHY Julia?

In modern AI/ML and modeling frameworks (e.g., PyTorch, BLAST, WarpX, ImpactX [3]), the trend to combine a precompiled, high-performance language such as C++ with a scriptable interface language such as Python has been highly successful in ensuring performance, modularity, and productivity. Revisiting the language choice, a new alternative emerged with the Julia programming language, which adopts just-in-time (JIT) compilation and multiple dispatch as central paradigms of the language. Such features significantly simplify the development process, enable the entire ecosystem to be fully differentiable, and provides performance on-par with that of C. At the same time, Julia can be rapidly scripted (dynamic dispatch) in simple syntax, effectively removing the "two-language" challenge for developers [4]. Although still relatively young, Julia provides capabilities that simplifies important aspects of this project [5].

Julia comes with a full-featured interactive command-line REPL (Read-Evaluate-Print-Loop) built into the Julia executable similar to the REPL in Python. This means that the lattice description format, and the interaction between a user and simulation software in general, can be through the Julia language itself. Consequently, simulations will not be constrained by some program-defined language (like with MAD, Elegant, Bmad, etc.), and the user will automatically have access to such features as plotting, optimization packages, linear algebra packages, etc. This is a massive boost to the versatility and usability of any simulation program. Additionally, code maintainability is greatly improved since the quantity of code that needs to be developed is reduced.

That Julia is an excellent choice for the Bmad-Julia project is exemplified by the conclusions of the paper "Potential of the Julia programming language for high-energy physics computing" [6] written by a team of researchers from various laboratories around the world: "Julia and its ecosystem are impressively fulfilling all these requirements [for use in HEP applications]... The capacity to provide, at the same time, ease of programming and performance makes Julia the ideal programming language for HEP ... the HEP community will definitively benefit from a large scale adoption of the Julia programming language for its software development."

Bmad-Julia PROJECT GOALS

The general goals of the Bmad-Julia project are (a) to develop an ecosystem of modular and extensible packages that allows for the easy construction of accelerator simulation programs, and (b) to use these packages to develop simulation programs. Initially, there will be a strong ML/AI orientation.

The success and sustainability of Bmad-Julia will depend heavily on the involvement of the entire accelerator physics community. To this end, the Bmad-Julia project seeks community engagement with weekly planning meetings, open to all, and with a SLACK workspace where development can be discussed. Additionally, regular workshops and other informational meetings will be scheduled as the project advances.

In the short-term, needed are packages for defining and manipulating lattices, tracking of particles, Truncated Power Series Algebra (TPSA), differential algebra maps, and normal form decomposition and analysis of differential algebra maps to extract such things as emittances, Twiss parameters, resonance strengths, etc. In cases where packages external to Bmad-Julia have the needed functionality, rather than reinvent the wheel, appropriate interfaces will be developed.

Experience with Bmad has shown that there is no ideal way to track through a lattice with different tracking methods, all having their own strengths and weaknesses. Bmad-Julia will thus include multiple tracking packages. Considerations in developing these packages include speed, accuracy, and the ability to simulate various models of machine components. Needed is software that has the ability (not necessarily at the same time) to track TPSA maps, track using GPUs, track with backwards differentiation, etc.

Bmad-Julia CURRENT STATE

All Bmad-Julia development is in public open-source Git repositories hosted on GitHub. At present, the packages being developed are:

AcceleratorLattice.jl [7] Package for accelerator lattice construction and manipulation. Lattice elements are Julia structs which have a Dict component that can store arbitrary information which is important for flexibility as Bmad-Julia is developed. As in Bmad, ordered lists of elements, which can represent rings, linacs, injection and extraction lines, etc., are grouped into Branch structures. A Lat structure — which is essentially a set of Branches — can contain all the information about an entire accelerator complex including interconnections between different branches and is thus capable of being used for start-to-end simulations.

Bmad concepts such as the superposition of elements on top of other elements and control elements which control the parameters of other elements will exist in Bmad-Julia.

AtomicAndPhysicalConstants.jl [8] Library of atomic and subatomic particle properties and other physical constants (speed of light, etc.). The package has Particle structs for defining particles such as Helium-3+ and positrons. The units used are flexible and can be set differently in different packages that use this package. EG: Units of mass may be set to MeV/c^2, AMU, or something else as desired.

GTPSA.jl [9] Full-featured Julia interface to the Generalized Truncated Power Series Algebra (GTPSA) library developed by Laurent Deniau [10]. GTPSA performs automatic differentiation (AD) of real and complex multi-variable functions to arbitrary orders. GTPSA.jl is significantly faster than other Julia AD packages for calculating derivatives above first-order.

NonlinearNormalForm.jl [11] Nonlinear normal form analysis using differential algebra maps *including spin*. Included are methods for calculating parameter-dependent normal forms using vector fields (Lie operators), operations including vector fields and maps (e.g. logarithms of maps, Lie brackets, etc.), map inversion/partial inversion, and more.

SimUtils.jl [12] Simulation utility routines for needed functionality that does not exist in any external packages.

Except for the AcceleratorLattice package, all the packages currently being developed are useful in fields outside of accelerator physics from cosmology to chemistry.

MACHINE LEARNING

Machine Learning (ML) and Artificial Intelligence (AI) are having a large impact on accelerator operations and optimizations. ML/AI can provide advanced optimization tools that are uniquely suited for many accelerator tasks, e.g., physics-informed Bayesian Optimization addresses tasks where measurement uncertainties are relevant [13] or where obtaining measurements is time-consuming or expensive. Additionally, the construction of ML-based surrogate models has shown powerful speedups when describing hard-to-model sections of accelerators, e.g., for space charge-dominated beams [14].

Despite this, no comprehensive accelerator simulation toolkit has been designed with ML/AI in focus. The development of Bmad-Julia will address this. Bmad-Julia will provide a maximally-differentiable simulation and lattice design environment, with the full-features of present Bmad and more. This enables machine learning techniques (e.g. backwards differentiation to train ML models) to be employed naturally, as well as any optimization techniques utilizing automatic differentiation for lattice design purposes; Julia's entire ecosystem of optimizers and machine learning packages are at one's fingertips in Bmad-Julia. Alternatively, Julia provides bindings to existing Python ML/AI packages such as OpenAI Gym [15] and PyTorch [16], allowing for their utilization as well.

Computation speed is critical for ML/AI applications as well as for other accelerator-based simulations. Hence, there will be a heavy focus on developing GPU compatible code along with multi-threading and multi-processing capabilities.

INTEROPERABILITY

Bmad-Julia is being developed with consideration for compatibility with the wider accelerator modeling community. This includes data standards and lattice description. This will facilitate cross-checking and benchmarking results, and enable integration between toolkits. This is important for start-to-end simulations, integration with external optimization tools, and seamless training of machine learning models. For example, routines for reading and writing particle distributions using the openPMD standard [17, 18] will be developed and Bmad-Julia will leverage prior investment in a scalable data stack that was developed for the US Exascale Computing Project (ECP) [19].

Translation software will be developed to translate between the Bmad-Julia lattice format and the existing Bmad format, as well as other commonly used formats (MAD, etc.). Since Bmad-Julia will encompass the major features found in Bmad, translation from Bmad to Bmad-Julia should be close to 100%. Back translation to Bmad will be fairly good except in a few areas such as control element descriptions since Bmad-Julia has a more expressive syntax in this area.

For translation to non-Bmad lattice formats, translation will never be perfect (far from it) since different programs have different element types and differing constructs. Yet experience has shown that just being able to translate basic information, like type of element lengths and strengths, is very helpful.

For portability for programs that do not use Julia, or do not use the Bmad-Julia lattice parsing package, a standard lattice syntax will be developed using a standard data format (EG: YAML, TOML, JSON). Along with this, standardized parameter names and meanings will be established. The lattice syntax will be extensible in the sense that the syntax will allow custom information to be stored along side the information defined by the standard. It is envisaged that this portable lattice standard could serve as a lingua franca for lattice communication among non Bmad-Julia programs.

OUTLOOK

The Bmad-Julia project is in its infancy but already shows much promise. Bmad-Julia has the potential to have an impact similar to that of Geant4. The versatility of the Geant4 toolkit for the simulation of particle passage through matter [20] has resulted in Geant4 having a large impact not only in the field of particle physics, but in other domains as well — including nuclear physics, astrophysics and space science, biomedical physics, and archaeology. Geant4's main reference [21] has achieved more than 16,000 citations in the Web of Science [22], making it the most cited publication in particle and field physics, nuclear physics, and other fields [20]. Similarly, the versatility of the Bmad-Julia package ecosystem means that the potential payback for the Bmad-Julia project is very large compared to the investment.

The key to achieving success of the Bmad-Julia project is, ultimately, engaging the entire accelerator physics community just as there is a thriving community for Geant4 that supports it. A strong effort will be made to create such a community.

ACKNOWLEDGEMENTS

Thanks must go to the many people who have contributed their expertise, knowledge, to this project especially Laurent Deniau and Etienne Forest. Other contributors are Dan Abell, Juan Pablo Gonzalez-Aguilera, Scott Berg, Oleksii Beznosov, Auralee Edelen, Axel Huebl, Li Liching, Chris Mayes, Ryan Roussel, Hugo Slepicka, Qi Tang, and Jean-Luc Vay.

REFERENCES

[1] S. Biedron *et al.*, "Snowmass21 accelerator modeling community white paper," *arXiv*, 2022. doi:10.48550/arXiv.2203.08335

[2] D. Sagan, "Bmad: A relativistic charged particle simulation library," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 558, no. 1, pp. 356–359, 2006, Proceedings of the 8th International Computational Accelerator Physics Conference. doi:10.1016/j.nima.2005.11.001

[3] R. T. Sandberg *et al.*, "Synthesizing particle-in-cell simulations through learning and gpu computing for hybrid particle accelerator beamlines," *Proceedings of the Platform for Advanced Scientific Computing*, vol. PASC24, 2024, accepted. doi:10.48550/arXiv.2402.17248

[4] J. Storopoli, R. Huijzer, and L. Alonso, *Julia Data Science*. 2021. https://juliadatascience.io

[5] J. Revels, M. Lubin, and T. Papamarkou, *Forward-mode automatic differentiation in julia*, 2016.

[6] J. Eschle *et al.*, "Potential of the julia programming language for high energy physics computing," English (US), *Computing and Software for Big Science*, vol. 7, no. 1, 2023. doi:10.1007/s41781-023-00104-x

[7] D. Sagan *et al.*, *AcceleratorLattice.jl*. https://github.com/bmad-sim/AcceleratorLattice.jl

[8] A. Coxe *et al.*, *Atomicandphysicalconstants.jl*, https://github.com/bmad-sim/AtomicAndPhysicalConstants.jl.

[9] M. G. Signorelli *et al.*, *GTPSA.jl*. https://github.com/bmad-sim/GTPSA.jl

[10] L. Deniau and C. I. Tomoiagă, "Generalised Truncated Power Series Algebra for Fast Particle Accelerator Transport Maps," in *Proc. IPAC'15*, Richmond, VA, USA, May 2015, pp. 374–377. doi:10.18429/JACoW-IPAC2015-MOPJE039

[11] M. G. Signorelli *et al.*, *NonlinearNormalForm.jl*. https://github.com/bmad-sim/NonlinearNormalForm.jl

[12] D. Sagan, L. Li, *et al.*, *SimUtils.jl*. https://github.com/bmad-sim/SimUtils.jl

[13] J. Duris *et al.*, "Bayesian optimization of a free-electron laser," *Phys. Rev. Lett.*, vol. 124, p. 124 801, 12 2020. doi:10.1103/PhysRevLett.124.124801

[14] C. Garcia-Cardona and A. Scheinker, "Machine learning surrogate for charged particle beam dynamics with space charge based on a recurrent neural network with aleatoric uncertainty," *Phys. Rev. Accel. Beams*, vol. 27, no. 2, p. 024 601, 2024. doi:10.1103/PhysRevAccelBeams.27.024601

[15] G. Brockman *et al.*, "Openai gym," *arXiv*, 2016. doi:10.48550/arXiv.1606.01540

[16] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[17] A. Huebl *et al.*, *openPMD: A meta data standard for particle and mesh based data*, https://github.com/openPMD, 2015. doi:10.5281/zenodo.591699

[18] F. Poeschel *et al.*, "Transitioning from file-based hpc workflows to streaming data pipelines with openpmd and adios2," in *Smoky Mountains Computational Sciences and Engineering Conference*, Springer, 2021, pp. 99–118. doi:10.1007/978-3-030-96498-6_6

[19] A. Huebl *et al.*, "Next Generation Computational Tools for the Modeling and Design of Particle Accelerators at Exascale," in *Proc. NAPAC'22*, Albuquerque, NM, USA, 2022, pp. 302–306. doi:10.18429/JACoW-NAPAC2022-TUYE2

[20] T. Basaglia, Z. Bell, D. D'Agostino, P. Dressendorfer, M. Pia, and E. Ronchieri, "Geant4 silver anniversary: 25 years enabling scientific production," *J. Instrum.*, vol. 19, no. 01, p. C01037, 2024. doi:10.1088/1748-0221/19/01/C01037

[21] S. Agostinelli *et al.*, "Geant4–a simulation toolkit," *Nucl. Instrum. Meth. A*, vol. 506, no. 25, 2003. doi:10.1016/S0168-9002(03)01368-8

[22] *Web of science*. https://www.webofscience.com/wos