# ParaMonte: An Efficient Serial/Parallel MCMC Library

Parvat Sapkota[1], Joshua Osborne[1], Shashank Kumbhare[1], Fatemeh Bagheri[1] ,and Amir Shahmoradi[1,2]
[1]Department of Physics, College of Science, The University of Texas, Arlington, TX 76019, USA
[2]Data Science Program, College of Science, The University of Texas, Arlington, TX 76019, USA

## Abstract

The scientific inference is a multistep process requiring observational data from which a model/hypothesis is derived. The parameters of this physical model then have to be tuned to more accurately represent data in a process known as model calibration. This calibrated model is then validated and is finally used to predict different quantities of interest. The most fundamental tool for model calibration and uncertainty quantification is the Markov Chain Monte Carlo (MCMC). While existing packages achieve many of the goals of the MCMC simulations, none currently addresses all critical aspects of an MCMC simulation. For instance, packages are frequently limited to only one programming language environment, perform serial or parallel simulations, or lack restart functionality. We present ParaMonte, a generic user-friendly, high performance Monte Carlo simulation toolbox for serial and parallel Monte Carlo simulations accessible from multiple programming languages. ParaMonte features automatically-enabled restart functionality of all simulations in serial or parallel and comprehensive post-processing and visualization of the simulation results. This package is available to the public under the MIT license from its permanent repository: https://github.com/cdslaborg/paramonte.

## Introduction

During model calibration, a major task is to find the best-fit model parameters given a mathematical objective function which requires a)Parameter tuning b) Uncertainty quantification, and c) Model selection.

The Markov Chain Monte Carlo (**MCMC**), in particular, the Metropolis-Hastings (**MH**) algorithm, is among the most popular tools to achieve the aforementioned inference goals. However, the traditional MH algorithm has fundamental limitations that make its usage inefficient in practice.

## ParaMonte

Existing Monte Carlo simulation packages do not address all critical aspects of an MCMC simulation. Some limitations include: 1) Only serial or parallel calculations, 2) Serving users of one particular programming language only, and 3) Significant dependencies on external libraries.

To solve these outstanding issues, we have developed a simulation toolbox for serial and parallel Monte Carlo simulations, known as ParaMonte [1] with the following designs in mind: **1)** Full automation, **2)** Interoperability, **3)** High-Performance, **4)** Parallelizability, **5)** Zero external-library dependencies, **6)**Fully-deterministic reproducibility and automatically-enabled restart functionality, and **7)** Comprehensive-reporting and post-processing.

The ParaMonte library aims to implement the following Monte Carlo algorithms. This poster focuses on the ParaDRAM sampler of the ParaMonte library.

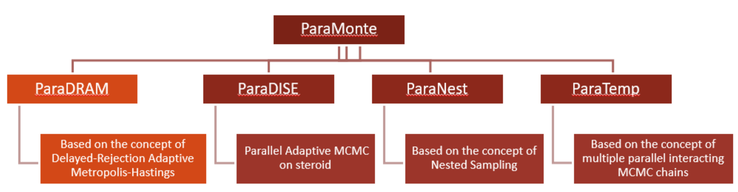
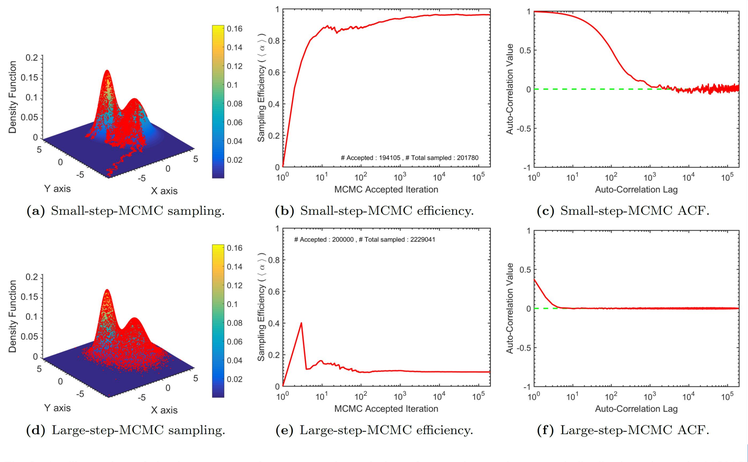



Fig1: Current and future algorithms in the ParaMonte library. Orange box refers to the algorithm that is currently present. Dark red box refers to the algorithm that are proposed for future development.

## Problem with Traditional MCMC Samplers

Traditional MCMC algorithms such as the Metropolis-Hastings method have a significant drawback: They frequently require manual tuning of the free parameters of the algorithm for every simulation to ensure fast convergence of resulting Markov chain to the target density function.




(a) Small-step-MCMC sampling. (b) Small-step-MCMC efficiency. (c) Small-step-MCMC ACF.

(d) Large-step-MCMC sampling. (e) Large-step-MCMC efficiency. (f) Large-step-MCMC ACF.

Fig 2: An illustration of the importance of an appropriate choice of step size and proposal distribution shape in MCMC simulations. Plots (a),(b),(c) shows MCMC sample, evolution of the efficiency of the MCMC simulation, and autocorrelation function(ACF) of the chain of uniquely-sampled states for small-step sizes. (d), (e), (f) represent the same quantities but for the large-step-size.

**Efficiency:**
The efficiency(<α>) of the MCMC simulation is given by the formula:

$$\langle \alpha \rangle = \frac{\text{the number of accepted MCMC proposed moves}}{\text{the total number of accepted and rejected proposed moves}}$$

Accordingly, for the small step-size, the number of rejected proposed moves is low, thus the sampling efficiency will be high. But, for large step-size, the number of rejected proposed moves is high, thus the sampling efficiency will be low.

**Auto-Correlation Value:**
The autocorrelation value simply demonstrates how good or bad are the mixing results. Because of the small-step size, new points will be highly correlated with each other thus showing poor mixing results. However, for large-step size, new points will not be highly correlated with each other thus demonstrating superior mixing results.

Thus, considering both efficiency and auto-correlation value, we see that we cannot take sizes either in small-steps or in large-steps as both will be inefficient. As such, we need to find the optimal step-size situated between those two.

## The ParaDRAM Algorithm

The automation of the tuning was successfully brought through the algorithm called Delayed-Rejection Adaptive Metropolis MCMC (**DRAM**).

The ParaDRAM algorithm [2][3] in the ParaMonte package attempts to address the existing algorithmic and performance deficiencies in DRAM algorithm by providing a parallel implementation of the algorithm and offering diagnostic checks that ensure the ergodicity and reversibility of the DRAM algorithm in practice.

## Adaptation of proposal distribution

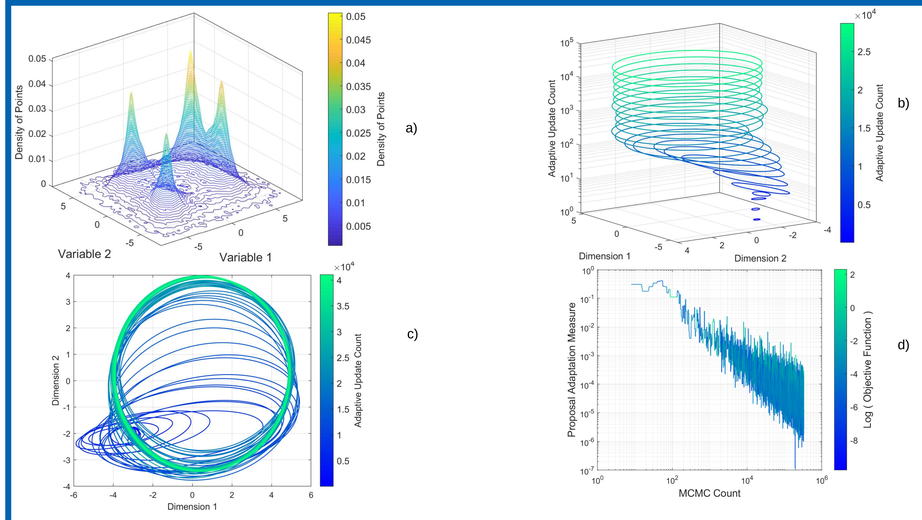



Fig: 3a) 3D contour map of ParaDRAM simulation output for a certain function 3b) 3D dynamic adaptation for it. 3c) 2D dynamic adaptation for it. 3d) Diminishing adaptation of the proposal distribution

Continuous adaptation of the proposal distribution of the adaptive MCMC samplers is an issue. But, the research indicates that as long as the adaptation of the Markov chain decreases throughout the simulation, the convergence is guaranteed.
Accordingly, although there are other methods, Shahmoradi & Bagheri[2] have also introduced a new technique which helps to monitor these adaptations.

## Parallelization of the Sampler

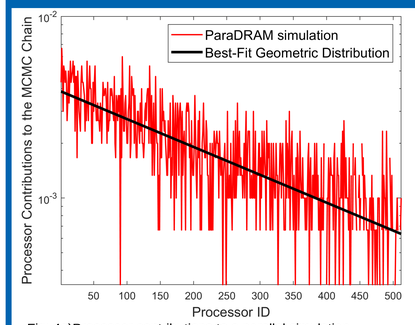



Fig: 4a)Processor contributions to a parallel simulation

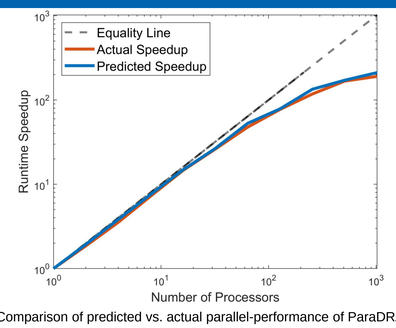



Fig: 4b) Comparison of predicted vs. actual parallel-performance of ParaDRAM simulations

The sampler in the ParaMonte package supports two modes of parallelism[2]:
a)Fork-join Parallelism & b)Perfect Parallelism.
Figure (2a) shows that as the number of processors is increased, the contributions from each processor decreases. Figure (2b) illustrates that as the numbers of processors are increased, the performance also increases thus demonstrating the scalability of this package.

## References

[1]Shahmoradi, A., Bagheri, F., & Osborne, J. A. (2020). Fast fully-reproducible serial/parallel Monte Carlo and MCMC simulations and visualizations via ParaMonte:: Python library. arXiv preprint arXiv:2010.00724.
[2]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
[3] Kumbhare, S., & Shahmoradi, A. (2020). MatDRAM: A pure-MATLAB Delayed-Rejection Adaptive Metropolis-Hastings Markov Chain Monte Carlo Sampler. arXiv preprint arXiv:2010.04190.