Review and Implementation: A Shuttle-Efficient Qubit Mapper for Trapped-Ion QC

Lucas Kong Joshua Friend Roman Parra Bravo

The Pennsylvania State University

Trapped Ion QC System

- Traps are connected by a shuttle path, allowing movement from one trap to another if needed.
- Shuttle operations are required to bring both ions into the same trap.
- Ion2 is split from the chain and shuttled from T0 to T1, which adds energy to the ion.
- The initial mapping influences the total number of shuttles.

a

#	Gate	#	Gate
1.	MS q[0],q[1];	6.	MS q[0],q[1];
2.	MS q[4],q[5];	7.	MS q[2],q[3];
3.	MS q[1],q[2];	8.	MS q[0],q[1];
4.	MS q[3],q[5];	9.	MS q[1],q[2];
5.	MS q[4],q[2];	10.	MS q[0],q[1]

Figure 1. A sample quantum program consisting of ten 2-qubit gates.

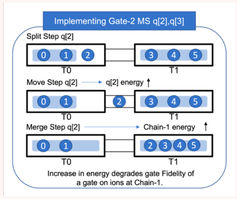

Figure 2. A two-trap TI system with three qubits in each trap, and shuttle steps to move ion-2 from trap T0 to trap T1.

Greedy Mapping Policy

- This Policy considers the number of gates between a pair of qubits.
- It maps the edges in descending order of weight, prioritizing edges with higher weights. This ensures that qubits with a high number of gates (weights) are placed close together within the same trap.
- Limitation: It assigns a constant edge weight for every occurrence of a gate between two qubits, regardless of the gate's position in the program (time locality).

Algorithm 1 Allocation of Edge Weight Using the Greedy Policy

Input: gate map **Output:** edge weights
  for gate ∈ gate map **do**
    if qubit edge weight ∈ edge register, **then** qubit edge weight += 1;
    else, qubit edge weight = 1;
    update edge register with qubit edge weight;
  end

Optimized Initial Mapping Huristic

Goal: Implement a mapping heuristic that is more sophisticated than the greedy policy. The mapping should consider additional parameters, such as the number of qubits, program depth, and the total number of gates.

Algorithm 2 Optimized Edge Weight Heuristic

  for gate ∈ gate map **do**
    if cnt ≤ number of gates **then**
      if qubit edge weight ∈ edge register **then**
        qubit edge weight += decaying function(f(cnt)); cnt += 1;
      else: qubit edge weight = total number of gates; cnt += 1;
    update edge register with qubit edge register;
  end

Decaying Step Function

- The quantum circuit is divided into n blocks, where n is determined empirically.
- All gates within a block are initially assigned the same weight.
- The first block is assigned a weight of n, the second block $n-1$, and so on.
- Each reoccurrence of a gate increases the weight of the corresponding edge.

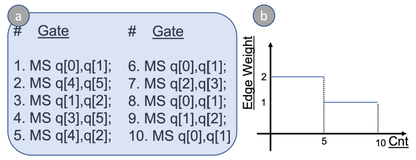

Figure 3. A decaying step function for the sample program.

Linear Decay Function

- Drawback of the step function: the "lump-sum" issue, where all gates within a block are treated equally.
- To address this issue, the linear function is continuous and assigns a linearly decreasing edge weight to later gate reoccurrences, prioritizing earlier gates.

The edge weight in the linear function can be modeled as:

$$W_{\text{linear}} = G - (a \times \text{cnt})$$

Where G is the number of gates, $a = 0.1$ (empirically determined), and cnt varies between 0 and (number of gates in the program - 1).

Exponential Decay Function

- Uses the same logic as the linearly decaying function.
- However, it has an exponential decrease.
- This approach more heavily favors earlier gate occurrences.

The edge weight in the exponential function can be modeled as:

$$W_{\text{exponential}} = G \times \frac{a - \text{cnt}}{G}$$

where $a = 2$ (empirically determined). Other definitions remain the same.

Penalized Linear Decay Function

Gate reoccurrences are assigned negative weight (penalized) if they happen after threshold because shuttles will occur more frequently at the end if mapping is determined by early gate occurrences.

The edge weight is modeled as:

$$W_P = G - \left(\frac{S \times Q \times D}{G}\right) \times \text{cnt}$$

where Q is the total number of qubits, D is the depth of the circuit, and S is the symmetry factor.

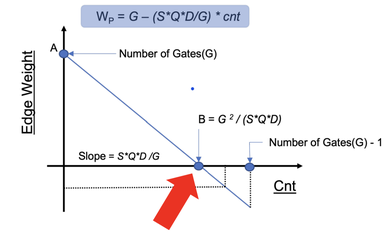

Figure 4. Graphical Representation of Penalized Decaying Policy

Conclusion

- Overall, there is a decrease in shuttle operations when using the presented algorithms, particularly the penalizing function.
- We conclude that this is a valid approach for initial ion-trap qubit mapping to reduce shuttle operations.

Python-based Implementation

- **Setting:** 79 qubits, 500 gates per program, average of 40 executions.
- All qubits are ensured to be used at least once.
- **Circuit Generation Function:** A function has been developed to generate two types of circuits: Symmetric Circuits (0) and Asymmetric Circuits (1).
- A symmetric program is defined as one that has a fixed, repetitive pattern of gates occurring throughout the program.
- **Implemented Functions:** Functions were implemented for each policy and simulated accordingly. Each function outputs a different qubit mapping to the trap, depending on its policy.
- The circuit generation is influenced by random functions, but general tendencies can be identified across different policies.

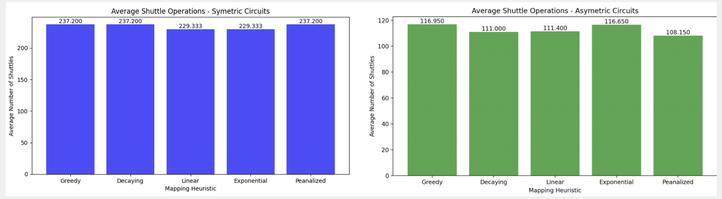

Figure 5. Average Shuttle Operations (Left: Symmetric Circuits, Right: Asymmetric Circuits)

Future Research

Classification of Circuit/Program Types: The analysis of patterns will help determine the most suitable policies for different types of circuits and programs.

References

[1] S. Upadhyay, A. A. Saki, R. O. Topaloglu, and S. Ghosh, "A Shuttle-Efficient Qubit Mapper for Trapped-Ion Quantum Computers," *Proceedings of the Great Lakes Symposium on VLSI 2022*, pp. 305–308, Jun. 2022, doi: https://doi.org/10.1145/3526241.3530366.