ParaMonte: A high-performance serial/parallel Monte Carlo simulation library for C, C++, Fortran

Amir Shahmoradi[1, 2] **and Fatemeh Bagheri**[1]

1 Department of Physics, The University of Texas, Arlington, TX **2** Data Science Program, The University of Texas, Arlington, TX

DOI:

Software

- Review
- Repository
- Archive

Editor:

Submitted: 29 September 2020
Published:

Summary

ParaMonte (standing for Parallel Monte Carlo) is a serial and MPI/Coarray-parallelized library of Monte Carlo routines for sampling mathematical objective functions of arbitrary-dimensions, in particular, the posterior distributions of Bayesian models in data science, Machine Learning, and scientific inference. The ParaMonte library has been developed with the design goal of unifying the **automation**, **accessibility**, **high-performance**, **scalability**, and **reproducibility** of Monte Carlo simulations. The current implementation of the library includes **ParaDRAM**, a **Para**llel **D**elyaed-**R**ejection **A**daptive **M**etropolis Markov Chain Monte Carlo sampler, accessible from a wide range of programming languages including C, C++, Fortran, with a unified Application Programming Interface and simulation environment across all supported programming languages. The ParaMonte library is MIT-licensed and is permanently located and maintained at https://github.com/cdslaborg/paramonte.

Statement of need

Monte Carlo simulation techniques (Metropolis & Ulam, 1949), in particular, the Markov Chain Monte Carlo (MCMC) (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953) are among the most popular methods of quantifying uncertainty in scientific inference problems. Extensive work has been done over the past decades to develop Monte Carlo simulation programming environments that aim to partially or fully automate the problem of uncertainty quantification via Markov Chain Monte Carlo simulations. Example open-source libraries in C/C++/Fortran include `MCSim` in C (Bois, 2009), `MCMCLib` and `QUESO` (Prudencio & Schulz, 2012) libraries in C++, and `mcmcf90` in Fortran (Haario, Laine, Mira, & Saksman, 2006). These packages, however, mostly serve the users of one particular programming language environment. Some can perform only serial simulations while others are inherently parallelized. Furthermore, majority of the existing packages have significant dependencies on other external libraries. Such dependencies can potentially make the build process of the packages an extremely complex and arduous task due to software version incompatibilities, a phenomenon that has become known as the *dependency-hell* among software developers.

The ParaMonte library presented in this work aims to address the aforementioned problems by providing a standalone high-performance serial/parallel Monte Carlo simulation environment with the following principal design goals,

- **Full automation** of the library's build process and all Monte Carlo simulations to ensure the highest level of user-friendliness of the library and minimal time investment requirements for building the library as well as running and post-processing the Monte Carlo simulations.

- **Interoperability** of the core of the library with as many programming languages as currently possible, including C, C++, Fortran, as well as MATLAB and Python via the `ParaMonte::MATLAB` (Kumbhare & Shahmoradi, 2020) and `ParaMonte::Python`

(Shahmoradi, Bagheri, & Osborne, 2020) libraries.

- **High-Performance**, meticulously-low-level implementation of the library that guarantees the fastest-possible Monte Carlo simulations **without** compromising the reproducibility of the simulations or the extensive external reporting of the simulation progress and results.
- **Parallelizability** of all simulations via both MPI and PGAS/Coarray communication paradigms while **requiring zero-parallel-coding efforts from the user**.
- **Zero external-library dependencies** to ensure hassle-free library builds and Monte Carlo simulation runs.
- **Fully-deterministic reproducibility** and **automatically-enabled restart functionality** for all ParaMonte simulations (up to 16 digits of decimal precision if requested by the user).
- **Comprehensive-reporting and post-processing** of each simulation and its results as well as their efficient compact storage in external files to ensure the reproducibility and comprehensibility of the simulation results at any time in the future.

The origins of ParaMonte

The ParaMonte library grew out of the need for a free user-friendly high-performance parallel software for stochastic optimization, sampling, and integration problems in scientific inference and Data Science applications. The project started in 2012 to aid the research goals of the primary author of the package in the field of High Energy Astrophysics and Bioinformatics (Shahmoradi, 2013), (Shahmoradi, 2013), (Shahmoradi & Nemiroff, 2014), (Shahmoradi & Nemiroff, 2015), (Shahmoradi & Nemiroff, 2019), (Osborne, Shahmoradi, & Nemiroff, 2020), (J. A. Osborne, Shahmoradi, & Nemiroff, 2020). It remained in private usage over the years until summer 2020, when version 1.0 of the library was as an open-source project for public usage and contributions.

Many of the contemporary research problems are computationally demanding and require the analysis of vast amounts of high-dimensional data by a community of non-computer-science domain researchers who might be neither familiar with details of software dependencies, build processes, and complexities nor even with the inner-workings of the various stochastic optimization and sampling techniques. As such, since its inception, the ParaMonte library has been built upon the two pillars of user-friendliness and high-performance.

The build process

The ParaMonte library is permanently located on GitHub and is available to view at: https://github.com/cdslaborg/paramonte. The build process of the library is fully automated. Extensive detailed instructions are also available on the documentation website of the library.

For the convenience of users, each versioned release of the library's source code also includes prebuilt, ready-to-use, copies of the library for `x64` architecture on Windows, Linux, macOS, in all supported programming languages, including C, C++, Fortran. These prebuilt libraries automatically ship with the language-specific example codes and build scripts that fully automate the process of building and running the examples. Users can either adapt the example's source files and build scripts to their own needs or in more sophisticated scenarios, they can simply link their applications against the supplied prebuilt library.

Where the prebuilt libraries cannot be used, users can readily call the Unix-Bash and Windows-Batch build-scripts that are provided with the source code of the library to fully automate the build process of the library. These build scripts have been developed to automate the installation of any missing components that may be required for the successful build of the library, including the `cmake` build software, the GNU C/C++/Fortran compilers, as well as the MPI/Coarray libraries. All of these tasks are performed with the explicit permission granted by the user. The ParaMonte build scripts are heavily inspired by the impressive `OpenCoarrays` open-source software (Fanfarillo et al., 2014) developed and maintained by the Sourcery Institute.

The ParaMonte samplers

The current implementation of the ParaMonte library includes the **Para**llel **D**elayed-**R**ejection **A**daptive **M**etropolis Markov Chain Monte Carlo (`ParaDRAM`) sampler (Shahmoradi & Bagheri, 2020), (Shahmoradi & Bagheri, 2020a), (Shahmoradi & Bagheri, 2020b), (Kumbhare & Shahmoradi, 2020), and several other samplers whose development is in progress as of writing this manuscript. The ParaDRAM algorithm is a variant of the DRAM algorithm of (Haario et al., 2006) and can be used in either serial or parallel mode.

In brief, the ParaDRAM sampler continuously adapts the shape and scale of the proposal distribution throughout the simulation to increase the efficiency of the sampler. This is in contrast to the traditional MCMC samplers where the proposal distribution remains fixed throughout the simulation. The ParaDRAM sampler provides a highly customizable MCMC simulation environment whose complete description goes beyond the scope and limits of this manuscript. All of these simulation specifications are, however, expensively explained and discussed on the documentation website of the ParaMonte library. The description of all of these specifications are also automatically provided in the output `*_report.txt` files of every simulation performed by the ParaMonte samplers.

Several additional more advanced samplers are also currently under development and are scheduled for release in 2021.

Parallelism

Two modes of parallelism are currently implemented for all ParaMonte samplers,

- The **Perfect Parallelism** (multi-Chain): In this mode, independent instances of a particular ParaMonte sampler of choice run concurrently. Once all simulations are complete, the sampler compares the output samples from all processors with each other to calculate various statistics and to ensure that no evidence for a lack of convergence to the target density exists in any of the output chains.
- The **Fork-Join Parallelism** (single-Chain): In this mode, a single processor is responsible for collecting and dispatching information, generated by all processors, to construct the full sample from the target density function.

For each parallel simulation in the Fork-Join mode, the ParaMonte samplers automatically compute the speedup gained compared to the serial mode. The speedup for a wide range of the number of processors is also automatically computed and reported in the output `*_report.txt` files that are automatically generated for all simulations. The processor contributions to the construction of the final output sample are also reported along with all visited states in the output `*_chain.*` files. Such information is particularly useful for finding the optimal number of processors for a given problem at hand, by first running a short simulation to predict the optimal number of processors from the sampler's output information, followed by the full production run using the optimal number of processors. For a comprehensive description and algorithmic details see (Shahmoradi & Bagheri, 2020), (Shahmoradi & Bagheri, 2020a).

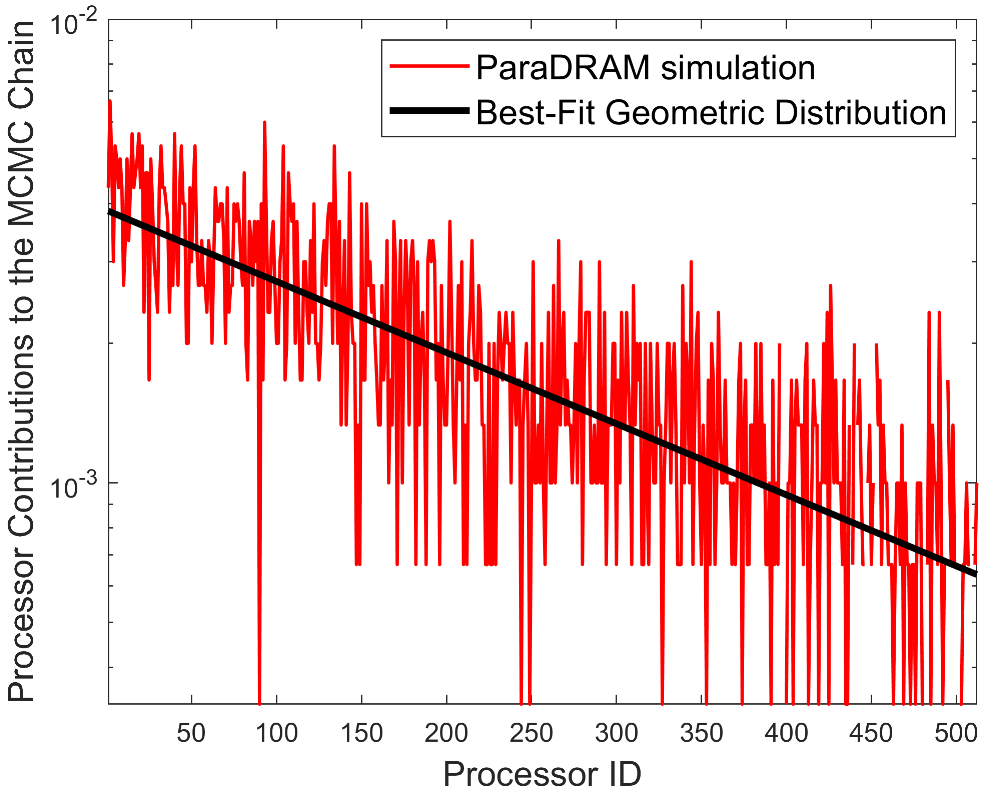

Figure 1: An illustration of the contributions of 512 Intel Xeon Phi 7250 processors to a ParaMonte-ParaDRAM simulation parallelized via the Fork-Join paradigm. The predicted best-fit Geometric distribution from the post-processing phase of the ParaDRAM simulation is shown by the black line. The data used in this figure is automatically generated for each parallel simulation performed via any of the ParaMonte samplers.

As we argue in (Shahmoradi & Bagheri, 2020, Shahmoradi & Bagheri (2020a)) for the particular case of MCMC simulations, the distribution of processor contributions to the construction of a final sample from an objective function in the Fork-Join parallelism paradigm follows a Geometric curve. Figure 1 depicts the processor contributions to an example ParaDRAM simulation of a variant of Himmelblau's function. Superposed on the distribution of processor contributions is the Geometric fit to the data. Figure 2 illustrates an example strong-scaling behavior of the sampler and the predicted speedup by the sampler for a range of processor counts.

The exact strong scaling behavior of Fork-Join parallel simulations is highly dependent on the sampling efficiency of the problem. For a fixed number of processors, the parallel efficiency of the samplers monotonically increases toward the maximum theoretical speedup with decreasing sampling efficiency. This theoretical maximum parallel speedup and the corresponding predicted optimal number of processors are also computed for each parallel simulation and reported in the final output `*_report.txt` files.

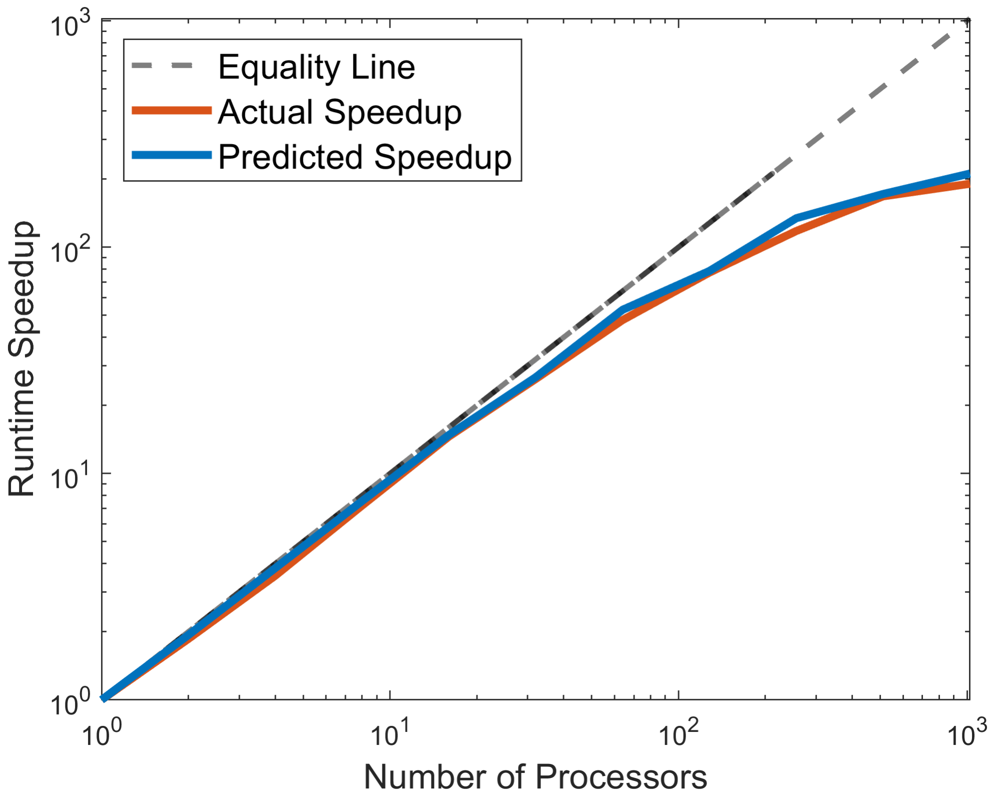

Figure 2: A comparison of the actual strong scaling behavior of an example ParaMonte-ParaDRAM simulation from 1 to 1088 processors with the strong-scaling behavior predicted during the post-processing phases of the corresponding individual ParaDRAM simulations. The data used in this figure is automatically generated for each parallel simulation performed via any of the ParaMonte samplers. The black dashed line represents the perfect parallelism.

Efficient compact storage of the output samples

Efficient continuous external storage of the output of Monte Carlo simulations is essential for both post-processing of the results and the restart functionality of the simulations, should any interruptions happen at runtime. However, as the number of dimensions or the complexity of the target density increases, such external storage of the output can easily become a challenge and a bottleneck in the speed of an otherwise high-performance sampler. Given the currently-available computational technologies, input/ouput (IO) to external hard-drives can be 2-3 orders of magnitude slower than the Random Access Memory (RAM) storage.

To alleviate the effects of such external-IO speed bottlenecks, the ParaMonte samplers have been devised to carefully store the resulting samples in a small *compact*, yet ASCII human-readable format in external output files. This **compact** storage, as opposed to the **verbose** (or in the case of the ParaDRAM sampler, the **Markov-chain**) storage format, leads to significant speedup of the simulations while requiring 4-100 times less external memory to store the output samples in the external output files. The exact amount of reduction in the external memory usage depends on the runtime sampling efficiency of the samplers. Additionally, the format of output files can be set by the user to `binary`, further reducing the memory foot-print of the simulations while increasing the simulation speed. The implementation details of this compact storage format are extensively discussed in (Shahmoradi & Bagheri, 2020), (Shahmoradi & Bagheri, 2020a).

The restart functionality

Each ParaMonte sampler is automatically capable of restarting an existing interrupted simulation, whether in serial or parallel. All that is required is to rerun the interrupted simulation with the same prefix for the simulation output file names. The ParaMonte samplers automatically detect the presence of an incomplete simulation in the output files and restart the simulation from where it was left off.

Furthermore, if the user sets the seed of the random number generator of the sampler before running the simulation, *the ParaMonte samplers are capable of regenerating the same output sample that would have been produced if the simulation had not been interrupted in the first place*. Such **fully-deterministic reproducibility into-the-future** is guaranteed with 16 digits of decimal precision for the results of any ParaMonte simulation, whether serial or in parallel. To our knowledge, this is a unique feature of the ParaMonte library that does not appear to exist in any of the contemporary libraries for Monte Carlo simulations.

The ParaMonte Application Programming Interface

Special care has been made to develop highly-similar (if not the same) Application Programming Interface (API) to the ParaMonte library samplers across all supported programming languages. Ensuring backward-compatibility has been also one of the high priorities in the development of the library and will remain a priority for any future development and expansion of the project.

The project currently uses semantic versioning to readily inform the users about the backward-compatibility of each release. This semantic versioning, however, only applies to the ParaMonte samplers' API. Conversely, the API to the foundational procedures of the ParaMonte kernel tend to be dynamic and subject to changes without incrementing the library's major version. This is line with the primary purpose of the library as a Monte Carlo optimization and sampling toolbox.

Nevertheless, the kernel documentation is well developed and accessible to all, including the end-users, should they decide to use any of the basic routines in the library beyond the samplers. The kernel API documentation is permanently available on GitHub at its dedicated repository: https://cdslaborg.github.io/paramonte-kernel-doc/html/ and is accessible from the ParaMonte's main documentation portal.

Tests and code coverage

The entire kernel routines of the ParaMonte library are currently tested with over 866 individual tests. Out of these, 510 test units currently cover and verify the functionalities of the basic building-block routines of the library and 156 tests verify the semantics and different aspects of the ParaMonte samplers built on top of the basic routines.

The code coverage report analyses of the ParaMonte kernel library can be generated for a wide range of input build configurations. For example, corresponding to each programming language (C, C++, Fortran, MATLAB, Python, R, . . .), a separate code coverage report for the kernel routines can be generated. Similarly, different library builds (debug, testing, release), library types (static, dynamic), and memory allocation schemes (stack or heap) can lead to different code coverage reports. This is primarily because the kernel routines heavily utilize compiler preprocessor directives to accommodate the support for multiple programming languages, Operating Systems and architectures, as well as multiple parallelism paradigms.

Among all build configurations, however, the single most important specification that yields the largest differences in code coverage reports is the parallelism paradigm (serial, mpi, and Coarray parallelism). The code coverage reports for the serial, MPI, and Coarray modes are remarkably different since each parallelism paradigm activates a different set of codes in the ParaMonte kernel routines.

Thus, with each tagged release of the library, the code coverage reports for the three parallelism paradigms are also generated and stored in a separate repository dedicated to the code coverage analyses of the ParaMonte library. This repository is permanently located on GitHub at: https://www.cdslab.org/paramonte/notes/codecov/ and the reports are automatically and conveniently accessible from the ParaMonte documentation website.

The current set of 866 unit tests in version 1.5.0 release of the ParaMonte library collectively cover 47565 out of 48384 lines of code ($\sim 98\%$ line coverage) and 4467 out of 4476 procedures ($\sim 100\%$ function coverage) in the library's kernel for the three serial, MPI, and Coarray parallelism paradigms.

Funding

The development cost of ParaMonte is estimated over 1 million dollars by Open Hub. Considering the presence of multiple high-level dynamic languages such as MATLAB, Python, and R which together comprise approximately 40% of the repository, an estimate of $500 - 600$ thousand dollars seems more realist for the development costs of the library's kernel routines. The current state of the library has resulted from multiple years of continuous development predominantly by the core developers of the library as graduate student, postdoc, and faculty in their free times. The project was also partially supported in its final stages of development by the Peter O'Donnell, Jr. Postdoctoral Fellowship awarded by the Oden Institute for Computational Engineering and Sciences at The University of Texas at Austin to the primary developer of the library, Amir Shahmoradi.

Documentation and Repository

Extensive documentation and examples in C, C++, Fortran (as well as other programming languages) are available on the documentation website of the library at: https://www.cdslab.org/paramonte/. The ParaMonte library is MIT-licensed and is permanently located and maintained at https://github.com/cdslaborg/paramonte.

A high number of comments in a codebase are frequently indicative of an organized well-documented project and considered a sign of a helpful and disciplined development team. As of January 2021, comments approximately make up 31% of the entire ParaMonte codebase. Currently, the Open Hub indexer service ranks the ParaMonte project's level of documentation as "impressive" and among the top 10% of all projects in the same category on Open Hub.

Acknowledgements

We thank the Texas Advanced Computing Center for providing the supercomputer time for testing and development of this library.

References

Bois, F. Y. (2009). GNU mcsim: Bayesian statistical inference for sbml-coded systems biology models. *Bioinformatics*, *25*(11), 1453–1454.

Fanfarillo, A., Burnus, T., Cardellini, V., Filippone, S., Nagle, D., & Rouson, D. (2014). OpenCoarrays: Open-source transport layers supporting coarray fortran compilers. In *Proceedings* *of the 8th international conference on partitioned global address space programming models* (pp. 1–11).

Haario, H., Laine, M., Mira, A., & Saksman, E. (2006). DRAM: Efficient adaptive mcmc. *Statistics and computing*, *16*(4), 339–354.

Kumbhare, S., & Shahmoradi, A. (2020). MatDRAM: A pure-MATLAB Delayed-Rejection Adaptive Metropolis-Hastings Markov Chain Monte Carlo Sampler. *arXiv e-prints*, arXiv:2010.04190.

Kumbhare, S., & Shahmoradi, A. (2020). Parallel adapative monte carlo optimization, sampling, and integration in c/c++, fortran, matlab, and python. *Bulletin of the American Physical Society*.

Metropolis, N., & Ulam, S. (1949). The monte carlo method. *Journal of the American statistical association*, *44*(247), 335–341.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

Osborne, J. A., Shahmoradi, A., & Nemiroff, R. J. (2020). A multilevel empirical bayesian approach to estimating the unknown redshifts of 1366 batse catalog long-duration gamma-ray bursts.

Osborne, J. A., Shahmoradi, A., & Nemiroff, R. J. (2020). A Multilevel Empirical Bayesian Approach to Estimating the Unknown Redshifts of 1366 BATSE Catalog Long-duration Gamma-Ray Bursts, *903*(1), 33. doi:10.3847/1538-4357/abb9b7

Prudencio, E., & Schulz, K. (2012). The parallel C++ statistical library queso: Quantification of uncertainty for estimation, simulation and optimization. In M. Alexander, P. D'Ambra, A. Belloum, G. Bosilca, M. Cannataro, M. Danelutto, B. Martino, et al. (Eds.), *Euro-par 2011: Parallel processing workshops*, Lecture notes in computer science (Vol. 7155, pp. 398–407). Springer Berlin Heidelberg. ISBN: 978-3-642-29736-6

Shahmoradi, A. (2013). A multivariate fit luminosity function and world model for long gamma-ray bursts. *The Astrophysical Journal*, *766*(2), 111.

Shahmoradi, A. (2013). Gamma-Ray bursts: Energetics and Prompt Correlations. *arXiv e-prints*, arXiv:1308.1097.

Shahmoradi, A., & Bagheri, F. (2020). ParaDRAM: A cross-language toolbox for parallel high-performance delayed-rejection adaptive metropolis markov chain monte carlo simulations. *arXiv preprint arXiv:2008.09589*.

Shahmoradi, A., & Bagheri, F. (2020a). ParaDRAM: A Cross-Language Toolbox for Parallel High-Performance Delayed-Rejection Adaptive Metropolis Markov Chain Monte Carlo Simulations. *arXiv e-prints*, arXiv:2008.09589.

Shahmoradi, A., & Bagheri, F. (2020b, August). ParaMonte: Parallel Monte Carlo library.

Shahmoradi, A., & Nemiroff, R. (2014). Classification and energetics of cosmological gamma-ray bursts. In *American astronomical society meeting abstracts# 223* (Vol. 223).

Shahmoradi, A., & Nemiroff, R. J. (2015). Short versus long gamma-ray bursts: A comprehensive study of energetics and prompt gamma-ray correlations. *Monthly Notices of the Royal Astronomical Society*, *451*(1), 126–143.

Shahmoradi, A., & Nemiroff, R. J. (2019). A Catalog of Redshift Estimates for 1366 BATSE Long-Duration Gamma-Ray Bursts: Evidence for Strong Selection Effects on the Phenomenological Prompt Gamma-Ray Correlations. *arXiv e-prints*, arXiv:1903.06989.

Shahmoradi, A., Bagheri, F., & Osborne, J. A. er. (2020). Fast fully-reproducible serial/parallel

Monte Carlo and MCMC simulations and visualizations via ParaMonte::Python library. *arXiv e-prints*, arXiv:2010.00724.