A Shuttle-Efficient Qubit Mapper for Trapped-Ion Quantum Computers

Suryansh Upadhyay
The Pennsylvania State University
PA, USA
sju5079@psu.edu

Rasit Onur Topaloglu
IBM Corporation
NY, USA
rasit@us.ibm.com

Abdullah Ash Saki
The Pennsylvania State University
PA, USA
axs1251@psu.edu

Swaroop Ghosh
The Pennsylvania State University
PA, USA
szg212@psu.edu

ABSTRACT
Trapped-ion (TI) quantum computer is one of the forerunner quantum technologies. Execution of a quantum gate in multiple trap TI system may frequently involve ions from two different traps, hence one of the ions needs to be shuttled (moved) between traps to be co-located, degrading fidelity, and increasing the program execution time. The choice of initial mapping influences the number of shuttles. The existing Greedy policy neglects the depth of the program at which a gate is present. Intuitively, the contribution of the late-stage gates to the initial mapping is less since the ions might have already shuttled to a different trap to satisfy other gate operations. In this paper, we target this gap and propose a new program adaptive policy especially for programs with considerable depth and high number of qubits (valid for practical-scale quantum programs). Our technique achieves an average reduction of 9% shuttles/program (with 21.3% at best) for 120 random circuits and enhances the program fidelity up to 3.3X (1.41X on average).

CCS CONCEPTS
• Hardware → Quantum computation.

KEYWORDS
Quantum computing, Trapped-ion, Compiler, Initial mapping, Shuttle operation, Fidelity.

1 INTRODUCTION
TI qubit offers several advantages such as perfectly identical qubits, long coherence times, and all-to-all connectivity among qubits [1], making them one of the most promising technology candidates for building NISQ devices. A major hurdle in realizing large TI systems is confining many ions in a single trap as it decreases the spacing between ions, making it challenging to pulse a qubit using laser controllers selectively. Therefore, the pathway to scalability in TI systems involves multiple interconnected traps. However, in a multi-trap system, computation is sometimes required on data from ions situated in different traps. For such cases, one ion needs to be shuttled (moved) from one trap to another to satisfy the intertrap communication, which increases program execution time and degrades quantum gate fidelity. To scale up TI technology for near-term applications, an extensive architectural study for multi-trap trapped ion systems has been reported in [4] where a compiler and a simulator has been developed with experimentally calibrated values. The initial mapping entails the assignment of logical qubits from the program to the traps and the relative position of qubits inside a trap. A Greedy mapping policy [4] has been proposed for the initial mapping (elaborated with an example in Section III). However, it does not consider the gate position in the program and program-specific parameters like the number of qubits, gates, and depth [5].

Various initial qubit mapping heuristics have been extensively reported for superconducting qubits [2], however literature lacks work on TI qubit mapping policy which is fundamentally different; this is a technical gap we target in this paper. For superconducting qubit systems, qubit-mapping algorithms explicitly attempt to minimize the amount of swap gates employed, also considering other parameters such as two-qubit gate error rates, single-qubit error rates, and execution time. However for TI qubits, as shuttle operations are expensive in terms of time and fidelity loss, the mapping heuristics aim to minimize the number of shuttles between the traps. In this paper, we present an efficient initial qubit mapping heuristic to target the inefficiency in the Greedy mapping policy. Our basic idea is to attenuate the weight of late occurring gates in initial mapping decision. We explore three different edge weight allotment heuristics, i.e., step, linear, and decay. Based on the intuition developed from the above explorations, we propose an optimized initial qubit mapper based on a penalizing approach which is also program adaptive. The rest of the paper is organized...
as follows: Section II describes the background for quantum computing and TI systems. Section III presents the proposed initial mapper heuristics. Section IV compares the Greedy mapper with the proposed heuristics. Finally, we conclude in Section V.

2 BACKGROUND

2.1 Qubits and Quantum gates

Qubits are the building block of a quantum computer that store data (i.e., $|0\rangle$ and $|1\rangle$) as various internal states. Due to quantum superposition property, a qubit can be in both $|0\rangle$ and $|1\rangle$ simultaneously unlike a classical bit which can either be 0 or 1. Computation in quantum systems is performed by manipulating the information stored in the qubits using quantum gates. They are realized using pulses such as radio frequency (RF) or laser. Fig. 1(a) illustrates a program with 2-qubit MS gates.

2.2 Trapped Ion Quantum Computer (QC)

2.2.1 Traps and Ion Chain. Trapped ion QC system are implemented by trap ionized atoms like Yb or Ca between electrodes using electromagnetic field [1]. Fig. 1b, illustrates various components of a 2-trap TI system. The ions are organized in the form of an ion chain inside a trap. Trap capacity is the maximum number of ions that a trap can accommodate. The traps are connected by a shuttle path which allows movement (shuttle) of an ion from one trap to another if needed. During the initial allocation of ions, a part of the total trap capacity is loaded with ions and the remaining capacity (termed as communication capacity) is kept unoccupied to allow for shuttled ions from other traps.

2.2.2 Shuttle Operation. A 2-qubit gate between ions from different traps requires a shuttle. For example, in Fig. 1a, the 5th gate MS q[4], q[2] involves ions from different traps. Therefore, a shuttle operation is needed to bring both ions into the same trap. A shuttle operation increases the vibrational energy ($\bar{\nu}$) affecting the gate fidelity as follows [4]:

$$F = 1 - \Gamma\tau - A(2\bar{\nu} + 1)$$ (1)

where, ($\tau$) is gate time, ($\Gamma$) is trap heating rate and ($\bar{\nu}$) is the vibrational energy and $A$ is a scaling factor. With increased $\bar{\nu}$, the subsequent gate operations on chain-1 will experience lower gate fidelities per Eq. 1. A lower gate fidelity will introduce more errors in the output and can completely decimate the result.

2.3 Initial Qubit Mapping

The initial mapping refers to mapping of logical qubits to physical traps and their relative positions inside a trap for the first time. The program execution will start with this allocation.

2.3.1 Greedy Mapping Policy and its limitations. In the Greedy policy, the qubits are mapped to the physical traps by considering the number of gates between the qubit pair. The edge weight represents the number of 2-qubit gates between a pair of qubits and a node represents a qubit. The Greedy policy maps the edges in the descending order of the weight, placing edges with high weight first, allowing qubits with a high number of gates close together, i.e., in the same trap. It starts by placing the highest weighted edge in one of the traps. Next, for each edge with one mapped and one unmapped endpoint, the algorithm maps the unmapped qubit to the adjacent position, minimizing the total distance between the qubit and its neighbors. The process is repeated for each unmapped edge in the descending order of the edge-weight. The Greedy mapping policy can be explained using the sample program in Fig. 1a. The edge weights of the program are, wt(0,1) = 4 (as the MS q[0],q[1] gate appears 4 times throughout the program), wt(1,2) = 2, and wt(4,5) = wt(2,3) = wt(3,5) = wt(4,2) = 1. Therefore, ions 0 and 1 are allocated first followed by ion 2, and finally, ions 3, 4, and 5. With greedy policy a gate appearing at the beginning of the program is assigned the same edge weight as the one appearing at the end. In programs with considerable depth and many qubits, the initial mapping using this policy is not optimum. Intuitively, the gates appearing toward the end of a deep program should not determine the initial mapping since the qubits might already have shuttled to a different trap.

3 OPTIMIZED INITIAL MAPPER HEURISTICS

3.1 Basic Idea

We propose a mapping heuristic where edge weights are assigned based on the number of gates between qubits and considering the stage of the program where the gate appears. We also consider program-specific parameters such as, the number of qubits, depth of the program, and the total number of gates while assigning edge weights. Priority is given to the gates re-occurring at the earlier stages of the program.

3.2 Edge Weight Function

We propose to improve the logic by assigning edge weights for any re-occurrence of a gate using a function rather than a constant value. However, the first occurrence of a gate in the program will be assigned the same constant value similar to the Greedy mapper policy. This process can be illustrated using the Algorithm 1. Here $cnt$ is just a variable used to implement a counter loop; it will take values from 0 to (number of gates in the program) - 1. For the first occurrence of a gate, edge weight will be assigned a constant value (equal to the total number of gates in the program) - 1. For the second occurrence of a gate, the edge weight will be incremented by 1. The process of assigning edge weights is repeated for every gate occurring after the first occurrence.
used for any re-occurrence of the gate later in the program. Our initial compilations are carried out using three different functions. By studying the effect of these functions on the number of shuttles, we arrive at our optimized initial mapping heuristic discussed later in this section.

**Algorithm 1: Optimized edge weight heuristic**

```plaintext
Input: gate map  
Output: edge weights  
for gate ∈ gate map do  
if cnt ≤ # of gates then  
    if qubit_edge_weight ∈ edge register then  
        qubit_edge_weight += decaying function(f(cnt));  
        cnt += 1;  
    else  
        qubit_edge_weight = total # of gates;  
        cnt += 1;  
    end  
else  
    qubit_edge_weight = total # of gates;  
    cnt += 1;  
end  
update edge register with qubit_edge_register;  
end
```

3.2.1 The Decaying Step Function. We divide our program into $n$ (n being an empirical parameter) equal blocks for simplicity. For each gate in a block, we assign a constant edge weight. The edge weight is assigned to the block in the form of a decaying staircase function, i.e., the first block is assigned a constant weight ‘$n$’, followed by ‘$n-1$’ for the next block, and so on.

**Example 3.1.** We consider a sample program with 10 MS gates (Fig. 1a) and divide it into two equal parts with five gates each. Any re-occurring gate in the first (second) block is assigned an edge weight of 2 (1). The first occurrence of the gate in the program irrespective of the block will be assigned a constant value equal to the number of gates in the program. The edge weights of the program are as follows: $w(t(1,1)) = 10 + 1 + 1 + 1 = 13$ (as the MS $q[0],q[1]$ gate appears 4 times throughout the program, the three re-occurrences occur at the second block hence this block gets assigned edge weight of 1), $w(t(1,2)) = 10 + 1 = 11$, and $w(t(4,5)) = w(t(2,3)) = w(t(3,5)) = w(t(4,2)) = 10$.

3.2.2 Linear Decay Function. One of the major drawbacks of using the step function becomes evident for programs with many gates where the blocks themselves contain a large number of gates with the same edge weight without considering the order in which the gates occur. We circumvent this problem by using continuous decaying functions. Using a continuous linearly decaying function, we assign any re-occurrence of a gate a linearly reducing edge weight value as we go deeper into the program to prioritize the gates that re-occur at the start of the program. We varied a (empirically parameter) between 0 to 1 to compare the number of shuttles with the Greedy mapper for various benchmark programs and random circuits (mentioned in Section IV). The edge weight in linear function can be modeled as:

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

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

3.2.3 Exponential Decay Function. The motivation behind using an exponential decay function is to get a sharp decrease in the edge weight as we go deep into the program. We tune empirical parameter $a$ to experimentally determine an optimal solution for the least number of shuttles across all benchmark programs. The edge weight can be modeled as

$$W_{exponential} = G \times a^{-cnt/G}$$

Here $a = 2$ (empirically determined). Other definitions remain same as before.

3.3 Penalized Linear Decay Function

All the three functions mentioned above assign an edge weight (though reduced) as we go deeper into the program. However, in penalized approach, we assign a negative weight to a gate re-occurrence after a certain point in the program i.e., we penalize any re-occurrence of a gate after a certain point in the program. Intuitively, the qubits with gates occurring at a later stage in a program will need to shuttle out more often if mapped predominantly by the weights determined by the early occurring gates. Hence, such qubits would require a different mapping negating the weights determined by the early occurring gates. The proposed function also considers the program parameters such as, the total number of gates, depth of the program, number of qubits, and symmetric/asymmetric nature of the program. We **define a symmetric program to have a fixed repetitive occurring pattern of gates throughout the program**. The edge weight is modeled as

$$W_p = G - (S \times Q \times D/G) \times cnt$$

Here $G$ is the number of gates, $Q$ is the total number of qubits, $D$ is the depth of the circuit, and $S$ is the symmetry factor. For symmetric circuits $S$ takes the value 0, else 1.

![Figure 2: Graphical representation of a Linear Decay function for implementing the penalizing policy.](image)

The edge weight allotment policy using this heuristic can be explained using Fig. 2. A program is divided into two parts by the point $B$. Any re-occurrence of a gate before that point will be assigned a positive edge weight though small in magnitude compared to a preceding gate as per the function. Any re-occurrence of a gate post point $B$ in the program will be penalized by assigning a
We report the number of circuits for which the number of shuttles is reduced when compared to the greedy mapping policy as shown in Table 3. We use the linear TI hardware model as the one used in [4]. We observe a decrease in the number of shuttles for 100 circuits using 120 randomly generated circuits which covers a wide range of communication patterns, number of qubits, number of gates and depths. Table 2 outlines the performance of different decaying functions and our policy (with Greedy policy being the baseline). The best gain of 1.41X on average (3.3X at best) is achieved for multi-trap TI quantum computers. Our technique achieves an average reduction of 9% shuttles/program (with 21.3% at best) for 120 random circuits and enhances the program fidelity up to 3.3X (1.41X on average) compared to the state-of-the-art Greedy mapper.

Table 3: Compilation Time Overhead

| Benchmark | Compile Time(sec) (Penalised) | Compile Time(sec) (Greedy) | Δ(↑) |
|-----------|-------------------------------|----------------------------|------|
| SquareRoot | 7.81                          | 8.14                       | -.33 |
| Supremacy  | 3.26                          | 3.55                       | -.29 |
| QAOA      | 19.18                         | 19.25                      | -.07 |
| QFT       | 20.82                         | 20.91                      | -.09 |

motion mode resulting in the improved gate and overall program fidelity. For the random circuits, the exponential and linear decay functions provide 10% and 6% fidelity improvement, respectively (Table 2). The best gain of 1.41X on average (3.3X at best) is achieved with the penalized approach due to reduction in the net shuttle operations.

5 CONCLUSION

In this paper, we present an efficient and more holistic initial mapper for multi-trap TI quantum computers. Our technique achieves an average reduction of 9% shuttles/program (with 21.3% at best) for 120 random circuits and enhances the program fidelity up to 3.3X (1.41X on average) compared to the state-of-the-art Greedy mapper.

6 ACKNOWLEDGMENTS

The work is supported in parts by NSF (CNS-1722557, CNS-2129675, CCF-1718474, OIA-2040667, DGE1723687, DGE-1821766, and DGE-2113839) and seed grants from Penn State ICDS and Huck Institute of the Life Sciences.

REFERENCES

[1] Wright et al. 2019. Benchmarking an 11-qubit quantum computer. (2019). https://doi.org/10.1038/s41467-019-13534-2
[2] Lingling Lao, Daniel Manzano, Hans van Someren, Imran Ashraf, and Carmen Garcia Almudever. 2019. Mapping of quantum circuits onto NISQ superconducting processors. arXiv: Quantum Physics (2019).
[3] Klaus Mølmer and Anders Sørensen. 1999. Multiparticle Entanglement of Hot Trapped Ions. Phys. Rev. Lett. 82 (Mar 1999), 1835–1838. Issue 9. https://doi.org/10.1103/PhysRevLett.82.1835
[4] Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. 2020. Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers. (2020), 529–542. https://doi.org/10.1109/ISCA45697.2020.00051
[5] Abdullah Ash Saki, Rasit Onur Topaloglu, and Swaroop Ghosh. 2021. Muzzle the Shuttle: Efficient Compilation for Multi-Trap Trapped-Ion Quantum Computers. (2021). https://doi.org/10.48550/ARXIV.2111.07961

---

Table 1: Reduction in the number of shuttles

| Benchmark | Qubits | Gates | Greedy | This Work | Δ(↓) |
|-----------|--------|-------|--------|-----------|------|
| SquareRoot| 78     | 1028  | 355    | 348       | 7    |
| Supremacy | 64     | 560   | 233    | 217       | 16   |
| QAOA      | 64     | 1260  | 975    | 975       | 0    |
| QFT       | 64     | 4032  | 196    | 196       | 0    |

Table 2: Analysis of 120 random circuits (Baseline: greedy)

| Parameters | Exponential (a = 2) | Linear (a = 0.1) | Penalized |
|------------|---------------------|------------------|-----------|
| # of ckts w/less shuttles | 73 | 77 | 100 |
| Avg. reduction in shuttles | 46.52 | 50.22 | 62.29 |
| # of ckts w/more shuttles | 47 | 43 | 20 |
| Avg. increase in shuttles | 38.86 | 43.60 | 29.55 |
| Net reduction in shuttles | 13.75 | 16.6 | 56.88 |
| Net % reduction in shuttles | ≈ 1.8% | ≈ 2% | ≈ 9% |
| Avg. net fidelity improvement | 1.1X | 1.06X | 1.41X |
| Max. fidelity improvement | 1.8X | 2.67X | 3.3X |

Table 3: Compilation Time Overhead

| Benchmark | Compile Time(sec) (Penalised) | Compile Time(sec) (Greedy) | Δ(↑) |
|-----------|-------------------------------|----------------------------|------|
| SquareRoot| 7.81                          | 8.14                       | -.33 |
| Supremacy  | 3.26                          | 3.55                       | -.29 |
| QAOA      | 19.18                         | 19.25                      | -.07 |
| QFT       | 20.82                         | 20.91                      | -.09 |

4 EVALUATION AND RESULTS

4.1 Experimental Setup

We use the linear TI hardware model as the one used in [4]. We consider the “L6” trap topology [4] where six traps are connected linearly. For each trap, initially, 15 ions can be loaded. Considering the baseline capability for 50-100 qubit NISQ systems, we selected applications with 60-80 qubits and 500-4000 two-qubit gates [4]. Our benchmark suite includes circuits from Google’s supremacy experiment, quantum approximate optimization algorithm (QAOA), quantum Fourier transform (QFT), a quantum arithmetic circuit Square Root [4]. To expand the benchmark suite, we also test for 120 randomly generated circuits which covers a wide range of communication patterns, number of qubits, number of gates and depths.

4.2 Results

4.2.1 Shuttle Reduction. Table 1 shows the reduction in the number of shuttle operations using the new heuristics compared to the Greedy mapper for the benchmarks circuits especially for square root and supremacy. For symmetric programs such as, QAOA and QFT circuits, our algorithms still show similar performance as that of Greedy as expected. The basic idea of prioritizing gates based on their re-occurrence becomes irrelevant for a symmetric program as each gate will have the same number of re-occurrences and at the same relative time in the program. Our program has a Symmetry factor to compensate for that. We also test our policy using 120 randomly generated circuits covering a wide variety of programs with Qubits varying from ≈ (60 − 75), number of gates between ≈ (900-2000), high depth, and diverse communication patterns. Table 2 outlines the performance of different decaying functions and our policy (with Greedy policy being the baseline). We report the number of circuits for which the number of shuttles are reduced and increased, average reduction and increment in the number of shuttle operations and net reduction in the number of shuttles for the whole 120 random circuits. Out of 120 circuits, we observe a decrease in the number of shuttles for 100 circuits with an average percentage decrease of ≈9% for the penalizing weighting function. Furthermore, our mapping optimization for shuttle reduction does not show any increase in compilation time when compared to the greedy mapping policy as shown in Table 3.

4.2.2 Program fidelity improvement. Shuttle operation increases vibrational energy , of an ion-chain and degrades gate fidelity as per Eq. 1. As our proposed policies reduces shuttles, it curbs negative weight. The point B for a program depends on an empirically determined function of number of gates, depth, symmetry, and number of qubits.