ParallelKDE.jl: A Package for Highly Parallel Kernel Density Estimation

Christian Sustay Martinez[1], **Martin Zacharias**[1], and **Patrick K. Quoika**[1¶]

1 Center for Functional Protein Assemblies, School of Natural Sciences, Technical University of Munich, Ernst-Otto-Fischer-Str. 8, Garching 85748, Germany ¶ Corresponding author

DOI: 10.21105/joss.09616

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @hisacro
- @mantzaris

Submitted: 05 November 2025
Published: 06 March 2026

Summary

Kernel density estimation (KDE) is a valuable tool in exploratory analysis, simulation, and probabilistic modeling across the sciences. However, its susceptibility to the curse of dimensionality can make routine KDE prohibitively slow on large, high-dimensional datasets. A central problem in KDE is the choice of the bandwidth of the kernels, which is oftentimes solved by applying an empiric rule-of-thumb (ROT). Furthermore, estimation of non-normal distributions is sometimes approached through non-constant bandwidths—so called adaptive bandwith—which adds an additional layer of complication. For additional background on KDE and bandwidth selection, see Silverman (1986), Scott (1992), Wand & Jones (1995) and Jones et al. (1996).

`ParallelKDE.jl` provides a fast, open-source KDE toolkit in Julia (Bezanson et al., 2017) that provides extensible infrastructure to support a variety of methods with serial, multi-threaded and GPU implementations. The package includes (i) a highly parallel implementation of established KDE rules-of-thumb (ROTs) as well as (ii) an implementation of GradePro, a new algorithm described in Sustay Martinez et al. (2025), engineered here for serial, threaded, and CUDA backends. A lightweight Python wrapper, `ParallelKDEpy` exposes the same functionality to Python workflows. Our goal is to make robust density estimation practical for large sample sizes of moderate dimensionality through careful parallelization of, mainly, grid-based KDE estimators.

Statement of need

Many researchers rely on Python implementations of KDE. Popular examples include SciPy's `gaussian_kde` (SciPy developers, 2025; Virtanen et al., 2020) with automatic bandwidth selection via ROTs; scikit-learn's `KernelDensity` (Pedregosa et al., 2011; Scikit-learn developers, 2025), which also supports ROTs; statsmodels' (Seabold & Perktold, 2010) `KDEUnivariate` and `KDEMultivariate` estimators with ROTs and cross-validated bandwidths (Statsmodels Developers, 2024); KDEpy's (Odland, 2018) `FFTKDE` with ROTs and plug-in rules (Sheather & Jones, 1991); and X-Entropy (Kraml et al., 2021), which implements a plug-in rule. While mature and widely used, these tools are CPU-centric and can become slow as data grow in size and dimension, especially if many evaluations are needed in downstream tasks.

`ParallelKDE.jl` addresses this gap with:

1. A unified device interface so that the estimations can be executed on serial, threads (via Julia's `Threads.@threads`), or GPU (e.g. via `CUDA.jl`) without changing user code;
2. Parallelized calculation of FFT-based KDE with ROTs and GradePro;

3. A reproducible benchmark comparing against SciPy, scikit-learn, statsmodels and KDEpy under matched estimation tasks.

Performance benchmarks

To demonstrate that our approach—moving density estimation to parallel execution or parallel hardware—produces marked acceleration compared with standard CPU-based libraries, we performed an extensive benchmark. We decided to carry it out in Python, since the most popular KDE tools are provided for that ecosystem. To ensure consistent interface across methods, we ran all experiments through our lightweight wrapper `ParallelKDEpy`. Naturally, our tool can conveniently be used directly in Julia, too. We evaluated different methods on synthetic data drawn from 1D and 2D standard normal distributions, generated with SciPy. For compatibility with the existing Python tools, we executed CPU baselines single-threaded; in addition, we benchmarked our CUDA implementation to exemplify the acceleration achievable through parallel execution. To reproduce the benchmark, please refer to the corresponding section in the `README.md`.

The results are shown in Figure 1 and Figure 2. Our implementations in `ParallelKDE` are particularly fast for high-resolution grids and/or large datasets. Additionally, our novel method, GradePro, achieves high accuracy on diverse distributions. While the CPU version is slower than several 1D baselines and comparable in 2D, its performance is still competitive and the extra computation yields better accuracy. In CUDA, the method's parallelism is exposed, making it faster and produces superior scaling with problem size. A detailed introduction of GradePro along with an extensive investigation of its accuracy is provided in Sustay Martinez et al. (2025).

Further comparison of GradePro with other methods is in progress. In the future, we aim to provide general recommendations for users on the optimal choice of bandwidth selection methods for different use-cases. Here, we focus on the software engineering aspects, which enable our fast implementation.

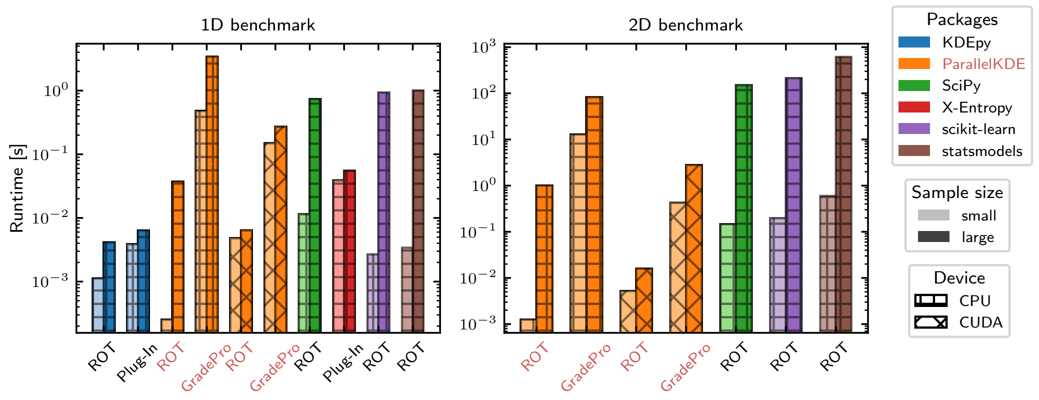


Figure 1: Benchmark of common KDE packages and their estimators—rule-of-thumb (ROT), plug-in, and GradePro—along with estimators in ParallelKDE at different sample sizes. The estimations were performed on data drawn from a standard normal distribution. In 1D, 100 and 100,000 samples were used with a grid of 500 points. In 2D, 1,000 and 1,000,000 samples were used with a grid of 100 points per dimension. Reported runtimes are averages over 10 repetitions. Hardware: Intel Core i7-6700 (single-threaded) and NVIDIA GTX 1080.

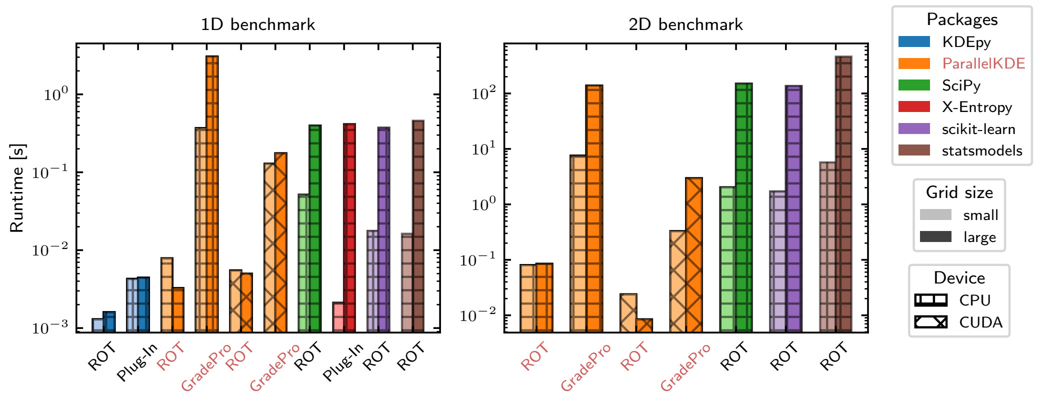


Figure 2: Benchmark of common KDE packages and their estimators along with estimators in ParallelKDE at different grid sizes. The estimations were performed on data drawn from a standard normal distribution. In 1D, 100 and 2,500 grid points were used with 10,000 samples. In 2D, 33 and 300 grid points per dimension were used with 100,000 samples. Reported runtimes are averages over 10 repetitions. Hardware: Intel Core i7-6700 (single-threaded) and NVIDIA GTX 1080.

Functionality

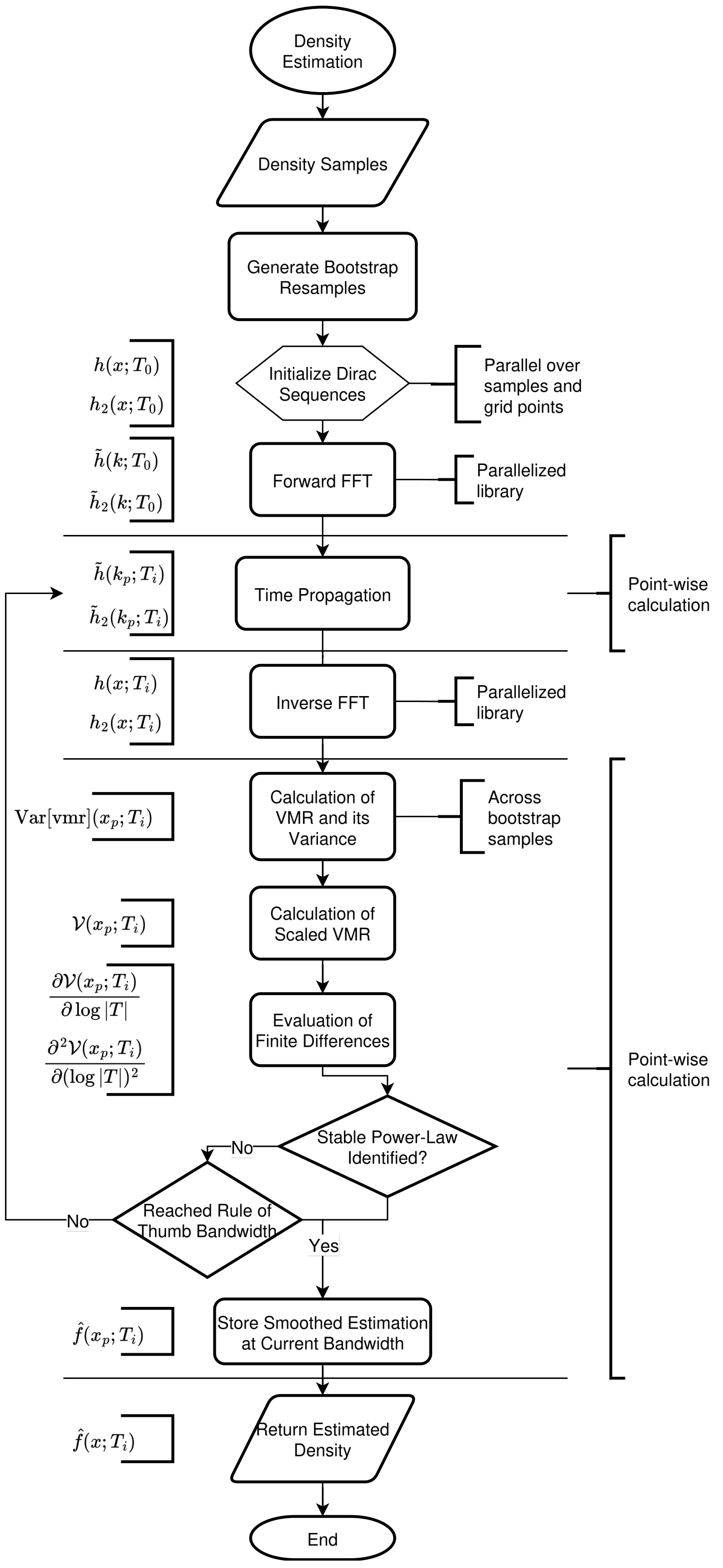
Design and architecture. `ParallelKDE.jl` is organized around a minimal device interface that isolates parallelism from algorithmic logic. We ship three backends:

- Serial CPU for portability and determinism;
- Threaded CPU using Julia's built-in multithreading;
- CUDA GPU via `CUDA.jl` (Besard, Foket, et al., 2019; Besard, Churavy, et al., 2019).

Implemented methods. The package includes (a) KDE with parallel ROTs, namely Scott and Silverman, for rapid baselines and (b) GradePro, engineered to perform rapid adaptive KDE in multiple dimensions. This implementation effectively performs independent grid-point-wise density estimation using FFT (Frigo & Johnson, 2005). Throughout the routines, the necessary memory is reduced by reusing allocated memory, without damaging performance. The serial version serves as a reference; the threaded version parallelizes routines applied over the bootstrap re-samples (which are required by the method); the CUDA routines provide parallelization at each estimated grid-point. Each implemented method naturally exposes all its parameters up to the user. Convenient defaults are set where reasonable. Figure 3 shows a flowchart of the implemented algorithm.

Reproducibility and testing. We include timing benchmarks to illustrate the speed gains obtained from parallelizable KDE algorithms.

Python wrapper. The companion package `ParallelKDEpy` provides, with the help of `PythonCall.jl` (Rowley, 2022), Python bindings to `ParallelKDE.jl`, offering identical estimators and device selection for seamless integration in Python-based workflows.



Density Estimation
Density Samples
Generate Bootstrap Resamples
$h(x;T_0)$
$h_2(x;T_0)$
Initialize Dirac Sequences
Parallel over samples and grid points
$\tilde{h}(k;T_0)$
$\tilde{h}_2(k;T_0)$
Forward FFT
Parallelized library
$\tilde{h}(k_p;T_i)$
$\tilde{h}_2(k_p;T_i)$
Time Propagation
Point-wise calculation
$h(x;T_i)$
$h_2(x;T_i)$
Inverse FFT
Parallelized library
$\mathrm{Var}[\mathrm{vmr}](x_p;T_i)$
Calculation of VMR and its Variance
Across bootstrap samples
$\mathcal{V}(x_p;T_i)$
Calculation of Scaled VMR
$\frac{\partial \mathcal{V}(x_p;T_i)}{\partial \log|T|}$
$\frac{\partial^2 \mathcal{V}(x_p;T_i)}{\partial (\log|T|)^2}$
Evaluation of Finite Differences
Stable Power-Law Identified?
No
Reached Rule of Thumb Bandwidth
No
Yes
Point-wise calculation
$\hat{f}(x_p;T_i)$
Store Smoothed Estimation at Current Bandwidth
$\hat{f}(x;T_i)$
Return Estimated Density
End

Figure 3: Flowchart of the parallelizable point-wise density estimation algorithm.

Acknowledgements

We thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—Project number DFG-INST 95/1610-1 FUGG—for funding this research.

References

Besard, T., Churavy, V., Edelman, A., & Sutter, B. D. (2019). Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software*, *132*, 29–46. https://doi.org/10.1016/j.advengsoft.2019.02.002

Besard, T., Foket, C., & De Sutter, B. (2019). Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *30*(4), 827–841. https://doi.org/10.1109/tpds.2018.2872064

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Frigo, M., & Johnson, S. G. (2005). The Design and Implementation of FFTW3. *Proceedings of the IEEE*, *93*(2), 216–231. https://doi.org/10.1109/JPROC.2004.840301

Jones, M. C., Marron, J. S., & Sheather, S. J. (1996). A Brief Survey of Bandwidth Selection for Density Estimation. *Journal of the American Statistical Association*, *91*(433), 401–407. https://doi.org/10.1080/01621459.1996.10476701

Kraml, J., Hofer, F., Quoika, P. K., Kamenik, A. S., & Liedl, K. R. (2021). X-Entropy: A Parallelized Kernel Density Estimator with Automated Bandwidth Selection to Calculate Entropy. *Journal of Chemical Information and Modeling*, *61*(4), 1533–1538. https://doi.org/10.1021/acs.jcim.0c01375

Odland, T. (2018). *Tommyod/KDEpy: Kernel Density Estimation in Python*. https://doi.org/10.5281/ZENODO.2392268

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

Rowley, C. (2022). *PythonCall.jl: Python and Julia in harmony*. https://github.com/JuliaPy/PythonCall.jl

Scikit-learn developers. (2025). *KernelDensity - scikit-learn 1.7.2 Documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html#sklearn.neighbors.KernelDensity

SciPy developers. (2025). *Scipy.stats.gaussian_kde - SciPy v1.16.2 Documentation*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html

Scott, D. W. (1992). *Multivariate Density Estimation: Theory, Practice, and Visualization* (1st ed.). Wiley. https://doi.org/10.1002/9780470316849

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. *Proceedings of the 9th Python in Science Conference*. https://doi.org/10.25080/majora-92bf1922-011

Sheather, S. J., & Jones, M. C. (1991). A Reliable Data-Based Bandwidth Selection Method for Kernel Density Estimation. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *53*(3), 683–690. https://doi.org/10.1111/j.2517-6161.1991.tb01857.x

Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. Chapman; Hall.

ISBN: 978-0-412-24620-3

Statsmodels Developers. (2024). *Statsmodels v0.14.4 Documentation*. https://www.statsmodels.org/stable/generated/statsmodels.nonparametric.kernel_density.KDEMultivariate.html

Sustay Martinez, C., Quoika, P. K., & Zacharias, M. (2025). Novel Rapid Approach for Adaptive Gaussian Kernel Density Estimation: Gridpoint-wise Propagation of Anisotropic Diffusion Equation. *Researchsquare*. https://doi.org/10.21203/rs.3.rs-8038701/v1

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Van Der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wand, M. P., & Jones, M. C. (1995). *Kernel smoothing* (1. ed). Chapman & Hall. ISBN: 978-0-429-17059-1