TRIM: crossTalk-awaRe qubIt Mapping for multiprogrammed quantum systems

Soheil Khadirsharbiyani*, Movahhed Sadeghi*, Mostafa Eghbali Zarch†, Jagadish Kotra‡, and Mahmut Taylan Kandemir*
*Computer Science and Engineering Department, Pennsylvania State University, University Park, USA
†Electrical and Computer Engineering Department, NC State University, Raleigh, USA
‡AMD, Austin, USA
*szk921@psu.edu, *mus883@psu.edu, †meghbal@ncsu.edu, ‡Jagadish.Kotra@amd.com, *mtk2@psu.edu

2023 IEEE International Conference on Quantum Software (QSW) | 979-8-3503-0479-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/QSW59989.2023.00025

***Abstract*—The challenge of mapping logical qubits to physical qubits in quantum systems has been addressed in prior proposals that optimize the Probability of Successful Trial (PST) by considering the coherence and gate error rates. However, these proposals do not account for crosstalk errors, which occur when active qubits interact during execution. The reason for this is that crosstalk only appears after the initial mapping, while previous strategies allocate qubits based on program and quantum system characteristics using one-step mapping methods. Scheduling-based solutions have been created to address this problem by inserting barriers between gates to reduce crosstalk, but at the expense of increased execution time and coherence error rates, ultimately decreasing overall accuracy. This paper presents and evaluates TRIM, a novel strategy that characterizes crosstalk and eliminates it in an iterative fashion using a multi-step greedy search method, which can be applied to any qubit mapping to reduce crosstalk while keeping execution time and coherence errors in check. Evaluations of TRIM using multiple workloads show PST improvements of 7.3% for single-programmed execution and 7.7% for multiprogramming scenarios, while reducing or keeping the number of gates, compared to a state-of-the-art mapping scheme. Additionally, TRIM achieves 5.4% and 3.3% PST improvements for single-programmed and multiprogrammed executions, respectively, compared to a state-of-the-art scheduling strategy.**

***Index Terms*—Quantum Computing, Reliability, Mapping Techniques, NISQ, Compiler**

I. Introduction

Quantum computing (QC) is a paradigm that aims to reduce the execution time of various applications from different domains, including finance [1], chemistry [2], and data mining [3], through the use of qubits and quantum gates. However, the "reliability" of quantum computing is a major concern due to various error types, such as crosstalk, coherence, and gate errors, which arise from manufacturing variations in quantum hardware [4]. The effect of these errors can be exacerbated when the size of the circuit is increased or when multiple programs run on the same quantum hardware, leading to intense interactions [4].

While various error correction techniques [5], [6], [7], [8], [9] have been introduced over the last two decades, due to enormous size requirements they impose, they *cannot* be effectively employed in current systems. Therefore, the concept of *NISQ* (Noisy Intermediate-Scale Quantum) [10] architecture has been introduced, with the goal of executing programs without any error correction technique while trying to minimize the reliability issues using alternative approaches. One of the approaches to improve reliability is *mapping*, which aims to minimize errors by using more reliable physical qubits when executing a quantum program while minimizing the number of gates by reducing the SWAP operations that need to be performed. Basically, a mapping scheme maps virtual/logical qubits (i.e., qubits of the quantum program) to physical qubits (i.e., qubits of the hardware), aiming to ensure the correct execution of the program. [4], [11], [12], [13], [14], [15], [16], [17] are a subset of representative research papers in this area.

Table I reveals that previous mapping and routing techniques have primarily focused on minimizing error rates by reducing coherence and gate errors through optimizing SWAP operations using routing algorithms. However, to our knowledge, *none of these techniques directly address crosstalk errors*, which occur when an operation on one qubit unintentionally impacts other qubits. This oversight is significant because crosstalk errors can increase the error rate on neighboring qubits by up to 11x [4], thereby significantly impacting system reliability. A state-of-the-art scheduler proposal [4] attempts to minimize crosstalk effects by scheduling gates in separate steps, using barriers between them. Unfortunately, this method increases execution time and, while eliminating crosstalk, introduces coherence errors, ultimately affecting overall reliability.

In this paper, we first investigate various strategies that can be incorporated into "one-step" mapping algorithms. Our results reveal that current one-step mapping approaches *cannot* remove crosstalk errors in single-programming or multiprogramming execution scenarios. To tackle this issue, next, we present **TRIM** (crossTalk awaRe qubIt Mapping), an iterative, search-based greedy strategy, which starts with an initial mapping and then searches through the design space for a superior group of qubits with fewer crosstalk cases. TRIM keeps running the search algorithm until it runs out of better candidates, reaching a local or global minimum (in terms of error rate). To further improve the effectiveness of TRIM, we also introduce a relaxed version of a recent mapping scheme (QuCloud [17]). We observe that TRIM when coupled with this relaxed mapping

The material presented in this paper is based upon work supported by the National Science Foundation under Grant Numbers 2119236, 2122155, 2028929, 1931531, and 1763681.

979-8-3503-0479-4/23/$31.00 ©2023 IEEE
DOI 10.1109/QSW59989.2023.00025

	Mapping	Scheduler	Routing Optimizer	Coherence Mitigation	Crosstalk Mitigation	Gate Error Reduction	Multiprogramming or Single Execution
SABRE [11]	✓	✗	✓	✗	✗	✓	SP
VQA [12]	✓	✗	✗	✗	✗	✓	SP
VQM [12]	✗	✗	✓	✗	✗	✓	SP
HA [14]	✓	✗	✓	✗	✗	✓	SP
Nash et al. [15]	✓	✗	✓	✗	✗	✓	SP
DIS [18]	✗	✓	✗	✓	✗	✗	SP
FRP [18]	✓	✗	✓	✗	✗	✓	MP
QuCloud [17]	✓	✗	✓	✗	✗	✓	MP
Murali et al. [4]	✗	✓	✗	✗	✓	✗	SP
TRIM	✓	✓	✗	✗	✓	✓	Both MP and SP

TABLE I: Key characteristics of the existing mapping/scheduling schemes and TRIM.

scheme improves the *Probability of Successful Trial* (PST) over the baseline mapping scheme. Our main **contributions** in this work can be summarized as follows:

- We characterize the crosstalk error and its extent in quantum hardware using different programs. In lieu of characterizing the entire system, our method only characterizes a subset of links that satisfy the criteria of our algorithm, minimizing the characterization overhead by 5.6x.
- We discuss that existing mapping proposals are agnostic to crosstalk effects and demonstrate that the crosstalk can severely hurt the PST of the system. Also, focusing on the scheduling strategy proposed in [4], we show the extent of the error rate introduced because of the increase in execution time.
- To address this issue, we propose a novel approach, TRIM, which is a crosstalk-aware iterative logical-to-physical qubit mapping scheme that minimizes the crosstalk errors while – at the same time – maintaining or reducing the gate errors and coherence errors. We want to emphasize that our iterative algorithm can start with *any* initial logical-to-physical qubit mapping.
- We also present a relaxed mapping algorithm for the initial mapping, aiming to increase the chances for achieving the global minimum for the crosstalk error.[1]
- We present an experimental evaluation of TRIM using a set of 10 single-programmed and 16 multiprogrammed workloads using a real quantum system. For single-programmed cases, our approach achieves 7.3% and 5.4% average PST improvements over [17] (a state-of-the-art mapping scheme) and [4] (a state-of-the-art scheduling strategy), respectively. For multi-programmed cases, the collected experimental data indicate that our proposed scheme improves the PST by, on average, 7.7% and 3.3%, compared to [17] and [4], respectively.

II. Background and Methodology

A. Quantum Computing and Quantum Errors

In quantum computing (QC), information is stored in "qubits" rather than bits. A qubit is a linear combination of two basic states, $|0\rangle$ and $|1\rangle$, and can be represented as $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$, with $|\alpha|^2 + |\beta|^2 = 1$. Quantum gates modify qubit states by influencing complex numbers α and β. Manufacturing challenges, qubit/gate imperfections, and external interferences can cause different types of reliability issues in quantum circuits. These errors can be broadly classified into three main categories:

- *Coherence Errors:* Qubits maintain their state for only a limited amount time, with errors increasing exponentially, potentially causing a qubit to change to state $|0\rangle$.
- *Gate Errors:* Errors may occur during quantum operations on qubits. Single-qubit gates have lower error rates (about 10^{-3}) compared to two-qubit gates like *CNOT* (10^{-2}).
- *Crosstalk and Measurement Errors:* Quantum operations can affect both involved and adjacent qubits. Crosstalk errors arise when two *CNOT*s are applied to neighboring qubits, increasing the error rate significantly. Minimizing simultaneous *CNOT*s on nearby qubits is essential. Additionally, measuring a qubit's state can impact surrounding qubits. To minimize this, measurements are typically performed towards the end of the program.

B. Execution Flow in QC

To execute a program correctly on real quantum hardware, six steps must be completed in sequence. First, the program should be converted into 1-qubit and 2-qubit gates to be executable on real hardware. Next, the "logical qubits" (qubits in the application program) should be *mapped* to "physical qubits" (the physical implementation of qubits). This mapping enhances system fidelity by reducing the number of SWAP operations and crosstalk, ultimately lowering gate errors. The compiler then inserts SWAP operations between gates, ensuring that CNOTs are executed only on directly linked physical qubits. Following this, various *optimizations* such as [19], [20], [21], [22] are applied to the quantum program's instructions to minimize gate count. Lastly, before executing on real hardware, gates are scheduled to minimize execution time and/or maximize reliability.

When mapping is not effective, current quantum systems use *scheduling mechanisms* to further mitigate reliability issues. One example of such scheduling is shortening the time between the preceding operation and measuring the result qubits, which helps reduce coherence errors. The resulting instructions are then sent to the quantum machine, and the final result is formed upon measuring the output qubits. Due to the *probabilistic nature* of quantum computing, the outcomes of different executions of a quantum program may vary. As a result, multiple *runs* (also known as *shots*) are typically performed to determine the probability of obtaining the correct output. In

[1]The global minimum mapping is a qubit allocation (mapping) that minimizes crosstalk to the greatest extent possible.

this context, the likelihood of achieving the desired outcome is denoted by the term *Probability of Successful Trial* (PST).

Workload	Abb.	Qubits#	Gates#	CNOT#
Small Benchmarks				
pea_5	P1	5	98	42
mod5mils_65	P2	5	35	16
Medium Benchmarks				
mini_alu	P3	10	168	77
qpe_9	P4	9	123	43
rd53_138	P5	8	132	60
simons_6	P6	6	44	14
bv_14	P7	14	41	13
Large Benchmarks				
multiply_13	P8	13	98	40
sat_11	P9	11	679	252
rd73_140	P10	10	230	104

TABLE II: Benchmark characteristics. (P: Program)

C. Experimental Setup

In this section, we provide details about our evaluation methodology and the benchmark programs used in our study. The quantum programs utilized in our evaluations are sourced from RevLib [23] and QASMBench [24]. Table II summarizes the benchmarks used, their abbreviations (used in result tables), the number of qubits, and gates. The TRIM evaluation is performed on ibmq_montreal [25], which is a Quantum Falcon processor from Qiskit [26]. The high-level view of this architecture, including qubits and their connections, is illustrated in Figure 1-a. We execute all our experiments using 4000 shots, and we report PST and gate number in all results. In case of multiple correct results, we calculate the PST using the following formula: $PST = \sum_i^{Res} \frac{PST[i]}{Count[Res]} = \sum_i^{Res} \frac{1-\frac{|x_i-n_i|}{n_i}}{Count[Res]}$

$if\ (x_i = 0 \vee |x_i - n_i| \geq n_i) \rightarrow PST_i = 0,$

where x_i is the observed output count and n_i is the expected output count. If we apply a Hadamard gate[2] to a qubit and take measurements for 4000 shots, for instance, we have an idea of the qubit's behavior. The result of this scenario should contain 2000 zeroes and 2000 ones. Therefore, if we count 1500 zeroes and 2500 ones, we may derive the PST as follows: $PST = \frac{(1-\frac{|1500-2000|}{2000})+(1-\frac{|2500-2000|}{2000})}{2} = 0.75 = 75\%.$

To evaluate the effectiveness and reliability of our proposed TRIM method, we also present results from a multiprogrammed (MP) execution scenario, where multiple quantum applications run concurrently on the same quantum hardware. The MP execution typically leads to higher hardware utilization and throughput but also intensifies reliability challenges that must be mitigated to minimize their impact. To achieve optimal performance and reliability in MP execution scenarios, we utilize state-of-the-art mapping and scheduling algorithms as baselines; specifically, QuCloud [17] for mapping and the method proposed in [4] for scheduling. QuCloud aims to enhance reliability in MP environments by reducing the number of SWAP operations, while the method in [4] minimizes crosstalk by altering the execution order of simultaneous CNOTs on neighboring qubits. It is important to emphasize that TRIM can be combined with *any* mapping and scheduling algorithm. In single-programmed cases, we use the same mapping as in multiprogrammed cases, with the only difference being the absence of the second program.

[2]When applied to a constant qubit, the Hadamard gate forms a uniformly random qubit, which, when measured, behaves like a coin toss.

III. Motivation

A. Mutliprogramming in QC

Over last decade, cloud computing has become a norm in modern compute platforms, where multiple concurrently-running application programs share the available hardware and system resources (also called consolidation). QC is no exception, and a number of recent proposals (see [17], [18], [27] and the references therein) have attempted to incorporate multiprogramming into various quantum systems and consolidate resources. The increase in system utilization and throughput is one of the most important advantages of multiprogramming, as compared to single-program execution. The existing research on multiprogramming of quantum hardware [17], [18] indicates that, despite the fact that modern quantum systems have a restricted number of qubits (on the range of 10-100s), they are still underutilized, making multiprogramming a promising option for such systems.

While multiprogramming improves the system's throughput, it also introduces a slew of reliability issues, affecting the system's final output. One issue is that the quantum programs that are executed simultaneously can be (and, in most cases, will be) "incompatible", in the sense that they can finish at different times, thereby causing significant coherence errors [18]. Another issue is deciding how to route two qubits via SWAP insertions when the best path between them could pass through other (concurrently-running) programs. Since a given qubit of a program can interact with the qubits of other programs, allowing inter-program SWAPs can improve gate error at the cost of crosstalk error. If inter-program SWAPs are not allowed, routing between two qubits may be impossible or very expensive to achieve in some cases, depending on the number of CNOTs. The third and possibly most important issue is about quantum program mapping. Since different qubits have different characteristics, e.g., the number of links, error rate of each link and error rate of the qubit itself, an inefficient mapping may have a low chance of producing correct output.

B. Related Works and Their Limitations

Mapping and scheduling techniques attempt to improve the reliability (PST) of quantum programs from different angles. More specifically, the existing approaches to logical-to-physical qubit mapping mostly focus on minimizing the number of SWAPs by introducing new routing strategies and ensuring that programs can fit into available quantum hardware (shown in Table I). In contrast, existing scheduling techniques mostly attempt to minimize the different types of errors by changing the execution sequence (order of computations) while maintaining the correctness of the program.

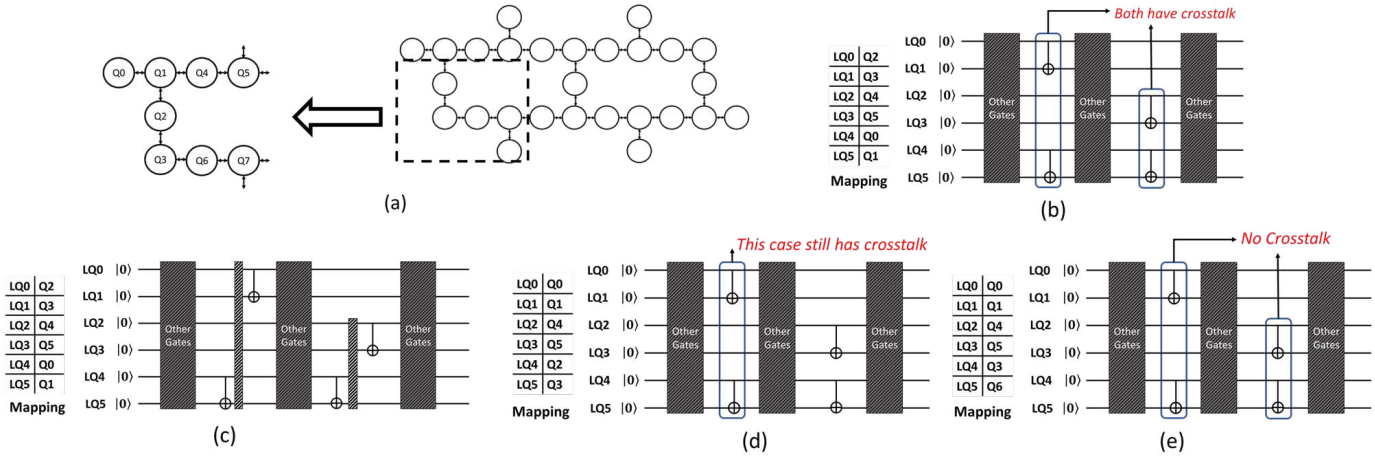


Fig. 1: (a) Connectivity of the 27-qubit ibmq_montreal, (b) Basic mapping, (c) Basic mapping+crosstalk scheduler [4], (d) A new mapping that eliminates only one of the cases, and (e) Optimal target.

		QuCloud [17]	QuCloud [17]+Murali et al. [4]
Case#	**Benchmark**	**PST[%]**	**PST[%]**
S1	**P1**	63.2%	59.4%
S2	**P2**	58.2%	52.8%
S3	**P3**	47.6%	42.2%
S4	**P4**	23.4%	18.1%
S5	**P5**	78.3%	69.2%
S6	**P6**	49.1%	44.2%
S7	**P7**	36.1%	29.5%
S8	**P8**	21.5%	18.3%
S9	**P9**	38.2%	30.5%
S10	**P10**	38.7%	33.2%
Average		45.4%	39.7%

			QuCloud [17]			QuCloud [17]+Murali et al. [4]		
Case#	**Benchmarks**		**PST[%]**			**PST[%]**		
	WL1	WL2	WL1	WL2	Avg	WL1	WL2	Avg
M1	P10	P1	37.2%	56.5%	46.8%	33.3%	51.7%	42.5%
M2	P9	P1	37.3%	55.8%	46.5%	32.1%	50.4%	41.2%
M3	P8	P1	20.1%	56.5%	38.3%	14.1%	52.8%	33.5%
M4	P7	P1	35.7%	55.9%	45.8%	29.0%	50.4%	39.7%
M5	P6	P1	48.8%	56.8%	52.8%	42.9%	52.4%	47.6%
M6	P5	P1	76.8%	55.9%	66.3%	72.7%	47.3%	60.0%
M7	P4	P1	23.4%	57.7%	40.5%	16.2%	54.3%	35.2%
M8	P3	P1	46.6%	56.1%	51.4%	39.7%	48.1%	43.9%
M9	P10	P2	37.3%	51.9%	44.6%	32.5%	46.7%	39.6%
M10	P9	P2	36.9%	52.7%	44.8%	28.6%	43.8%	36.2%
M11	P8	P2	19.7%	50.5%	35.1%	15.8%	47.3%	31.6%
M12	P7	P2	34.7%	53.1%	43.9%	29.3%	49.7%	39.5%
M13	P6	P2	48.9%	51.3%	50.1%	40.0%	45.3%	42.6%
M14	P5	P2	77.2%	53.1%	65.2%	70.6%	49.2%	59.9%
M15	P4	P2	23.2%	51.8%	37.5%	15.5%	45.0%	30.3%
M16	P3	P2	45.9%	51.4%	48.6%	39.7%	45.9%	42.8%
Average			40.6%	54.2%	47.4%	34.5%	48.8%	41.6%

TABLE III: PST comparison between QuCloud and QuCloud+ [4] without considering crosstalk error. "P#" refers to the applications listed in Table II.

The state-of-the-art mapping mechanisms focus on minimizing the gate error rates. For example, SABRE [11], as shown in Table I, is a SWAP-based heuristic routing and mapping method that employs a search strategy to optimize the number of gates. Although it reduces the gate errors, it does *not* consider crosstalk which can be of significant concern. Additionally, it only considers a single application, obviating most crosstalk concerns. VQA and VQM [12] are complementary methods that employ routing and mapping techniques to minimize the SWAPs and to employ qubits with the lowest total link error rate. FRP [18] is an allocation/mapping technique specifically proposed to reduce the number of SWAPs, minimizing the gate error rate when multiple applications are running at the same time. Nash et al. [15] present a circuit synthesis technique that takes an input circuit and adjusts it to match the connectivity profile of the quantum hardware (on which the program is going to execute) in a single program environment. QuCloud [17], on the other hand, has introduced a graph-based mechanism that minimizes the number of SWAPs in a multiprogramming-based execution environment by allowing inter-program SWAPs. QuCloud does *not* consider crosstalk as part of the mapping; instead, it relies on the subsequent scheduling step to handle it. While all these prior approaches improve system reliability to varying degrees by lowering gate errors, *none of them, to our knowledge, has explicitly addressed crosstalk between applications caused by CNOTs operating on adjacent qubits.* This is primarily because crosstalk happens when two gates are scheduled to run at the same time, and *no* scheduling information is available during the "initial mapping" process since scheduling is the last step before executing the program on the target quantum hardware (recall the steps in running a program mentioned in Section II-B) Unfortunately, this crosstalk issue may increase the error rate by up-to 11x [4], and this in turn, can have a significant negative impact on the system's PST.

Several scheduling strategies have been proposed over the last decade to minimize different types of errors when mapping is not efficient. For example, DIS [18] is a scheduler that executes different applications at different steps; so, all of the applications finish at the same time, thereby minimizing the coherence error. The scheduling strategy, presented in [4], aims to minimize crosstalk errors by providing a scheduler that alters the sequence of parallel neighboring CNOTs by placing a *barrier* between them and thus eliminating crosstalk in the process. Figure 1-b illustrates an example where four CNOTs are running on neighboring qubits after scheduling, causing two potential crosstalk errors. As demonstrated in Figure 1-b, the

proposal in [4] postpones the execution of two of the CNOTs so that qubits do not have any interactions. *While this solves the reliability problem, it also increases the execution time, thus potentially magnifying the system's coherence error.* To carefully evaluate the impact of the approach presented in [4], we have conducted a theoretical study comparing the PST of the original case vs. when the scheduler in [4] is employed. In both the cases, we have assumed that crosstalk errors are non-existent to exclusively evaluate the PST difference due to changes in the order of instructions. Although [4] employs different techniques, such as increasing the parallelization to minimize the coherence error impact on the PST, a comparison of these two scenarios reveals a decrease in PST of 1% to 7% (on average, 2.8%) due to an increase in coherence errors (reported in Table III). Therefore, while the approach proposed in [4] minimizes the crosstalk errors, it causes another type of error – coherence error – which can easily offset any potential benefits from the crosstalk error minimization.

On the other hand, mapping-based approaches can solve this problem by limiting the occurrence of scenarios where two CNOTs share adjacent qubits. It is to be noted however that, since various crosstalks can occur at different stages of execution and it is possible to create other crosstalks when trying to eliminate one of the crosstalks, there is presently no simple solution to eliminate the crosstalks during the mapping process. Figure 1-c illustrates how modifying the mapping can solve the crosstalk for CNOTs between LQ2-LQ3 and LQ4-LQ5 but it cannot eliminate the crosstalk for CNOTs between LQ0-LQ1 and LQ4-LQ5. In this work, our main goal is to investigate various strategies to discover the best feasible mapping that eliminates all potential crosstalks while minimizing the increase in the execution time. For our current example, Figure 1-d shows the mapping result that we aim to achieve. Summarizing the observations from Table III and Figure 1, it is evident that current mapping and scheduling techniques are not able to reduce the crosstalk errors without causing other types of reliability concerns. To that end, we propose TRIM, an iterative greedy search algorithm that *modifies* a given "initial mapping", in an *iterative fashion*, to minimize the crosstalk scenarios while keeping the coherence error of the system in check.

In summary, our work stands out from existing mapping strategies by integrating crosstalk considerations into the initial mapping process. Additionally, it differs from current scheduling techniques as we do not increase coherence errors while eliminating crosstalk, unlike existing methods.

IV. Design Choices

In this section, we discuss various design choices to address crosstalk issues in a quantum system. We examine each choice in detail and explain its potential impact on the system's output.

Multi-Step vs Single-Step Mapping: The optimization for reducing crosstalk can be incorporated during or after the mapping stage. However, since the order of CNOT execution is only available *after* scheduling; direct information about crosstalk is inaccessible during the initial mapping stage. As a result, the only way to consider crosstalk ("indirectly") is by mapping programs to groups of qubits with limited links between groups. Unfortunately, this mapping cannot mitigate the PST degradation resulting from crosstalk between qubits within each program.

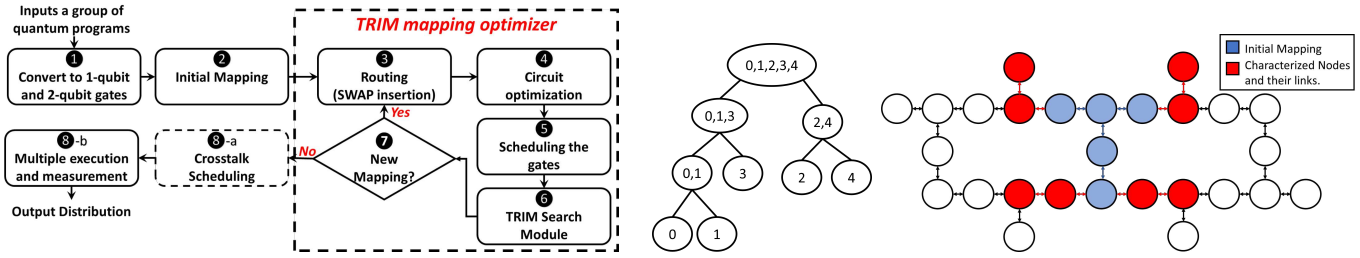
While previous approaches [17], [18] have utilized indirect minimization techniques for crosstalk, to our knowledge, *none* has managed to identify accurate crosstalk information, leading to false crosstalk detection instead. An example of this is when two neighboring qubits are prevented from being assigned to different programs in [17]. If these two qubits do not have any simultaneously running CNOTs, no crosstalk would occur, limiting the possibility of running other programs without providing any benefits. Consequently, a *two- or multi-step heuristic* should be designed to reduce crosstalk cases, progressively optimizing the target program's PST. Our paper proposes an iterative search algorithm using information from the "current mapping" to improve "subsequent iterations mapping" with the aim of reducing crosstalk.

Selection of the Initial Mapping: An essential criterion in developing a search-like heuristic algorithm is to avoid getting trapped in a local minimum. Our mapping should be able to search for different candidates to find the best logical-to-physical mapping with the highest output reliability. The initial mapping may not directly affect this issue but can create scenarios leading to limited options (or no options) for our search algorithm, reducing the chance of finding a better mapping.

Various mapping algorithms have been proposed over the past two decades, including SABRE [11] and QuClou-d [17]. While each has its advantages and disadvantages, none is definitively superior, as the possibility of finding a global minimum *cannot* be directly deduced from the mapping itself. Therefore, in this work, we conduct experiments with three (initial) mapping algorithms to determine the best-performing one.

Criteria to Differentiate among Mapping Algorithms: To distinguish between different mappings, we must identify the criteria affecting each mapping and the reliability of the resulting output. These criteria and their influences on the system can be summarized as follows:

- *Connectivity to qubits from the same program:* Assigning logical qubits from the same program to distant physical qubits can increase the number of CNOTs (due to SWAP operations), leading to higher gate errors. Hence, each program should be mapped to a strongly-connected group of physical qubits to minimize gate error effects.
- *Quality of qubits:* Different physical qubits may have varying error rates and numbers of links. A qubit with more links can enhance system performance if assigned to the logical qubit with the most CNOTs, reducing the number of SWAPs.
- *Number of gates:* A program with fewer gates is more optimized since it produces fewer gate errors.
- *Number and extent of crosstalks:* The number of distinct crosstalk cases indicates the extent of the crosstalks encountered; typically, the lower this number, the better the results. If



(a) (b) (c)

Fig. 2: (a) Updated flow to accommodate TRIM, and (b) the tree created by QuCloud [17], and (c) Crosstalk characterization needed for TRIM on ibmq_montreal for a sample 5-qubits benchmark.

we need a tie-breaker between two mappings (with the same amount of crosstalk), the magnitude of the crosstalk errors can be used to differentiate between them.

Since the effects of the above criteria on system reliability vary, a method for comparing their magnitudes should be developed. For example, a new CNOT gate causes less error than crosstalk; thus, reducing crosstalk should take precedence over optimizing gates.

V. Overall Design

In this section, we present our optimizations that focus on eliminating crosstalk errors during the mapping process. Current mapping approaches do *not* reduce crosstalk without increasing coherence errors; so, we propose TRIM, a novel mapping optimization algorithm that considers "crosstalk error characterization" to increase system reliability.

A. Crosstalk Characterization

The first step in incorporating crosstalk into the mapping process is to determine its effect on different qubits of the target quantum machine. The prior works [4], [28] indicate that crosstalk can vary dramatically from one qubit to another, thus necessitating a careful characterization of the target system to quantify its magnitude. To do this, prior work [4] schedules two CNOTs on adjacent qubits and calculates crosstalk by comparing the error rate against the scenario where these two qubits are operated independently. Note that, to reduce the total number of runs, multiple characterization scenarios can be executed simultaneously as long as they do not affect each other (when one crosstalk scenario between two CNOTs causes crosstalk in the other scenario). This process can take up to 10 hours for ibmq_montreal, which is significant.

To find crosstalk, we compare the qubits of each CNOT in each step to those of the others, detecting crosstalk between the qubits if they are neighbors. To make the procedure easier, a tree and an array containing the CNOTs qubits are used. More specifically, the physical qubits of *all* CNOTs are represented by an array in different steps, and by comparing the physical qubits of CNOTs with each other, we identify the crosstalk cases. In step *i*, for example, if two CNOTs are running at the same time, we compare the physical locations of the qubits to see if they are neighbors or not. If they are, we count it as crosstalk. Then, by executing three different jobs, one executing both CNOTs and the other two executing one CNOT at a time, we measure the difference in reliability of the system for each CNOT and report it as the crosstalk effect on that CNOT.

Algorithm 1: Relaxed QuCloud.

```
input  : Coupling Graph G, Quantum Circuits
output : Partitions
Relaxed_G ← G

Function Mapper:
    for circuit in circuits do
        Tree ← Tree_Constructor(Relaxed_G)      // creating the
          tree based on the hardware graph
        candidates ← []
        for leaf in leafs(Tree) do
            if circuit fits in leaf then
                candidates.add(leaf)
            else
                leaf ← leaf.parent
            end
        end
        Sort the candidates based on their intra_links/inter_links and choose the best to
        the circuit
        if circuit can fit in the chosen node then
            assign circuit to leaf
            Relaxed_G ← Graph_Updater(Relaxed_G, leaf)
        else
            Execute Separately
        end
    end

Function Graph_Updater(Graph_G):
for node in leaf do
    Graph_G.remove(node) Graph_G.remove(node.neighbors) return
      Graph_G
end

Function Tree_Constructor:
    Communities ← [] Create a leaf node for every qubit while
      Communities.size() > 1 do
        Find two-element combination (A,B) of communities for (A,B) in all
          the combinations do
            F(A,B) = Intra_Program_Links - Inter_Program_Links
        end
        New_Node = Union(A,B) Communities.remove(A)
          Communities.remove(B) Communities.add(New_Node)
    end
```

While our algorithm's execution also requires the characterization procedure, instead of characterizing the "entire" quantum hardware, we first apply the initial mapping to each of the single and multi-programmed benchmarks and gather a union of the chosen physical qubits. After that, TRIM characterizes the neighboring nodes within a two-node distance for the union of the qubits in the initial mappings (the physical qubits that are initial mappings use). Since most reliable qubits and links are *not* frequently changed, the mappings of most qubits are always identical, thus making the union of the mappings *not* significant in size. Also, although there may be a theoretical scenario in which the algorithm completes by remapping some

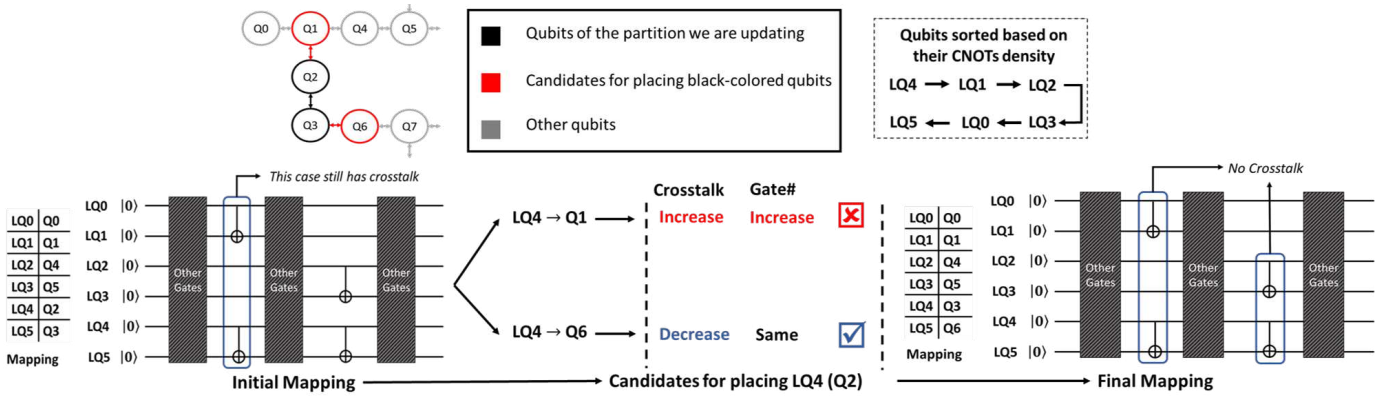


Fig. 3: An example illustrating how TRIM works in practice.

of the logical qubits to qubits with a distance greater than 2, in practice, this rarely occurs because it would increase the number of SWAPs, thereby reducing the PST of the algorithm. For example, as Figure 2c shows, for a sample initial mapping of an algorithm with 5 qubits (shown in blue color), we only characterize the crosstalk for the red portion of the graph (the red-colored links), while a complete characterization must characterize *all* the qubits and their links. Our results on all the benchmarks evaluated in Section VI indicate that in 96% of the cases, we do *not* need any additional characterization for the system by using our technique.

By leveraging our new characterization process and crosstalk characterization algorithm, we achieve a 5.6x reduction in crosstalk characterization overhead across all tested benchmarks. While a complete system characterization can be beneficial in some cases (4% of workloads), it will become infeasible in the future due to the increasing number of qubits in quantum systems. For instance, characterizing the new IBM quantum system with 128 qubits (ibm_washington) requires evaluating 242 scenarios, significantly more than the 27-qubit ibmq_montreal, which only needs characterization of 39 scenarios. That is, the larger machine spends 6.2x execution time only for characterization process. It is important to note that our characterization process is dependent on the number of links that the physically-mapped qubits have. Since the number of links for each qubit is typically limited to two or three in all IBM quantum systems we are aware of, the number of links that we need to characterize for the same workload using TRIM is almost identical in different quantum hardware with a different number of qubits, which shows our algorithm easily (and in a robust fashion) applies to different quantum hardware regardless of their qubit counts.

B. Modifications to Execution Flow

We modify the general flow of mapping and optimizations as, in its current form, it does *not* adequately address crosstalk minimization. The updated execution flow is shown in Figure 2a. The initial mapping (step ❷) will be performed without any adjustments. However, we aim to rectify the shortcomings of the general flow, in which mapping is conducted *only once* and without sufficient knowledge of circuit optimizations and crosstalk. To address this problem, we create a *feedback loop* (steps ❸-❼) that takes scheduling results, identifies crosstalk occurrences, and employs a heuristic strategy (discussed in detail in Section V-D) to find possible alternative mappings to eliminate crosstalk. This loop continues as long as a new mapping with improved reliability can be found. Once the loop terminates, the resulting mapping proceeds to step ❽, which consists of two parts. While our approach attempts to remove crosstalk, it can sometimes be unavoidable due to factors such as excessive qubit usage or poor initial mapping. Our method also includes an ***optional*** step (step ❽-a) to eliminate crosstalk using barriers, as proposed in [4]. This step ensures that TRIM will always outperform prior mapping proposals in terms of *both* reliability and execution time. Finally, in step ❽-b, instructions are executed, and the reliability is measured.

C. Initial Mapping

As mentioned earlier in Section IV, in our approach, the results generated from any existing mapping scheme can be used as the "initial mapping". However, our proposal will achieve the best result if the constraints[3] imposed by the initial mapping are minimal and the potential search space is large, thereby increasing chances to achieve a better PST. For this purpose, we also introduce a new (initial mapping) scheme, which is essentially a *relaxed version* of the mapping scheme proposed in QuCloud [17]. We want to emphasize that, although the choice of the initial mapping can affect the extent of the PST improvements achieved through our proposed approach, *TRIM is applicable to any existing mapping and improves upon it (in terms of PST).*

The QuCloud mapping scheme is based on creating a tightly-coupled group of nodes to minimize the number of SWAPs. To do so, first, by merging nodes into a group of tightly coupled communities, it creates a *tree*, as illustrated in Figure 2b, where each node captures the best possible community across its children's physical qubits. Criteria such as the ratio of intra-program to inter-program links and the total error rate of each community are used to implement this tree. Then, by sorting the input programs (in the target multiprogrammed workload), if we have multiple programs, based on their CNOTs

[3]Note that, if two programs are assigned to two neighboring strongly-connected groups of qubits, the search space would be small, decreasing the possibility of optimizing the initial mapping.

Algorithm 2: TRIM: our greedy mapping optimizer.

```
   input  : Partitions
   input  : Gate_TH
   ;      // Threshold for the increase in number of Gates.
   input  : CT_Info
   output : Updated Partitions

1  Function TRIM:
2      sorted_qubits ←
         qubits sorted based on their CNOT density
3      while sorted_qubits is not empty do
           /* find the neighbors of the qubits in the
              partition related to the head of the list.
              */
4          options ← partition[sorted_qubits.head()].neighbors;
5          for option in options do
6              New_Partition ←
                 Optimal_Partition.switch(sorted_qubits, option);
7              Num_CT, Num_Gates ← Scheduler(New_Partition);
               /* Condition 1                                  */
8              if Num_CT < Best_CT &
                 Num_Gates < Best_Gate + Gate_TH then
9                  Best_CT ← Num_CT;
10                 Best_Gate ← Num_Gates;
11                 Optimal_Partition ← New_Partition;
12                 sorted_qubits ←
                     Update based on new partition;
13                 break;
               /* Condition 2                                  */
14             else if Num_CT = Best_CT & Num_Gates < Best_Gate then
15                 Best_Gate ← Num_Gates;
16                 Optimal_Partition ← New_Partition;
17                 sorted_qubits ←
                     Update based on new partition;
18                 break;
               /* Condition 3                                  */
19             else if Num_CT = Best_CT & Num_Gates = Best_Gate then
20                 Total_CT_Optimal ← 1;
21                 for crosstalk in New_Partition do
22                     Total_CT_Optimal ←
                         Total_CT_Optimal × CT_Info[crosstalk];
23                 end
24                 for crosstalk in New_Partition do
25                     Total_CT_New ←
                         Total_CT_New × CT_Info[crosstalk];
26                 end
27                 if Total_CT_New < Total_CT_Optimal then
28                     Optimal_Partition ← New_Partition;
29                     sorted_qubits ←
                         Update based on new partition;
30                     break;
31                 else
32                     options.remove(option);
33                     continue;
34                 end
               /* Condition 4                                  */
35             else
36                 options.remove(option);
37                 continue;
38             end
39         end
           /* No better option is found.                       */
40         if options.empty() == 1 then
41             sorted_qubits.remove(head)
42         end
43     end
```

densities, it starts assigning them to different communities (with sufficient qubits), increasing the PST while minimizing the conflicts due to SWAPs. This approach minimizes the number of SWAPS by benefiting from strong connectivity between qubits of each program. Note however that, it can still lead to having communities located directly near each other, and this, unfortunately, reduces the search space. Additionally, although QuCloud effectively increases the PST of the system by using error rates as part of the tree creation, doing so would *not* be much useful for our approach since TRIM already aims to improve the PST by performing a greedy search strategy to identify a better mapping. Therefore, we relax the "tree creation step" in QuCloud by only creating the tree based on the intra-program qubits' connectivities.

The pseudo-code of the routines utilized in our relaxed-mapping strategy *R-QuCloud* (Relaxed QuCloud), designed to provide our algorithm with an expanded search space (over QuCloud), is given by Algorithm 1. This method generates partitions for the programs by obtaining the coupling graph of the hardware and the workloads we wish to run.

The overall algorithm contains three main routines. Our approach tries to discover the best feasible community across all nodes of the tree with enough qubits in the mapper stage (shown in line-2 to line-19). To begin, it stores all of the nominees/candidates and then chooses among them based on their $\frac{intra\_links}{inter\_links}$. If no choice for simultaneously executing multiple programs exists, the quantum programs in the target multiprogrammed workload are executed individually. Otherwise, the selected node receives the program. The tree is then updated using our **Graph Updater** (line-20 to line-23) and **Tree Constructor** (line-24 to line-30) functions. While QuCloud just updates the tree by eliminating the selected node and its children, our modified version takes a different approach by attempting to place the next program in a spot that does not have any other programs as neighbors. This is because we want to *expand* the search space of candidate mappings (possible mappings that we can traverse in our search of crosstalk-aware mappings; see Section V-D), so that our approach can traverse more communities and select the best among them. In the end, the algorithm returns the community assignment for *all* programs as its output.

D. TRIM: Our Greedy Mapping Optimizer

This section explains our approach to eliminate the crosstalk in the mapping process. Algorithm 2 gives the pseudo-code for the white box shown in Figure 2a. Our approach, shown in line-1 of the algorithm, already contains the router, scheduler, optimizer, and the TRIM search module previously mentioned in Section V-B. Algorithm 2 is invoked as soon as the initial mapping has been performed and it gradually minimizes the crosstalk errors. First, as shown in line-2, it sorts the qubits (based on their CNOT counts), to rank them according to their importance. Using this result, it then iteratively searches for better nominees from among the neighbors of each program's qubits. In this work, to compare two mappings (current one and new one), we have developed multiple criteria, which can be summarized as follows:

• **New mapping has lower crosstalk cases (lines 8-13):** In this case, we swap the qubit with its neighbor if the number of gates is not significantly higher (due to additional SWAPs). It is because crosstalk is frequently more important than a few CNOTs, thus justifying the revised mapping. However, significantly increasing the number of CNOTs could reduce the PST benefit we are getting from TRIM. As a result, we evaluate TRIM, in Section VI-C, using different upper-bounds for the gate number increase our algorithm allows.

		QuCloud [17]		QuCloud [17] + Murali et al. [4]		TRIM+R-QuCloud		TRIM+R-QuCloud+ Murali et al. [4]	
Case#	**Benchmark**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**
	WL	WL	Total	WL	Total	WL	Total	WL	Total
S1	P1	50.7%	209	53.7%	209	57.8%	197	62.3%	197
S2	P2	48.7%	77	49.3%	77	54.8%	77	55.0%	77
S3	P3	44.8%	278	44.4%	278	45.1%	260	45.1%	260
S4	P4	8.3%	255	11.0%	255	19.4%	234	22.1%	234
S5	P5	63.2%	276	68.2%	276	75.6%	255	80.4%	255
S6	P6	47.8%	83	40.0%	83	43.4%	71	46.4%	71
S7	P7	26.9%	80	27.2%	80	33.1%	92	33.4%	92
S8	P8	14.4%	278	17.8%	278	21.8%	272	22.3%	272
S9	P9	21.2%	1024	29.0%	1024	37.4%	1003	39.5%	1003
S10	P10	23.0%	443	27.0%	443	33.8%	437	38.7%	437
Average		34.9%	300.3	36.8%	300.3	42.2%	289.8	44.5%	289.8

			QuCloud [17]				QuCloud [17] + Murali et al. [4]				TRIM+R-QuCloud				TRIM+R-QuCloud+ Murali et al. [4]			
Case#	**Benchmarks**		**PST[%]**			**Gate#**	**PST[%]**			**Gate#**	**PST[%]**			**Gate#**	**PST[%]**			**Gate#**
	WL1	WL2	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total
M1	P10	P1	18.6%	41.7%	30.2%	727	22.4%	47.0%	34.7%	727	26.8%	49.5%	38.2%	703	29.3%	51.8%	40.6%	703
M2	P9	P1	17.7%	43.8%	30.7%	1305	20.7%	48.8%	34.7%	1305	27.5%	53.5%	40.5%	1293	28.8%	55.7%	42.3%	1293
M3	P8	P1	9.5%	45.1%	27.3%	281	14.3%	51.8%	33.0%	281	16.9%	54.4%	35.6%	302	18.5%	56.1%	37.3%	302
M4	P7	P1	24.6%	40.7%	32.7%	251	26.9%	44.4%	35.6%	251	32.3%	48.2%	40.3%	248	34.3%	50.2%	42.3%	248
M5	P6	P1	46.7%	42.6%	44.7%	269	48.5%	50.5%	49.5%	269	56.6%	51.2%	53.9%	239	58.8%	53.5%	56.2%	239
M6	P5	P1	58.6%	42.2%	50.4%	281	62.1%	47.1%	54.6%	281	65.6%	50.4%	58.0%	311	66.7%	51.4%	59.1%	311
M7	P4	P1	5.9%	41.8%	23.8%	239	7.4%	48.4%	27.9%	239	13.8%	50.0%	31.9%	230	15.3%	52.7%	34.0%	230
M8	P3	P1	40.3%	43.4%	41.9%	287	43.6%	50.7%	47.2%	287	43.4%	52.0%	47.7%	257	46.1%	54.8%	50.4%	257
M9	P10	P2	20.8%	40.8%	30.8%	601	25.5%	48.0%	36.8%	601	25.0%	49.1%	37.1%	598	26.9%	50.5%	38.7%	598
M10	P9	P2	19.4%	44.0%	31.7%	1128	21.1%	51.6%	36.4%	1128	24.9%	53.7%	39.3%	1110	27.9%	56.6%	42.2%	1110
M11	P8	P2	9.6%	42.2%	25.9%	394	12.5%	47.3%	29.9%	394	16.2%	50.2%	33.2%	376	19.0%	52.5%	35.8%	376
M12	P7	P2	24.7%	41.8%	33.2%	196	26.9%	47.4%	37.2%	196	32.7%	48.8%	40.7%	172	35.6%	50.9%	43.2%	172
M13	P6	P2	43.9%	44.1%	44.0%	247	47.5%	48.3%	47.9%	247	53.6%	52.2%	52.9%	220	55.6%	54.0%	54.8%	220
M14	P5	P2	61.7%	40.4%	51.1%	437	64.2%	48.0%	56.1%	437	67.9%	47.6%	57.8%	431	68.9%	50.1%	59.5%	431
M15	P4	P2	4.7%	41.9%	23.3%	416	7.1%	47.1%	27.1%	416	9.0%	51.7%	30.4%	407	10.5%	53.6%	32.0%	407
M16	P3	P2	42.5%	41.7%	42.1%	388	45.5%	45.2%	45.3%	388	51.7%	47.0%	49.4%	394	54.1%	48.5%	51.3%	394
Average			28.1%	42.4%	35.2%	465.4	31.0%	48.2%	39.6%	465.4	35.2%	50.6%	42.9%	455.7	37.3%	52.7%	45.0%	455.7

TABLE IV: Gate and PST comparison between TRIM and the baselines for single- and multi-programmed benchmarks. "WL" is abbreviation for workload.

• **The crosstalk counts are the same but number of gates is reduced (lines 14-18):** In this case, the new mapping is clearly beneficial and will lower the gate error rate of the program. As a result, the new mapping will be used as our mapping for the next step.

• **The crosstalk counts and the number of gates are the same (lines 19-34):** In this case, we compare two cases by considering the system's crosstalk characterization. To accomplish this, we compute the total crosstalk product of the crosstalk for the most optimized case we found so far and for the new mapping. We choose the new mapping as our new "optimized mapping" if it has a better crosstalk product; otherwise, we continue to use the current mapping. Note that we choose not to alter the mapping if the crosstalk characterization required for the comparison has not been previously computed.

• **Other cases (lines 39-42):** In all other remaining cases, we decide that the new mapping is not preferable.

If a new mapping passes these criteria, we add the corresponding SWAP operations, optimize the mapping, and schedule it. This strategy updates itself using the new partition, looking for better candidates in the instruction schedule. This process continues until no candidates are available in the graph. Additionally, since the number of neighbors of each partition of the qubits is not substantial in general, this process is *not* expected to take a large amount of time.

An example demonstrating how TRIM works in practice is given in Figure 3. As stated earlier, the input to TRIM is an initial mapping we would like to improve upon. In this case, the qubit we want to optimize is LQ4 (since it has the highest CNOT density), which is currently assigned to Q2 in the mapping. First, we locate all of the partition's neighbors (qubits that are used in the same program). Only Q1 and Q6 are available in this scenario. As a result, we calculate the number of SWAPs and gates for both cases (using routing and scheduling modules) when assigning LQ4 to these nodes. It can be seen that, when we assign LQ4 to Q1, we increase both the crosstalk and the number of gates, which is clearly *not* beneficial from the system's reliability standpoint. Another option is to assign LQ4 to Q6, which lowers the number of CNOTs while maintaining the number of gates. Therefore, we update the mapping accordingly and set it as the optimized baseline, which will be used in the subsequent step.

VI. Evaluation

A. Results

TRIM+R-QuCloud outperforms other baselines: As seen in Table IV, in comparison to QuCloud and QuCloud+ [4], TRIM+R-QuCloud achieves an average PST improvement of 7.7% and 3.3% for multi-programmed workloads, and 5.4% and 7.3% for single-programmed workloads, respectively. This improvement is notable because it is built upon an already heavily optimized baseline. TRIM enhances PST by decreasing crosstalk compared to QuCloud, which ignores crosstalk errors. In contrast to QuCloud+ [4], TRIM improves PST by reducing the number of gates while minimizing coherence error.

Combination of TRIM+R-QuCloud and [4] generates better results than all other approaches: TRIM and QuCloud+ [4] achieve 7.7% and 4.4% PST improvements for

		TRIM+R-QuCloud		TRIM+QuCloud [17]		TRIM+SABRE [11]		QuCloud [17]	
Case#	**Benchmark**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**	**PST[%]**	**Gate#**
	WL	WL	Total	WL	Total	WL	Total	WL	Total
S1	P1	57.8%	197	58.7%	197	59.7%	188	50.7%	209
S2	P2	54.8%	77	51.9%	89	49.9%	83	48.7%	77
S3	P3	45.1%	260	41.1%	269	38.1%	290	44.8%	278
S4	P4	19.4%	234	16.4%	240	15.7%	240	8.3%	255
S5	P5	75.6%	255	72.1%	246	72.9%	243	63.2%	276
S6	P6	43.4%	71	41.4%	65	39.0%	86	47.8%	83
S7	P7	33.1%	92	29.5%	98	25.7%	89	26.9%	80
S8	P8	21.8%	272	22.6%	290	23.2%	311	14.4%	278
S9	P9	37.4%	1003	37.0%	1018	37.9%	1036	21.2%	1024
S10	P10	33.8%	437	32.2%	434	28.8%	425	23.0%	443
Average		42.2%	289.8	40.3%	294.6	39.1%	299.1	34.9%	300.3

			TRIM+R-QuCloud				TRIM+QuCloud [17]				TRIM+SABRE [11]				QuCloud [17]			
Case#	**Benchmarks**		**PST[%]**			**Gate#**	**PST[%]**			**Gate#**	**PST[%]**			**Gate#**	**PST[%]**			**Gate#**
	WL1	WL2	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total	WL1	WL2	Avg	Total
M1	P10	P1	26.8%	49.5%	38.2%	703	26.6%	48.8%	37.7%	700	20.1%	43.5%	31.8%	787	18.6%	41.7%	30.2%	727
M2	P9	P1	27.5%	53.5%	40.5%	1293	25.9%	49.0%	37.4%	1305	18.7%	46.1%	32.4%	1293	17.7%	43.8%	30.7%	1305
M3	P8	P1	16.9%	54.4%	35.6%	302	13.5%	54.5%	34.0%	302	12.4%	47.3%	29.9%	308	9.5%	45.1%	27.3%	281
M4	P7	P1	32.3%	48.2%	40.3%	248	29.8%	47.3%	38.5%	263	27.1%	41.9%	34.5%	326	24.6%	40.7%	32.7%	251
M5	P6	P1	56.6%	51.2%	53.9%	239	54.1%	50.3%	52.2%	236	49.1%	45.1%	47.1%	299	46.7%	42.6%	44.7%	269
M6	P5	P1	65.6%	50.4%	58.0%	311	67.6%	50.7%	59.1%	299	59.6%	43.9%	51.7%	266	58.6%	42.2%	50.4%	281
M7	P4	P1	13.8%	50.0%	31.9%	230	15.3%	51.0%	33.1%	227	7.0%	43.9%	25.4%	275	5.9%	41.8%	23.8%	239
M8	P3	P1	43.4%	52.0%	47.7%	257	43.6%	49.6%	46.6%	275	42.1%	44.9%	43.5%	305	40.3%	43.4%	41.9%	287
M9	P10	P2	25.0%	49.1%	37.1%	598	22.9%	46.9%	34.9%	610	23.2%	43.8%	33.5%	586	20.8%	40.8%	30.8%	601
M10	P9	P2	24.9%	53.7%	39.3%	1110	24.1%	50.7%	37.4%	1134	22.4%	46.1%	34.3%	1143	19.4%	44.0%	31.7%	1128
M11	P8	P2	16.2%	50.2%	33.2%	376	17.2%	49.9%	33.6%	369	12.6%	43.3%	28.0%	385	9.6%	42.2%	25.9%	394
M12	P7	P2	32.7%	48.8%	40.7%	172	29.6%	48.3%	38.9%	181	27.4%	42.8%	35.1%	256	24.7%	41.8%	33.2%	196
M13	P6	P2	53.6%	52.2%	52.9%	220	52.1%	47.9%	50.0%	232	46.4%	45.3%	45.8%	298	43.9%	44.1%	44.0%	247
M14	P5	P2	67.9%	47.6%	57.8%	431	63.6%	42.8%	53.2%	434	64.5%	42.0%	53.2%	422	61.7%	40.4%	51.1%	437
M15	P4	P2	9.0%	51.7%	30.4%	407	10.4%	46.1%	28.2%	410	6.2%	44.2%	25.2%	419	4.7%	41.9%	23.3%	416
M16	P3	P2	51.7%	47.0%	49.4%	394	52.1%	45.6%	48.8%	394	44.3%	43.0%	43.7%	409	42.5%	41.7%	42.1%	388
Average			35.2%	50.6%	42.9%	455.7	34.3%	48.7%	41.5%	460.7	30.2%	44.2%	37.2%	486.1	28.1%	42.4%	35.2%	465.4

TABLE V: Gate and PST comparison for different initial mappings. "WL" means workload.

multi-programmed workloads and 7.3% and 1.9% for single-programmed workloads over the QuCloud baseline, respectively. As TRIM and [4] are orthogonal, targeting mapping and scheduling, their combination further improves PST in cases where crosstalk persists after mapping. TRIM minimizes crosstalk through mapping, while [4] eliminates remaining crosstalk by increasing execution time. Table IV shows a 9.8% and 9.4% PST improvement for multi-programmed and single-programmed cases, respectively, when combining TRIM and [4], indicating that the scheduler handles some crosstalk cases remaining after mapping.

Number of gates and PST are not strictly correlated: As shown in Table IV, the PST results are not strictly correlated with the number of gates since the coherence and crosstalk errors are also a factor in shaping the overall PST of the quantum system. It is worth noting that adding an extra CNOT can result in an error rate of roughly 10^{-2}, whereas crosstalk error can cause an error rate increase of 10x (10^{-1}), demonstrating the importance of limiting the crosstalk error.

Search algorithm in some cases can lead to a better mapping with a lower number of CNOTs: In many cases (like C1), the number of gates is also reduced, meaning that a better result is achieved by TRIM. This is because our proposal tries to adjust and swap qubits until a better location is found. Therefore, using line-14 of our algorithm, we search for a group of qubits that have not been considered by the prior works.

B. Comparison of Different Mappings

In order to carefully evaluate different mapping schemes, we perform a study that measures how TRIM performs on different initial mappings like R-QuCloud, QuCloud, and SABRE. Below, we summarize our main observations.

TRIM optimizes all the mappings to achieve a better qubit allocation: As the results in Table V indicate, TRIM optimizes the PST by taking into account crosstalk and minimizing gate error when compared against all the baselines tested. That is, *given any initial mapping, TRIM is able to improve upon it, demonstrating its robustness.*

TRIM+R-QuCloud achieves, on average, better performance compared to other baselines: When compared to TRIM+QuCloud and TRIM+SABRE, it can be seen that TRIM+R-QuCloud delivers an average PST improvement of 1.4% and 5.7%, respectively, across all multi-programmed workloads, and 1.9% and 3.1% for single-programmed benchmarks. This is because, R-QuCloud gives our approach an expanded search space to find crosstalk-aware mappings.

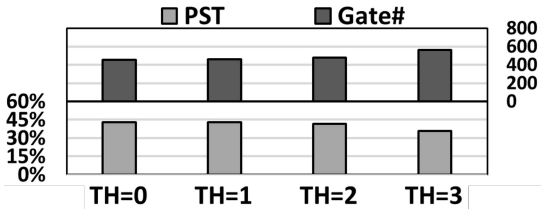


Fig. 4: PST and number of gates for Different gate threshold. All the results for TRIM are reported at top of R-QuCloud.

In a few cases, TRIM+R-QuCloud results in a lower PST: It is also worth noting that, due to the probabilistic nature of the mapping and since greedy algorithms do not always

produce the best results, other initial mappings can sometimes produce better final qubit allocations. However, even in such cases, the difference is minor (0.4%-1.3%), indicating that our initial mapping can operate quite efficiently in all cases.

C. Heuristic Gate Threshold Sensitivity Analysis

In Algorithm 2, we avoid changing the mapping if the increase in the number of gates exceeds a threshold, as this may negate crosstalk reduction benefits. Results presented so far used a threshold of 0. We also tested thresholds of 1, 2, and 3, as shown in Figure 4. Our findings reveal that increasing the threshold sometimes improves PST, but may also lower PST when the number of gates increases significantly. On average, a threshold of 0 yields the best results, suggesting that increasing the number of gates can counteract overall improvements.

VII. Conclusion

We introduce a new mapping optimizer called TRIM that addresses crosstalk errors in quantum workloads. We evaluate TRIM using 16 multi-programmed and 10 single-programmed quantum benchmarks against several optimized baselines. Our experimental results reveal that in multi-programmed workloads, TRIM achieves average PST improvements of 7.7% and 3.3% compared to QuCloud and QuCloud+ [4] while maintaining or decreasing the number of gates. For single-programmed workloads, TRIM achieves PST improvements of 7.3% and 5.4% compared to QuCloud and [4], respectively.

Acknowledgement

We acknowledge the use of IBM Quantum services. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team. AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

References

[1] Roman Orus, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, 2019.

[2] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19):10856–10915, 2019.

[3] Peter Wittek. *Quantum machine learning: what quantum computing means to data mining*. Academic Press, 2014.

[4] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1016, 2020.

[5] Eric M Kessler, Igor Lovchinsky, Alexander O Sushkov, and Mikhail D Lukin. Quantum error correction for metrology. *Physical review letters*, 112(15):150802, 2014.

[6] Ling Hu, Yuwei Ma, Weizhou Cai, Xianghao Mu, Yuan Xu, Weiting Wang, Yukai Wu, Haiyan Wang, YP Song, C-L Zou, et al. Quantum error correction and universal gate set operation on a binomial bosonic logical qubit. *Nature Physics*, 15(5):503–508, 2019.

[8] Lingling Lao and Carmen G Almudever. Fault-tolerant quantum error correction on near-term quantum processors using flag and bridge qubits. *Physical Review A*, 101(3):032333, 2020.

[7] Soraya Taghavi, Robert L Kosut, and Daniel A Lidar. Channel-optimized quantum error correction. *IEEE transactions on information theory*, 56(3):1461–1473, 2010.

[9] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Qecool: On-line quantum error correction with a superconducting decoder for surface code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 451–456. IEEE, 2021.

[10] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[11] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[12] Swamit S Tannu and Moinuddin K Qureshi. Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 987–999, 2019.

[13] Swamit S Tannu and Moinuddin K Qureshi. A case for variability-aware policies for nisq-era quantum computers. *arXiv preprint arXiv:1805.10224*, 2018.

[14] Siyuan Niu, Adrien Suau, Gabriel Staffelbach, and Aida Todri-Sanial. A hardware-aware heuristic for the qubit mapping problem in the nisq era. *IEEE Transactions on Quantum Engineering*, 1:1–14, 2020.

[15] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. Quantum circuit optimizations for nisq architectures. *Quantum Science and Technology*, 5(2):025010, 2020.

[16] Zhenyu Cai. Multi-exponential error extrapolation and combining error mitigation techniques for nisq applications. *npj Quantum Information*, 7(1):1–12, 2021.

[17] Lei Liu and Xinglei Dou. Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 167–178. IEEE, 2021.

[18] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. A case for multi-programming quantum computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 291–303, 2019.

[19] W. M. McKeeman. Peephole optimization. *Commun. ACM*, 8(7):443–444, jul 1965.

[20] B. Kraus and J. I. Cirac. Optimal creation of entanglement using a two-qubit gate. *Physical Review A*, 63(6), may 2001.

[21] D. Maslov, G.W. Dueck, D.M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, mar 2008.

[22] Ji Liu, Peiyi Li, and Huiyang Zhou. Not all swaps have the same cost: A case for optimization-aware qubit routing. In *2022 IEEE Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022.

[23] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[24] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation. *arXiv preprint arXiv:2005.13018*, 2021.

[25] Ibm quantum. https://quantum-computing.ibm.com/, 2021.

[26] MD SAJID ANIS et al. Qiskit: An open-source framework for quantum computing, 2021.

[27] Siyuan Niu and Aida Todri-Sanial. Enabling multi-programming mechanism for quantum computing in the nisq era. *arXiv preprint arXiv:2102.05321*, 2021.

[28] Abdullah Ash Saki and Swaroop Ghosh. Qubit sensing: A new attack model for multi-programming quantum computing. *arXiv preprint arXiv:2104.05899*, 2021.