The Journal of Open Source Software

TNC: Distributed Tensor Network Contractions in Rust

Manuel Geiger [1], **Qunsheng Huang** [1], **and Christian B. Mendl** [1,2]

1 School for Computation, Information and Technology, Technical University of Munich, Germany **2** Institute for Advanced Study, Technical University of Munich, Germany

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Vincent Knight

Reviewers:

- @Luthaf
- @emapuljak

Submitted: 09 September 2025
Published: unpublished

Summary

TNC is a Rust-based library for efficient contraction of tensor networks, a key technique for classical simulation of quantum circuits. Designed for distributed-memory environments common in high-performance computing (HPC), TNC partitions tensor networks to enable parallel contraction across multiple nodes. While optimized for quantum circuit simulation, TNC is general enough to handle arbitrary tensor networks, combining high performance with Rust's memory safety guarantees.

Statement of need

With the rapid advancements in quantum computing, ever larger quantum circuits are being explored, which require more computing resources. Since real quantum hardware is often not yet available and has to fight problems such as high error rates, research on efficient classical simulation methods is crucial to bridge this gap. Tensor networks have been shown to be a viable tool for classical simulation, disproving even two major claims of quantum advantage, i.e., the realization of an algorithm on real quantum hardware which would be infeasible to simulate on classical hardware (Pan et al., 2022; Patra et al., 2024).

However, existing libraries for tensor network contractions often only consider shared-memory parallelism (Pfeifer et al., 2015), are not open-source (Bayraktar et al., 2023), or are written in Python (Gray & Kourtis, 2021) and hence less suited for HPC. Furthermore, parallelization is usually done by a technique called *slicing*, which incurs computational overhead. TNC addresses these limitations by targeting distributed-memory systems, being fully open-source, implemented in Rust, and employing partitioning rather than slicing for parallelization.

Features

The user can construct quantum circuits using the library's API or import them from code written in the widely-used OpenQASM2 language (Cross et al., 2017). From a given circuit, the user can then generate tensor networks that compute the expectation value, the amplitudes to specific measurement outcomes, or the full state vector. Furthermore, random tensor networks can be created as well, e.g., in the form of the original Sycamore experiment (Arute et al., 2019).

For contraction, the library partitions tensor networks based on the number of available MPI ranks. It obtains an initial partitioning using the multi-graph partitioning library KaHyPar (Andre et al., 2018). Since the resulting partitionings are often not optimal regarding the computational cost required to contract them, our library features different algorithms to improve this initial partitioning, such as greedy rebalancing, simulated annealing, and a genetic algorithm.

The library can find contraction paths, which specify the order of contraction operations, using different methods from the cotengra library (Gray & Kourtis, 2021). It then performs the individual contractions using MKL for matrix-matrix multiplications and the hptt library (Springer et al., 2017) for data transposition.

In a recent publication, we showed that partitionings optimized with the simulated annealing method have contraction costs that are competitive with state-of-the-art methods (Geiger et al., 2025).

Acknowledgements

We acknowledge the funding received for the MUNIQC-SC initiative under funding number 13N16191 from the VDI technology center as part of the German BMBF program. The work is also supported by the Bavarian state government via the BayQS project with funds from the Hightech Agenda Bayern, and via the Munich Quantum Valley, section K7, with funds from the Hightech Agenda Bayern Plus.

References

Andre, R., Schlag, S., & Schulz, C. (2018). Memetic multilevel hypergraph partitioning. *Proceedings of the Genetic and Evolutionary Computation Conference*, 347–354. https://doi.org/10.1145/3205455.3205475

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G. S. L., Buell, D. A., Burkett, B., Chen, Y., Chen, Z., Chiaro, B., Collins, R., Courtney, W., Dunsworth, A., Farhi, E., Foxen, B., … Martinis, J. M. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, *574*(7779), 505–510. https://doi.org/10.1038/s41586-019-1666-5

Bayraktar, H., Charara, A., Clark, D., Cohen, S., Costa, T., Fang, Y.-L. L., Gao, Y., Guan, J., Gunnels, J., Haidar, A., Hehn, A., Hohnerbach, M., Jones, M., Lubowe, T., Lyakh, D., Morino, S., Springer, P., Stanwyck, S., Terentyev, I., … Yamaguchi, T. (2023). cuQuantum SDK: A high-performance library for accelerating quantum science. *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, *01*, 1050–1061. https://doi.org/10.1109/QCE57702.2023.00119

Cross, A. W., Bishop, L. S., Smolin, J. A., & Gambetta, J. M. (2017). *Open quantum assembly language*. https://doi.org/10.48550/ARXIV.1707.03429

Geiger, M., Huang, Q., & Mendl, C. B. (2025). *Optimizing tensor network partitioning using simulated annealing*. https://arxiv.org/abs/2507.20667

Gray, J., & Kourtis, S. (2021). Hyper-optimized tensor network contraction. *Quantum*, *5*, 410. https://doi.org/10.22331/q-2021-03-15-410

Pan, F., Chen, K., & Zhang, P. (2022). Solving the sampling problem of the sycamore quantum circuits. *Physical Review Letters*, *129*, 090502. https://doi.org/10.1103/PhysRevLett.129.090502

Patra, S., Jahromi, S. S., Singh, S., & Orús, R. (2024). Efficient tensor network simulation of IBM's largest quantum processors. *Physical Review Letters*, *6*, 013326. https://doi.org/10.1103/PhysRevResearch.6.013326

Pfeifer, R. N. C., Evenbly, G., Singh, S., & Vidal, G. (2015). *NCON: A tensor network contractor for MATLAB*. https://arxiv.org/abs/1402.0939

Springer, P., Su, T., & Bientinesi, P. (2017). *HPTT: A high-performance tensor transposition C++ library*. arXiv. https://doi.org/10.48550/ARXIV.1704.04374