Syndrilla: Simulating Decoders for Quantum Error Correction using PyTorch

Yanzhang Zhu*, Chen-Yu Peng†, Yun Hao Chen†, Siyuan Niu*, Yeong-Luh Ueng†, Di Wu*
*ECE Department, University of Central Florida †EE Department, National Tsing Hua University
{yanzhang.zhu, siyuan.niu, di.wu}@ucf.edu, {ss113061597, sdfg0424}@gapp.nthu.edu.tw, ylueng@ee.nthu.edu.tw

***Abstract*—Quantum error correction (QEC) via error decoding is essential towards fault-tolerant quantum computing, but extremely costly to simulate. However, existing simulators are often CPU-based and suffer from severe performance bottlenecks, especially when scaling to extensive simulations with large code distances or low physical error rate. In this work, we introduce Syndrilla, a PyTorch-based across-platform QEC simulation framework. The framework is highly modular, allowing flexible integration and customization of the error model, syndrome model, decoding configuration (algorithm and data format), and logical check type. Experimental results show that Syndrilla achieves $10\times \sim 20\times$ speedup over CPU, when running on both AMD and NVIDIA GPUs, and decoding data format does not degrade accuracy, demonstrating the practicality and efficiency of Syndrilla to instigate future QEC research.**

***Index Terms*—Quantum error correction, decoding simulation, PyTorch, GPU**

I. Introduction

Fault-tolerant quantum computing (FTQC) is on the horizon, given the rampaging investigation in quantum error correction (QEC). Given the advancement in lowering physical qubit errors, QEC is poised to eliminate quantum noises at the logical qubit level, unleashing the exponential computational power. During the transition from the Noisy Intermediate-Scale Quantum (NISQ) to FTQC era, quantum error suppression and mitigation techniques also play a key role. While algorithm-focused toolchains such as Qiskit have matured, a significant gap remains on the QEC side: There is a lack of easy-to-use open-source simulators specifically designed for QEC. The performance of existing CPU-based systems becomes a major bottleneck when scaling to large code distances or low physical error rates, limiting their practical utility in research and development.

To address this performance bottleneck, we propose to accelerate QEC simulation via the mature hardware and software stacks of artificial intelligence, which have been optimized for decades. In this paper, we develop Syndrilla, A PyTorch-based numerical simulator for decoders in QEC. Syndrilla is highly modular, allowing on-demand and independent customization of its modules, including error, syndrome, decoder, logical check, and metric. Backed by PyTorch, Syndrilla is capable of running on any platform that supports PyTorch, such as CPU, GPU, or even AI accelerators. Compared to CPU-based simulators, Syndrilla achieves up to $10\times - 20\times$ speedup when using high-performance GPUs from different vendors, such as AMD and NVIDIA, and large code distances.

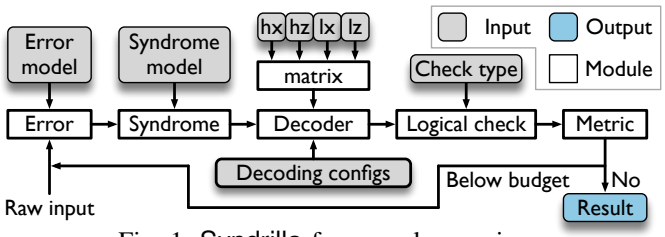


Fig. 1: Syndrilla framework overview.

II. Background

A. BPOSD Algorithm

Belief propagation (BP) exhibits outstanding performance in classical LDPC codes. However, BP fails to converge in QEC due to quantum degeneracy, where many distinct physical error patterns are logically equivalent under the stabilizer formalism. To address this, Ordered Statistics Decoding (OSD) is introduced as a post-processing step. BPOSD achieves robust performance across a wide range of QEC code families [1].

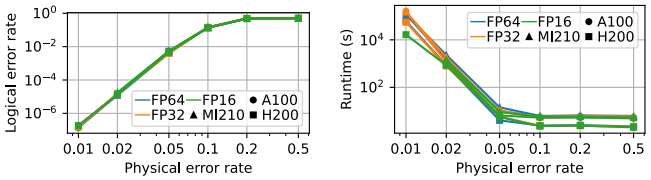
B. Existing QEC Frameworks

Existing decoders, like BPOSD [1], union-find [2], and minimum-weight perfect matching [3], are available but limited in usability and efficiency, since they require ad-hoc setup and run slowly on CPU. Other simulators, like Google's qsim [4] and Stim [5], remain grounded in quantum-circuit simulation, rather than decoding numerical errors in QEC.

III. Framework

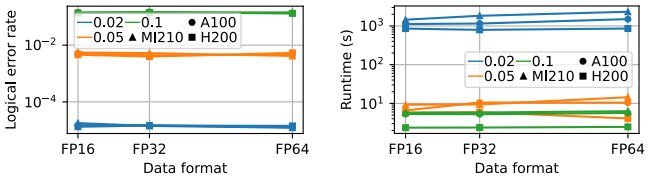
The full QEC decoding process can be separated into four stages: data encoding, syndrome measurement, error decoding, and error correction. Syndrilla focuses on the process from encoding to error decoding, leaving out the error correction, which happens on the physical quantum machines. Syndrilla virtualizes these concerned stages into five modules: error, syndrome, decoder, logical check, and metric, as shown in Figure 1. Each module allows independent, flexible customization to enable rapid prototyping of new algorithms. An error budget is set up, until which the simulation will terminate, since the exact number of input samples needed to report a stable logical error rate is indefinite beforehand. Syndrilla can also resume the simulation from checkpoints, if the error budget is not met in the last run.

Error module. In practical quantum systems, the error characteristics vary drastically across qubit technologies and hardware implementations. The error module injects errors into



(a) Accuracy (b) Runtime

Fig. 2: Comparison across GPUs.



(a) Accuracy (b) Runtime

Fig. 3: Comparison across data formats.

the raw inputs, based on the error model. This module takes in customized and generates data qubits with target error distributions. Currently, Syndrilla defaults to binary symmetric channel (BSC) [1].

Syndrome module. The syndrome module encodes the raw input with errors into a codeword and measures the corresponding syndrome. The syndrome can also have its own errors that model the imperfect measurement process. Current implementation assumes no syndrome measurement errors, but can easily support models beyond.

Decoder module. The decoder module allows arbitrary combinations of different local and global decoding algorithms to gain the best of both worlds, and currently Syndrilla supports BP+OSD. This module also configures the data type, e.g, FP64, during simulation for boosted simulation performance.

Matrix module. The matrix module loads parity check matrix and logical check matrix into the decoder, and offers flexibility in how users store these matrices. Currently, we support file formats of .alist from BPOSD [1], .npz from NumPy, and text.

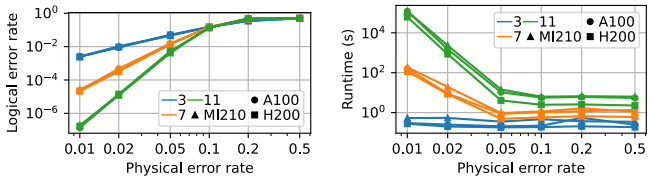
Logical check module. The logical check module concerns both X and Z checks for CSS codes, which can be configured per decoder.

Metric module. Syndrilla includes a dedicated metric module to output comprehensive evaluation statistics, including runtime, error rate, decoder invoke rate, etc.

IV. Evaluation

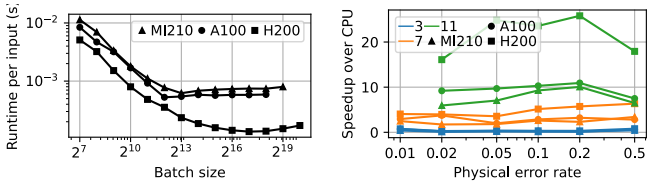
A. Experimental Setup

In the evaluation, we concern four hardware platforms: AMD Instinct MI210 GPU, NVIDIA A100 GPU, NVIDIA H200 GPU, and Intel i9-13900K CPU. We set the batch size during the simulation to 10000, unless otherwise stated. The simulation continues until 100 logical errors are reported.



(a) Accuracy (b) Runtime

Fig. 4: Comparison across code distances.



(a) Runtime across batch sizes (b) Runtime speedup

Fig. 5: Runtime across batch sizes and speedup over CPU.

B. Result Analysis

Figure 2 compares across different GPUs. It is observed that the simulation accuracy is very consistent across different GPUs and data formats. However, the runtime of H200 is significantly lower than that of MI210 and A100, which is decided by their peak throughput.

Figure 3 zooms into different data formats at different physical error rates. We conclude that data formats (almost) have no impact on accuracy, but significantly impact runtime, with FP16 being the fastest.

Figure 4 shows that different GPUs are consistent in accuracy across code distances. Regarding runtime, different GPUs also exhibit a similar ratio. Due to the sudden decrease of the logical error rate for distances 7 and 11, their runtime increases exponentially compared to distance 3.

Figure 5a compares the runtime per input sample across different batch sizes at a physical error of 0.5. Different GPUs have varying optimal runtimes, e.g., optimal batch sizes are 2^{13}, 2^{12}, and 2^{17} for MI210, A100, and H200.

Figure 5b compares the speedup of GPUs over CPU, and we observe that at large distances, GPUs exhibit significant speedup over CPU, $10\times \sim 20\times$.

V. Conclusion

This paper introduces Syndrilla, a highly modular QEC simulator to speed up decoding by up to $20\times$ on GPUs.

References

[1] J. Roffe, D. R. White, S. Burton, and E. T. Campbell, "Decoding across the quantum LDPC code landscape," *arXiv preprint arXiv:2005.07016*, 2020.

[2] C.-Y. Park and K. Meinerz, "Open-source c++ implementation of the union-find decoder," https://github.com/chaeyeunpark/UnionFind, 2020.

[3] Y. Wu and L. Zhong, "Fusion blossom: Fast mwpm decoders for qec," *arXiv preprint*, 2023.

[4] Q. A. team and collaborators, "qsim," Sep. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4023103

[5] C. Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, vol. 5, p. 497, Jul. 2021. [Online]. Available: https://doi.org/10.22331/q-2021-07-06-497