README.txt.pdf for the Quantum Tea submission to the Quantinuum challenge

Quantum Matter and Information group, University of Padova

June 2025

Workgroup: Francesco Pio Barone, Massimo Colombo, Alberto Coppi, Andrea De Girolamo, Asmita Datta, Guillermo Muñoz Menés, Marco Tesoro, Daniel Jaschke, Flavio Baccari, Simone Notarnicola, Davide Rattacaso, Ilaria Siloi, Darvin Wanisch, Simone Montangero

Contributors to the library: Flavio Baccari, Davide Bacilieri, Marco Ballarin, Francesco Pio Barone, Francesco Campaioli, Alberto Giuseppe Catalano, Giovanni Cataldi, Massimo Colombo, Alberto Coppi, Aurora Costantini, Asmita Datta, Andrea De Girolamo, Daniel Jaschke, Sven Benjamin Kožić, Giuseppe Magnifico, Carmelo Mordini, Guillermo Muñoz Menés, Simone Notarnicola, Alice Pagano, Luka Pavesic, Davide Rattacaso, Nora Reinić, Marco Rigobello, Simone Scarlatella, Ilaria Siloi, Marco Tesoro, Gianpaolo Torre, Darvin Wanisch, Lisa Zangrando

1 Hardware specifications

We used two Italian clusters, the INFN Padova HPC and the Leonardo HPC from CINECA. We used the former to run CPU-based simulations, while the latter was used for GPU-based ones. Each simulation runs on a single node of one of the two clusters. We report the details of the node configuration in the following table.

	node hardware		allocated resources per job		
computational device	CPU	GPU	RAM	CPU cores	GPU memory
GPU (default)	Intel Ice Lake Intel Xeon Platinum 8358	NVIDIA Ampere100 custom	60GB	8	68GB
CPU	AMD EPYC 9654	-	160GB	60	-

The GPU configuration corresponds to the base QoS (Quality of Service) of the Leonardo HPC Booster partition. The CPU configuration has been used on Padova's INFN HPC cluster. Both clusters have a job walltime of 24hrs. We use checkpoints to save the state of the simulation every 20 minutes. For simulations longer than the walltime, we restart the simulation from the last available checkpoint. The `simulation_time` of each circuit, if longer than 24hrs, is the sum of the time spent on the simulation across the several jobs that are needed to complete it.

Using GPUs was our preferred choice, but we used CPUs instead in those cases where the required RAM exceeded the GPU's capacity.

We set the number of cores in the CPU configuration according to the maximum number of cores available on the INFN cluster to maximize the parallelization. However, `numpy`, which we use as a backend for tensor computations on CPU, does not use all the available cores at full capacity. The numerical backend for GPU tensor operations is `cupy`.

2 Simulation algorithm

To run our simulations, we used the Quantum Matcha Tea library, available at www.quantumtea.it/. The simulation algorithm consists of initializing an MPS in the state $|0\rangle^{\otimes N}$ and applying the gates on the MPS. Even though our library supports many more tensor network ansätze, such as Tree Tensor Network (TTN) for high-dimensional quantum many-body systems or Tree Tensor Operators (TTO) for open systems, we have chosen to submit data on MPS only. Gates are applied following a TEBD-inspired approach. In fact, we distinguish three different cases:

- The single-qubit gates are directly contracted to the corresponding site of the MPS.

- Two-qubit gates that act on neighboring qubits can be applied directly to the MPS after an SVD decomposition. If the gate is not acting on neighboring qubits, we offer two options:
  1. Linearize the circuit, i.e., find an equivalent circuit that uses only gates acting on neighboring sites of the MPS. The linearization option involves the introduction of SWAP gates and is available through the Qiskit transpiler (see Sec. 5.1).
  2. Apply the gate via a long-range MPO (see Sec. 3.1).
- All > 2-qubit gates are applied via MPO (see Sec. 3.1).

3 Circuit format

We load the circuits via their qasm files for all the families but the quantum chemistry ones. Instead, we load the quantum chemistry circuits (`chemistry_uccsd`) and a few others (see the Table in Appendix A) directly from `/pytket_orig/`. Indeed, we implemented a mapping [1] to represent the Pauli Exponential Boxes as MPOs, circumventing expensive two-qubit gate decompositions.

3.1 Support of multiple qubit and long-range gates

Our MPS simulator supports n-qubit gates via MPOs. There are some gates whose MPO form is known and easy to write explicitly, such as $(n-1)$-times controlled gates - including the CNOT - and rotations. This is also the case of Pauli Exponential Boxes, for which we analytically construct the corresponding MPO without truncating their matrix representation, following the method described in Ref. [1].

In all other cases, we build the MPO starting from the $2^n \times 2^n$ matrix representation of the gate. First, we reshape the matrix into a $2n$-leg tensor, then we iteratively SVD-decompose it on each physical site.

After defining the MPO, we apply it to the MPS state. First, we canonize the MPS on the leftmost site to which the MPO is being applied. Then, we sweep towards the right-most site. For each physical site, we set the isometry center and then apply the corresponding local MPO with standard tensor contraction. To shift the isometry center, we use an SVD decomposition if a local MPO is to be applied on the next site, or with a QR decomposition otherwise.

4 Explored trade-offs

The simulations included in this submission have been obtained by tuning the parameters available in our simulator (see Sec. 5) to reach the **second best mirror fidelity band in the shortest simulation time**. This submission includes the circuits for which it was actually possible to reduce the mirror fidelity to values in the lower fidelity band. The circuits in the chemistry family always showed a mirror fidelity in the highest band, and thus, we excluded them from this submission. Also, we excluded those circuits for which the best mirror fidelity was in the lowest fidelity band.

4.1 Evaluation of the results

To certify the convergence of our numerical simulations, we repeated the simulation at different values of maximum bond dimension, measuring the mirror fidelity and the fidelity estimate, see Section 6. For increasing values of the maximum bond dimension, we observed the following typical behaviour:

- The mirror fidelity can be high (in the highest fidelity band) at small bond dimension values. This is due to the truncation, which trivially projects the state close to the initial one.
- Increasing the bond dimension, the mirror fidelity decreases as the dynamics is not trivial anymore, but the state poorly overlaps with the initial one after the round-trip circuit.
- Further increasing the bond dimension, the mirror fidelity increases, suggesting a correspondence between the convergence of the mirror fidelity and the convergence of fidelity at the end of the circuit.

Nevertheless, in some cases, the highest accessible bond dimension is not enough to reach the final stage of the mirror fidelity behavior. This reflects in a very low fidelity estimate, despite an order-of-magnitude higher mirror fidelity.

5 Simulation parameters

The values of the setting parameters used for each simulation are reported in the Table in Appendix A. We used the same setting parameters for simulating both the input circuit and its inverse.

5.1 Preprocessing parameters

When the input circuit is in qasm format, we exploit Qiskit transpiler features to optimize the circuit before passing it to the MPS simulation routine. The transpilation is controlled by the following parameters:

- `linearize` (default: False). The circuit gates are transpiled to achieve a linear connectivity of the qubits. This implies that any two-qubit gate that is not acting on non-neighboring qubits will be transpiled to a network of swap gates that bring the qubits next to each other, then the gate is applied. If linearize is off, then all long-range gates are applied via MPO, as described in Sec. 3.1.
- `optimization level` (default: 0). This parameter corresponds to the argument `optimization_level` of the `transpile()` routine of Qiskit. Quoting the Qiskit documentation, "higher levels generate more optimized circuits, at the expense of longer transpilation time." The optimization level ranges from 0 (no optimization) to 3 (heaviest optimization).

Additionally, we provide an optional `tensor compiler` optimization parameter, which is integrated in our software and is not part of Qiskit:

- `tensor compiler` (default: False). The tensor compiler maps the input circuit to a graph and contracts deterministically single-qubit gates to many-qubit gates. By doing so, the gate count is significantly reduced. In some cases, the tensor compiler reduces computational time while simultaneously improving mirror fidelity.

5.2 Convergence parameters

For the tensor network simulation, we exploit the following parameters:

- `bond dimension`. (default: 4). The maximum allowed bond dimension χ_{bd}. After each SVD, only the largest χ_{bd} singular values are kept. The maximum bond dimension has been changed extensively in our simulations to achieve the target mirror fidelity (see Table in Appendix A).
- `cut ratio` (default: 10^{-9}). The cutoff ratio η_{cr}, which truncates all the singular values w.r.t. the largest eigenvalue: if $|\lambda_i/\lambda_{\text{largest}}| < \eta_{cr}$, then λ_i is cut.
- `svd mode` (default: V). The svd mode specifies the numerical method used for the SVDs. In the following, the list of modes we used:
  - A, automatic. Some heuristic is run to choose the best mode for the algorithm.
  - V, gesvd. Stable but slow.
  - D, gesdd. Fast iterative method, although with some probability of failure.

6 Evaluation of inaccuracies: estimating the fidelity

The fidelity estimate is computed throughout the simulation as a function of the singular values that have been cut in each SVD operation [2]. Let $\lambda_i : i = 1, \ldots, N$ be the singular values returned from an SVD split at step k of the simulation. Let us stress that, as mentioned in Sec. 3.1, the application of a multi-qubit or long-range gate results in multiple SVD steps. Say that all the singular values with index greater than some cut index χ have been cut. Then, the corresponding fidelity cut of a single SVD step is:

$$\delta f_k = 1 - \frac{\sum_{i=\chi}^{N} \lambda_i^2}{\sum_{i=1}^{N} \lambda_i^2} \tag{1}$$

The cut χ is determined by the `cut bond dimension` χ_{bd} and the `cutoff ratio` η_{cr} (see Sec. 5).
The fidelity f of the state is approximated by the fidelity estimate f^*:

$$f^* \equiv \prod_k \delta f_k \approx f \ \leq 1 \tag{2}$$

where the product runs over all the SVDs required to simulate the circuit.

7 Software availability

As mentioned above, our software is publicly available at www.quantumtea.it/. The software version used for these benchmarks is currently under development, and it is hosted on a private branch. However, we periodically update the public branch as soon as we have a significant number of changes.

References

[1] Alberto Giuseppe Catalano. *Numerically efficient unitary evolution for Hamiltonians beyond nearest-neighbors*. 2024. arXiv: 2402.05198 [cond-mat.str-el]. URL: https://arxiv.org/abs/2402.05198.

[2] Yiqing Zhou, E. Miles Stoudenmire, and Xavier Waintal. "What Limits the Simulation of Quantum Computers?" In: *Phys. Rev. X* 10 (4 2020), p. 041038. DOI: 10.1103/PhysRevX.10.041038. URL: https://link.aps.org/doi/10.1103/PhysRevX.10.041038.

A Data summary for this submission

The simulations included in this submission have been obtained by tuning the parameters available in our simulator (see Sec. 5) to reach the **second best mirror fidelity band in the shortest simulation time**. This submission includes the circuits for which it was actually possible to reduce the mirror fidelity to values in the lower fidelity band. The circuits in the chemistry family always showed a mirror fidelity in the highest band, and thus, we excluded them from this submission. Also, we excluded those circuits for which the best mirror fidelity was in the lowest fidelity band.

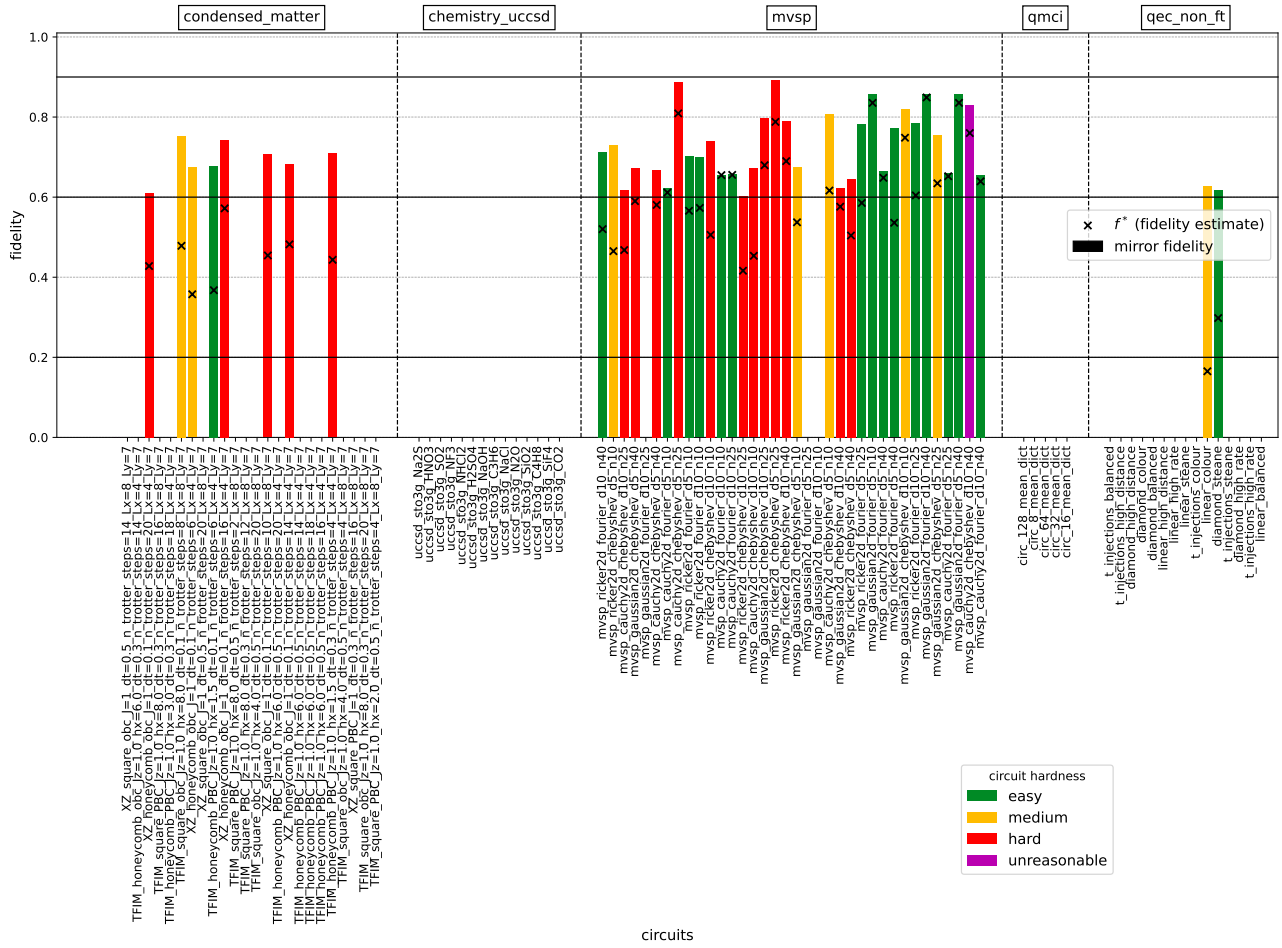

Figure 1: Mirror fidelity (colored bars) and fidelity estimate f^* (cross marker) of Eq. (2) for all the circuits included in this submission. The circuits are grouped by family. The colors of the bars represent the hardness of the circuit (either easy, medium, hard, or impossible), which is provided by the metadata of the challenge circuits.

In the following table, we summarize the parameters used for the circuits included in this submission.

circuit	format	device	linearize	optimize level	tensor compiler	bond dimension	cut ratio	SVD
TFIM_honeycomb_PBC_Jz=1.0_hx=1.5_dt=0.1_n_trotter_steps=6_Lx=4_Ly=7	qasm	gpu	True	1	False	16	1e-09	V
TFIM_honeycomb_PBC_Jz=1.0_hx=1.5_dt=0.3_n_trotter_steps=4_Lx=4_Ly=7	qasm	gpu	True	1	False	128	1e-09	V
TFIM_square_obc_Jz=1.0_hx=8.0_dt=0.1_n_trotter_steps=8_Lx=8_Ly=7	qasm	gpu	True	1	False	32	1e-09	V
XZ_honeycomb_obc_J=1_dt=0.1_n_trotter_steps=14_Lx=4_Ly=7	qasm	gpu	True	1	False	128	1e-09	V
XZ_honeycomb_obc_J=1_dt=0.1_n_trotter_steps=16_Lx=4_Ly=7	qasm	gpu	True	1	True	256	1e-09	V
XZ_honeycomb_obc_J=1_dt=0.1_n_trotter_steps=20_Lx=4_Ly=7	qasm	gpu	True	1	True	256	1e-09	V
XZ_honeycomb_obc_J=1_dt=0.1_n_trotter_steps=6_Lx=4_Ly=7	qasm	gpu	True	1	False	16	1e-09	V
XZ_square_obc_J=1_dt=0.1_n_trotter_steps=10_Lx=8_Ly=7	qasm	gpu	True	1	False	256	1e-09	V
diamond_steane	qasm	gpu	False	0	False	204	1e-10	V
linear_colour	qasm	gpu	False	0	False	500	1e-10	V
mvsp_cauchy2d_chebyshev_d10_n10	qasm	gpu	True	1	True	64	1e-09	V
mvsp_cauchy2d_chebyshev_d10_n25	qasm	cpu	True	0	True	80	1e-09	V
mvsp_cauchy2d_chebyshev_d10_n40	qasm	gpu	True	1	False	320	1e-09	V
mvsp_cauchy2d_chebyshev_d5_n10	qasm	gpu	True	1	False	64	1e-09	V
mvsp_cauchy2d_chebyshev_d5_n25	qasm	gpu	True	1	False	64	1e-09	V
mvsp_cauchy2d_chebyshev_d5_n40	qasm	gpu	True	1	False	128	1e-09	V
mvsp_cauchy2d_fourier_d10_n10	qasm	gpu	True	1	False	8	1e-09	V
mvsp_cauchy2d_fourier_d10_n25	qasm	gpu	True	1	False	8	1e-09	V
mvsp_cauchy2d_fourier_d10_n40	qasm	gpu	True	1	True	8	1e-09	D
mvsp_cauchy2d_fourier_d5_n10	qasm	gpu	True	1	False	8	1e-09	V
mvsp_cauchy2d_fourier_d5_n25	qasm	gpu	True	1	True	8	1e-09	D
mvsp_cauchy2d_fourier_d5_n40	qasm	gpu	True	1	True	8	1e-09	D
mvsp_gaussian2d_chebyshev_d10_n10	qasm	gpu	True	1	True	20	1e-09	V
mvsp_gaussian2d_chebyshev_d10_n25	qasm	gpu	True	1	True	25	1e-09	V
mvsp_gaussian2d_chebyshev_d10_n40	qasm	gpu	True	1	True	42	1e-09	D
mvsp_gaussian2d_chebyshev_d5_n10	qasm	cpu	True	0	False	12	1e-09	V
mvsp_gaussian2d_chebyshev_d5_n25	qasm	gpu	True	0	True	16	1e-09	V
mvsp_gaussian2d_chebyshev_d5_n40	qasm	gpu	True	1	False	40	1e-09	V
mvsp_gaussian2d_fourier_d10_n40	qasm	gpu	True	1	True	1	1e-09	D
mvsp_gaussian2d_fourier_d5_n10	qasm	gpu	True	1	True	1	1e-09	D
mvsp_gaussian2d_fourier_d5_n40	qasm	gpu	True	1	True	1	1e-09	D
mvsp_ricker2d_chebyshev_d10_n10	qasm	cpu	True	1	False	20	1e-09	V
mvsp_ricker2d_chebyshev_d10_n25	qasm	cpu	True	0	False	20	1e-09	V
mvsp_ricker2d_chebyshev_d10_n40	qasm	gpu	True	1	True	90	1e-09	D
mvsp_ricker2d_chebyshev_d5_n10	qasm	cpu	True	0	False	16	1e-09	V
mvsp_ricker2d_chebyshev_d5_n25	qasm	gpu	True	1	True	32	1e-09	D
mvsp_ricker2d_chebyshev_d5_n40	qasm	gpu	True	1	True	50	1e-09	V
mvsp_ricker2d_fourier_d10_n10	qasm	gpu	True	1	False	4	1e-09	V
mvsp_ricker2d_fourier_d10_n25	qasm	gpu	True	1	True	4	1e-09	V
mvsp_ricker2d_fourier_d10_n40	qasm	gpu	True	1	False	4	1e-09	V

Continued on next page

circuit	format	device	linearize	optimize level	tensor compiler	bond dimension	cut ratio	SVD
mvsp_ricker2d_fourier_d5_n10	qasm	gpu	True	1	False	4	1e-09	V
mvsp_ricker2d_fourier_d5_n25	qasm	gpu	True	1	True	4	1e-09	V
mvsp_ricker2d_fourier_d5_n40	qasm	gpu	True	1	True	4	1e-09	D