Practical Quantum Computing 2024 Project List

1. Compiling Quantum Circuits for Neutral Atoms Array Processors

Quantum programs can be described in terms of quantum circuits generated using quantum frameworks such as Cirq and Qiskit. However, these quantum circuits are not ready to be executed: they must be compiled into compatible language that the hardware can work with. The compilation of quantum circuits comes with many processes. Some of them are hardware-agnostic (for example, circuit optimization in the second week assignment of PQC), while others are hardware dependent.

One of such hardware dependencies is the routing on restricted topology. Two gate qubits can only be performed on adjacent qubits in hardware. To perform two qubits gates on distant qubits, swap gates are typically used to move qubits around to locations where such an operator is possible. The problem of finding a minimal amount of SWAPs is extremely complex.

In this project, we aim to solve the restricted topology problem in particular on the dynamic programmable quantum array (DPQA https://quantum-journal.org/papers/q-2024-03-14-1281/). DPQA is a potential candidate for quantum computing, where each qubit is by an atom trapped in an optical tweezer. One particularly useful property of DPQA is that qubit movement, in certain scenarios, has a lower error rate than a SWAP gate. Moreover, this architecture is particularly interesting for gate parallelism and the different zones of the computation (storage, computation, measurement).

Goals:

A. Familiarize yourself with the compiling process. Write a Qiskit/Cirq program that generates a SWAP schedule for a random circuit.
B. Write a program that draws the 2D atom arrays. Using this we will visualize the movement and scheduling of the operations. Learn the hardware restriction of DPQA. It is enough to assume that all qubits are moveable (of AOD type) as rows or columns.
C. Visualize the movement/SWAPS using the tool you wrote. You should see a sequence of bitmaps.
D. Create heuristic methods that use both qubit movement and swap gate to compile the circuit.

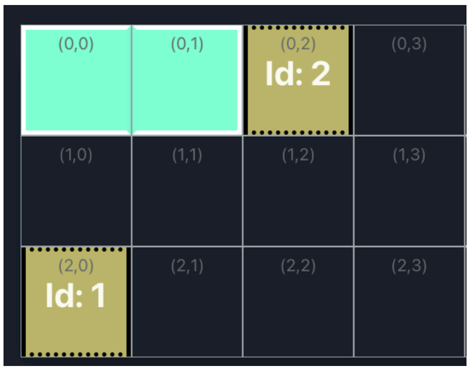
Criteria for Grading:

1. Pass (*): A, B, C
2. Full Points (**): Pass + D

2. 3D Routing for Lattice Surgery: Using Gravity in Minetest/Minecraft

Minetest is a clone of Minecraft and it shares a lot of similarities with how lattice surgery quantum circuits are looking. There are 3D space-time diagrams and the compiled circuit consists of many 3D voxels which are an expression of the quantum error-correction. Although inherently 3D, it is only recently that researchers started to use the third dimension for compilation/optimization. Before, everything has been treated like a sequence of 2D slices.

The project requires knowledge of Python and Lua (in Minetest). The goal is to map the compilation of lattice surgery circuits to the arrangement of 3D vertices into a Minecraft/Minetest environment. We have a proof of concept already working, and it can be used as a starting point. *More details upon starting the project.*

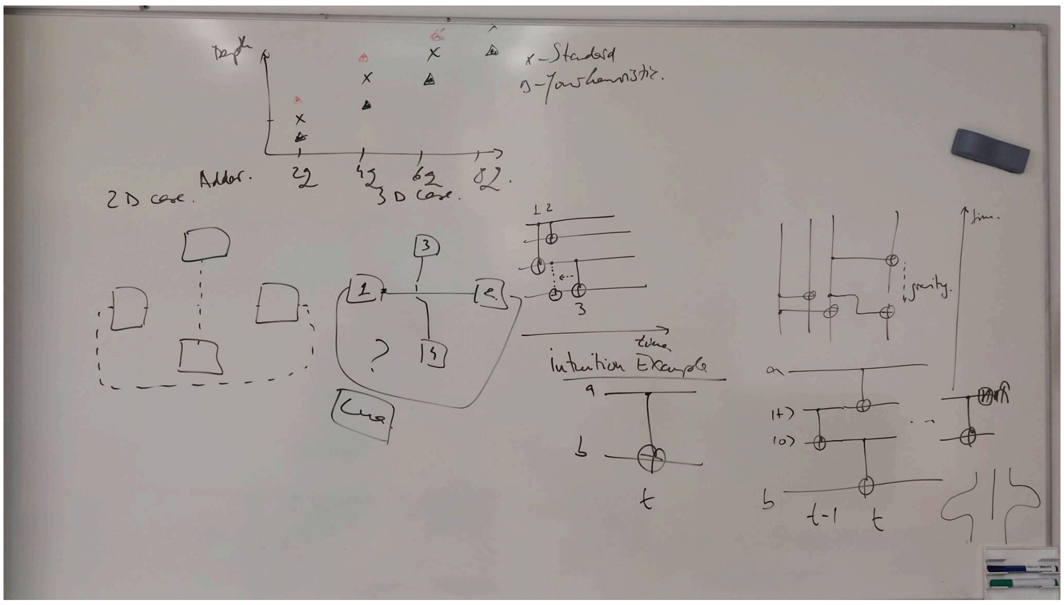


Goals:

A. starting from the existing Minecraft proof of concept https://github.com/latticesurgery-com/minetest/ , get the code running. We are also having our own state of the art lattice surgery compiler https://github.com/latticesurgery-com/
B. compile a random quantum circuit using the Minetest environment
C. implement a heuristic (similar to https://arxiv.org/abs/2401.15829) for analyzing the landscape before the voxels are falling
D. evaluate the resulting circuit depth -> lower is better

Criteria for Grading:

1. Pass (**): A, B
2. Full Points (***): Pass + C + D



3. Code Concatenation

Quantum Error Correction is theoretically enabled by the Threshold Theorem, and the theorem is formulated by the multiple concatenation of codes. Compiling the quantum circuits results after code concatenation is challenging because: a) there will be many qubits involved in the final circuit; b) each code used within the concatenation might have its own supported gate set (logical operators)

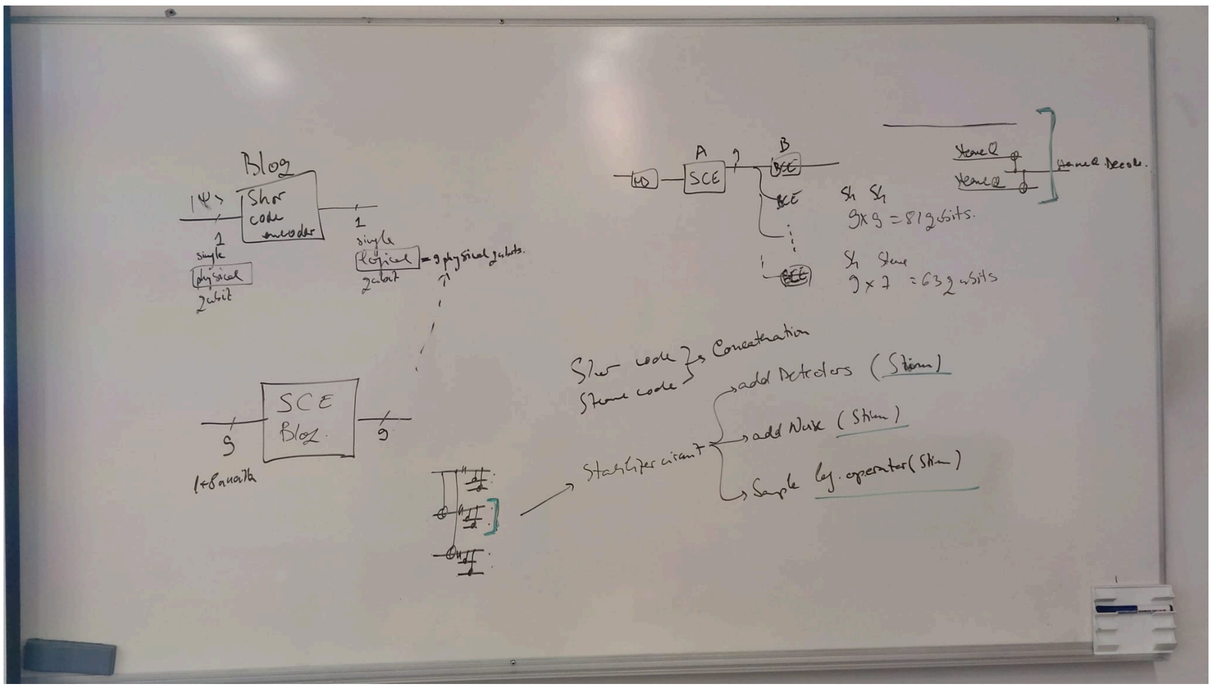
There are libraries, e.g. https://github.com/panqec/panqec/tree/main which already include codes (one can detect the circuit for measuring the stabilizers). However, you will not need those libraries, and instead write your own circuits for error-correction. *Use Google Qualtran to compile the concatenated codes.*

Goals:

A. Implement a simple code (e.g. Shor code) and concatenate it with itself
B. Implement a second code (e.g. Steane code, surface code) and concatenate it with the first one.
C. Draw the circuit diagrams for the resulting concatenations;
D. Plot logical error rates for the first code (without concatenation) - use Google Stim (start from the tutorial at https://github.com/quantumlib/Stim/blob/main/doc/getting_started.ipynb)
E. Plot logical error rates for the concatenated scheme of two levels.

Criteria for Grading:

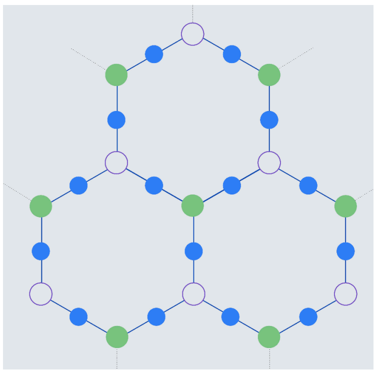
3. Pass (*): A, B, C
4. Full Points (**): Pass + D + E



4. Handcrafted Dynamic Circuits

Dynamic quantum circuits are quantum circuits which include mid-circuit measurements, and the application of some quantum gates is classically controlled by measurement results. Dynamic circuits are used in QECC, but these are also useful for the optimization of general computations. Moreover, dynamic circuits have been recently used for implementing complex computations on NISQ devices https://arxiv.org/abs/2308.13065

The main challenge with such circuits is that automatic compilation will often fail, because the dynamics is a function of the topology of the device. Therefore, usually these circuits are designed manually. Flag qubit schemes, distillation circuits, entanglement and even adders can be designed manually. The goal is to have low-depth, manually designed dynamic circuits for a given topology.



In this project we consider the IBM heavy hex topology. For the circuit derivations the Appendix of https://arxiv.org/abs/2308.13065 will be very useful.

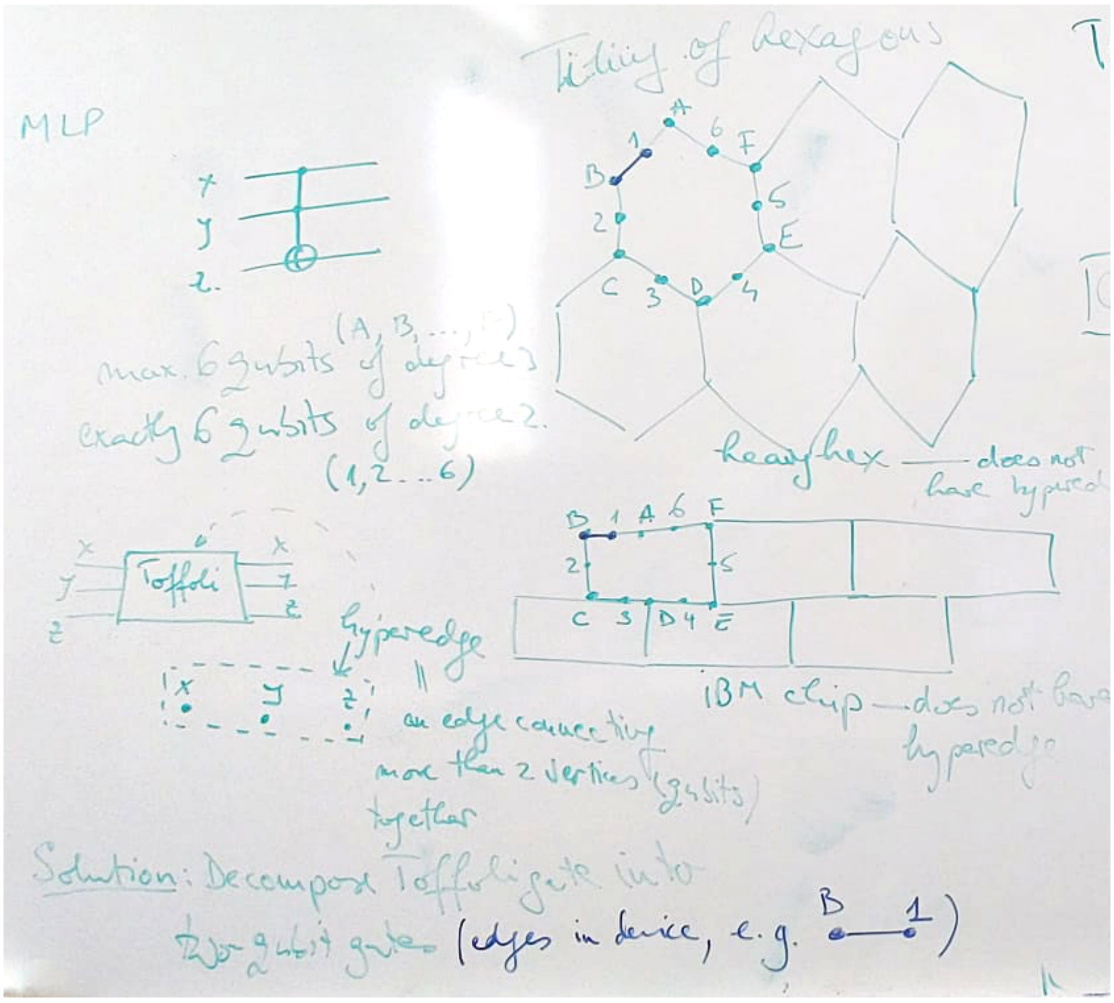
An example of decomposing Toffoli gates and mapping these to rectangular architectures can be found in https://arxiv.org/abs/2311.12510

Goals:

A. Map (relative-phase) Toffoli gates to heavy hex (https://arxiv.org/pdf/1709.06648, https://arxiv.org/abs/1508.03273)
B. Map the circuits from https://arxiv.org/pdf/1709.06648 to heavy hex
C. intuition (numerics using the noise model from https://arxiv.org/abs/2308.13065) regarding the minimum depth, error rate of the circuits

Criteria for Grading:

5. Pass (*): A, B
6. Full Points (**): Pass + C



MLP

Tiling of hexagons

max. 6 qubits of degree 3 (A, B, ..., F)

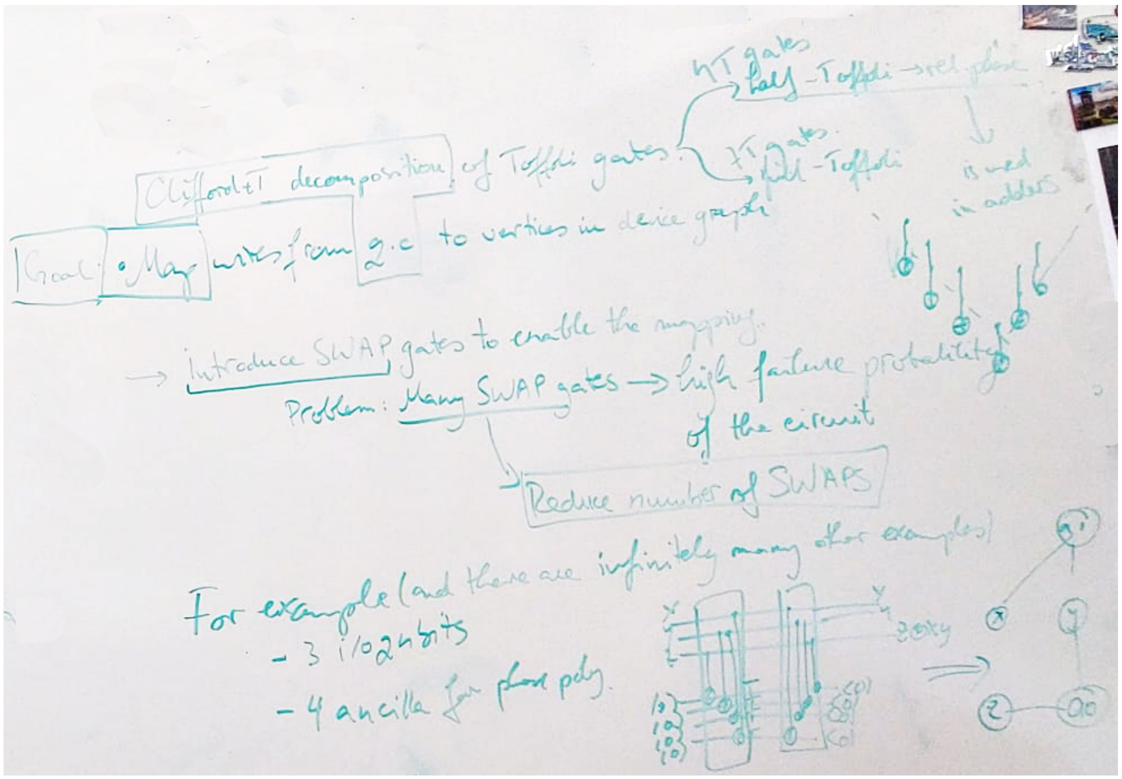
exactly 6 qubits of degree 2. (1, 2 ... 6)

heavy hex —— does not have hypered

IBM chip — does not have hyperedge

hyperedge ‖ an edge connecting more than 2 vertices (qubits) together

Solution: Decompose Toffoli gate into a two-qubit gates (edges in device, e.g. B —— 1)



Clifford+T decomposition of Toffoli gates: HT gates half-Toffoli → rel. phase ↓ is used in adders; 7T gates full-Toffoli

Goal: • Map wires from q.c to vertices in device graph

→ Introduce SWAP gates to enable the mapping.

Problem: Many SWAP gates → high failure probability of the circuit

→ Reduce number of SWAPS

For example (and there are infinitely many other examples)

- 3 i/o qubits
- 4 ancilla for phase poly.

5. Benchmarking BP vs other decoders

Recent advances in Belief Propagation based decoding techniques for quantum topological codes and LDPC codes has piqued the community's interest. BP is usually suboptimal for topological codes and requires a post processing stage such as ordered stage processing (OSD). Recently people have used BP as a predecoder to initialize more accurate second stage decoders such as Union Find (UF) and Minimum Weight Perfect Matching (MWPM) and achieved state of the art results. The goal of this project is to benchmark various combinations of BP decoders with the second stage decoders.

Goals:

A. Use ldpc-v2 library https://github.com/quantumgizmos/ldpc/tree/release_v2 to benchmark all available BP decoders for various noise models for surface code (phenological and circuit level noise).
B. Run benchmarks for BP with second stage decoders.
C. Find an optimal heuristic which outperforms existing ones.

Criteria for grading:

1. Pass (**): A and B
2. Full Points (***): A,B and C

References:

1. https://arxiv.org/abs/2307.14989
2. https://github.com/oscarhiggott/BeliefMatching

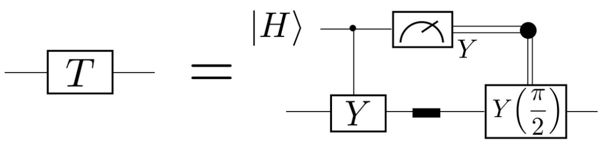
6. Fault-tolerant magic state preparation with flag qubits

Magic states are states such that when combined with Clifford gates can be used to make non-Clifford gates, which is needed for universal quantum computation. Magic stats are required for a procedure called gate teleportation, which is used to implement gates which are not native to a QECC. A variety of qubit magic state distillation routines (https://en.wikipedia.org/wiki/Magic_state_distillation) and distillation routines for qubits with various advantages have been proposed.

The goal of this project is to replicate the results from https://quantum-journal.org/papers/q-2019-05-20-143/ and to assess the cost of universal quantum computing with magic state.

Goals:

A. Verify that T gate can be simulated by a magic state |H> and the Clifford circuit:



B. Write in Google Stim / Google Cirq the circuits from Figs. 1- 7 from the referenced paper and simulate the circuits in order to replicate the presented results. Use the subcircuit from part A. Can you use Qualtran to make this modular?

C. Configure your circuit simulator to use the noise model from page 3.

D. Present plots similar to the ones from Fig. 9 and 10. Try initially for a single level distillation simulation. Based on the statistic obtained at level-0, predict the failure rate at higher levels.

Criteria for grading:

3. Pass (**): A, B, C
4. Full Points (***): Pass and D