Enabling multi-programming mechanism for quantum computing in the NISQ era

Siyuan Niu, Aida Todri-Sanial

Enabling Multi-programming Mechanism for Quantum Computing in the NISQ Era

Siyuan Niu[1] and Aida Todri-Sanial[2,3]

[1]LIRMM, University of Montpellier, 34095 Montpellier, France
[2]LIRMM, University of Montpellier, 34095 Montpellier, CNRS, France
[3]Eindhoven University of Technology, 5612 AE, Eindhoven, Netherlands

NISQ devices have several physical limitations and unavoidable noisy quantum operations, and only small circuits can be executed on a quantum machine to get reliable results. This leads to the quantum hardware under-utilization issue. Here, we address this problem and improve the quantum hardware throughput by proposing a Quantum Multi-programming Compiler (QuMC) to execute multiple quantum circuits on quantum hardware simultaneously. This approach can also reduce the total runtime of circuits. We first introduce a parallelism manager to select an appropriate number of circuits to be executed at the same time. Second, we present two different qubit partitioning algorithms to allocate reliable partitions to multiple circuits – a greedy and a heuristic. Third, we use the Simultaneous Randomized Benchmarking protocol to characterize the crosstalk properties and consider them in the qubit partition process to avoid the crosstalk effect during simultaneous executions. Finally, we enhance the mapping transition algorithm to make circuits executable on hardware using a decreased number of inserted gates. We demonstrate the performance of our QuMC approach by executing circuits of different sizes on IBM quantum hardware simultaneously. We also investigate this method on VQE algorithm to reduce its overhead.

1 Introduction

Quantum computing promises to achieve an exponential speedup to tackle certain computational tasks compared with the classical computers [20, 21, 36]. Quantum technologies are continuously improving, and IBM recently released the largest quantum chip with 127 qubits. But, current quantum devices are still qualified as Noisy Intermediate-Scale Quantum (NISQ) hardware [32], with several physical constraints. For example, for superconducting devices, which we target in this paper, connections are only allowed between two neighbouring qubits. Besides, the gate operations of NISQ devices are noisy and have unavoidable error rates. As we do not have enough number of qubits to realize Quantum Error Correction [5], only small circuits with limited depth can obtain reliable results when executed on quantum hardware, which leads to a waste of hardware resources.

With the growing demand to access quantum hardware, several companies such as IBM, Rigetti, and IonQ provide cloud quantum computing systems enabling users to execute

Siyuan Niu: siyuan.niu@lirmm.fr
Aida Todri-Sanial: a.todri.sanial@tue.nl

their jobs on a quantum machine remotely. However, cloud quantum computing systems have some limitations. First, there exists a latency when submitting jobs. Second, there are a large number of jobs pending on the quantum device in general, so that users need to spend a long time waiting in the queue.

The low hardware usage and long waiting time lead to a timely issue: how do we use quantum hardware more efficiently while maintaining the circuit fidelity? As the increase of hardware qubit number and the improvement of qubit error rates, the multi-programming problem was introduced by [10, 23] to address this issue. It has been demonstrated that the utilization (usage/throughput) of NISQ hardware can be enhanced by executing several circuits at the same time. However, their results showed that when executing multiple quantum circuits simultaneously, the activity of one circuit can negatively impact the fidelity of others, due to the difficulty of allocating reliable regions to each circuit, higher chance of crosstalk error, etc. Previous works [10, 23] have left these issues largely unexplored and have not addressed the problem holistically such that the circuit fidelity reduction cannot be ignored when executing simultaneously. Moreover, detrimental crosstalk impact for multiple parallel instructions has been reported in [1, 2, 26] by using Simultaneous Randomized Benchmarking (SRB) [14]. In the presence of crosstalk, gate error can be increased by an order of magnitude. Ash-Saki et al. [1] even proposed a fault-attack model using crosstalk in a multi-programming environment. Therefore, crosstalk is considered in the multi-programming framework [29].

Multi-programming, if done in an ad-hoc way would be detrimental to fidelity, but if done carefully, it can be a very powerful technique to enable parallel execution for important quantum algorithms such as Variational Quantum Algorithms (VQAs) [6]. For example, the multi-programming mechanism can enable to execute several ansatz states in parallel in one quantum processor, such as in Variational Quantum Eigensolver (VQE) [19, 31], Variational Quantum Linear Solver (VQLS) [4], or Variational Quantum Classifier (VQC) [17] with reliability. It is also general enough to be applied to other quantum circuits regardless of applications or algorithms. More importantly, it can build the bridge between NISQ devices to large-scale fault-tolerant devices.

In this work, we address the problem of multi-programming by proposing a novel Quantum Multi-programming Compiler (QuMC), taking the impact of hardware topology, calibration data, and crosstalk into consideration. Our major contributions can be listed as follows:

- We introduce a parallelism manager that can select the optimal number of circuits to execute simultaneously on the hardware without losing fidelity.
- We design two different qubit partition algorithms to allocate reliable partitions to multiple circuits. One is greedy, which provides the optimal choices. The other one is based on a heuristic that can give nearly optimal results and significantly reduce the time complexity.
- We consider crosstalk effect during the partition process to achieve crosstalk mitigation during simultaneous executions. This is the first crosstalk-aware partition algorithm.
- We improve the mapping transition step to execute multiple quantum circuits on quantum hardware with a reduced number of additional gates and better fidelity.
- We present a use case of applying our multi-programming framework to the VQE algorithm to reduce its overhead, which demonstrates the application of multi-programming on NISQ algorithms.

Figure 1: An example of the multi-programming mechanism. (a) A four-qubit circuit is executed on a 10-qubit device. The hardware throughput is 40%. (b) Two four-qubit circuits are executed on the same device in parallel. The hardware throughput becomes 80%.

We evaluate our algorithm on real quantum hardware by first executing circuits of different sizes at the same time, and then investigating it on VQE to estimate the ground state energy of deuteron. To the best of our knowledge, this is the first attempt to propose a complete multi-programming process flow for executing an optimal number of workloads in parallel ensuring the output fidelity by analyzing the hardware limitations, and the first demonstration of multi-programming application on NISQ algorithms.

2 Background

2.1 NISQ computing

Quantum computing has made huge progress in recent years. IBM launched the first cloud-based quantum computing service with a 5-qubit quantum machine in 2016, and the hardware qubit number reached 127 in only five years. In the meanwhile, the capabilities and error rates of the quantum hardware are continuously improving such that the Quantum Volume [9] arrived 128 for IBM quantum machines. However, today's quantum computers are considered as NISQ devices yet. The hardware topology is limited and the qubits are prone to different errors, such as (1) coherent errors due to the fragile nature of qubits, (2) operational errors including gate errors and measurement errors (readout errors), (3) crosstalk errors that violate the isolated qubit state due to the parallel operations on other qubits. NISQ computing still promises to realize quantum advantages using variational quantum algorithms despite the errors. Cloud-based quantum computing services facilitate researchers and developers to work in this area. However, it causes some online traffic. For example, there are usually more than 100 jobs pending on IBM Q 27 Toronto, which requires several hours to retrieve the result. Therefore, efficient and reliable cloud quantum computing services are demanded while taking good care of hardware utilization and qubit errors.

2.2 Multi-programming mechanism

The idea of the multi-programming mechanism is quite simple: executing several quantum circuits in parallel on the same quantum hardware. An example is shown in Fig. 1. Note that, the simultaneous circuits can always be scheduled using As Late As Possible (ALAP) method, allowing qubits to remain in the ground state as long as possible to avoid additional decoherence error caused by circuits with different depths. Since the waiting time is usually much longer than the circuit execution time, the difference between execution time for circuits with different depths can be ignored (see experimental demonstration in Section 8.2). By executing two circuits at the same time, the hardware throughput doubles and the total runtime (waiting time + execution time) is reduced twice. It is not trivial to achieve the multi-programming mechanism. The main concern is how to trade-off between

the circuit output fidelity and the hardware throughput (also indicates the reduction of total runtime). Even though it is possible to simply combine several programs to one large circuit and compile it directly, it has been shown in [23] that the circuit fidelity is decreased significantly due to the unfair allocation of partitions, unawareness of increased crosstalk, inflexibility of reverting back to independent executions for the case of serious fidelity drop, etc. Therefore, a new compilation technique for the multi-programming mechanism is required. Several problems need to be addressed to enable the multi-programming mechanism: (1) Find an appropriate number of circuits to be executed simultaneously such that the hardware throughput is improved without losing fidelity. (2) Allocate reliable partitions of the hardware to all the simultaneous circuits to make them execute with high fidelity. (3) Transform multiple circuits to make them executable on the hardware. (4) Reduce the interference between simultaneous circuit executions to lower the impact of crosstalk.

2.3 State of the art

The multi-programming mechanism was first proposed in [10] by developing a Fair and Reliable Partitioning (FRP) method. Liu et al. improved this mechanism and introduced QuCloud [23]. There are some limitations for the two works: (1) Hardware topology and calibration data are not fully analyzed, such that allocation is sometimes done on unreliable or sparse-connected partitions ignoring the robust qubits and links. (2) These works use only `SWAP` gate for the mapping transition process and the modified circuits always have a large number of additional gates. (3) Crosstalk is not considered when allocating partitions for circuits. For example, the X-SWAP scheme [23] can only be performed when circuits are allocated to neighbouring partitions, which is the case of more crosstalk. Ohkura et al. designed palloq [29], a crosstalk detection protocol that reveals the crosstalk impact on multi-programming. A similar idea of Concurrent Quantum Circuit Sampling (CQCS) [34] was proposed to increase the hardware usage by executing multiple instances of the same program simultaneously. The concept of multi-programming was also explored in quantum annealers of DWAVE systems to solve several QUBO instances in parallel [30].

In our work, we focus on the multi-programming mechanism and propose QuMC framework with different crosstalk-aware partition methods and mapping transition algorithm to increase the hardware usage while maintaining the circuit fidelity.

3 Our multi-programming framework

Our proposed QuMC workflow is schematically shown in Fig. 2, which includes the following steps:

- Input layer. It contains a list of small quantum circuits written in OpenQASM language [8], and the quantum hardware information, including the hardware topology, calibration data, and crosstalk effect.
- Parallelism manager. It can determine whether executing circuits concurrently or separately. If the simultaneous execution is allowed, it can further decide the number of circuits to be executed on the hardware at the same time without losing fidelity based on the fidelity metric included in the hardware-aware multi-programming compiler.
- Hardware-aware multi-programming compiler. Qubits are partitioned to several reliable regions and are allocated to different quantum circuits using qubit partition

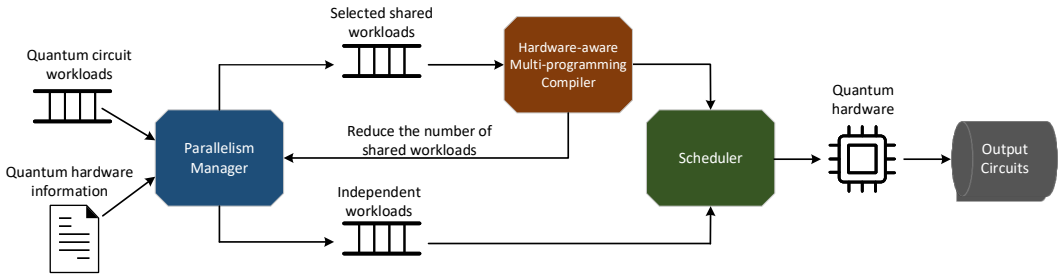


Figure 2: Overview of our proposed QuMC framework. The input layer includes the quantum hardware information and multiple quantum circuit workloads. The parallelism manager decides whether to execute circuits simultaneously or independently. For simultaneous executions, it works with the hardware-aware multi-programming compiler to select an optimal number of shared workloads to be executed in parallel. These circuits are allocated to reliable partitions and then passed to the scheduler. It makes all the circuits executable on the quantum hardware and we can obtain the results of the output circuits.

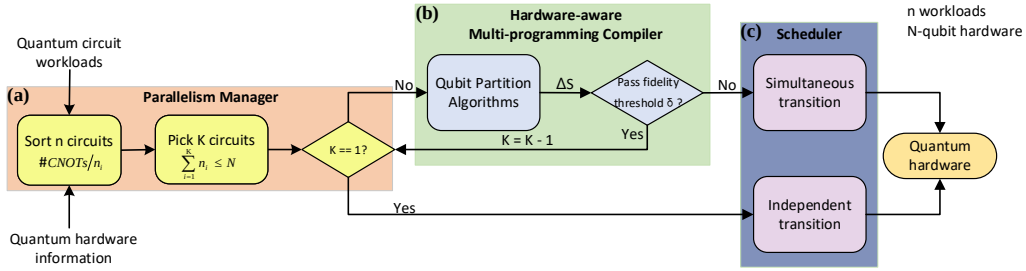


Figure 3: Process flow of each block that constitutes our QuMC approach. (a) The parallelism manager selects K circuits according to their densities and passes them to the hardware-aware multi-programming compiler. (b) The qubit partition algorithms allocate reliable regions to multiple circuits. ΔS is the difference between partition scores when partitioning independently and simultaneously, which is the fidelity metric. δ is the threshold set by the user. The fidelity metric helps to select the optimal number of simultaneous circuits to be executed. (c) The scheduler performs mapping transition algorithm and makes quantum circuits executable on real quantum hardware.

algorithms. Then, the partition fidelity is evaluated by the post qubit partition process. We introduce a fidelity metric here, which helps to decide whether this number of circuits can be executed simultaneously or the number needs to be reduced.

- Scheduler. The mapping transition algorithm is applied and circuits are transpiled to be executable on real quantum hardware.
- Output layer. Output circuits are executed on the quantum hardware simultaneously or independently according to the previous steps and the experimental results are obtained.

In this paper, we only focus on IBM quantum architecture. Our QuMC method can be generally adapted to quantum hardware with nearest-neighbor connectivity and also allows parallel operations if applied to different qubits.

4 Parallelism manager

In order to determine the optimal number of circuits that can be executed on the hardware in parallel without losing fidelity, here, we introduce the parallelism manager, shown in

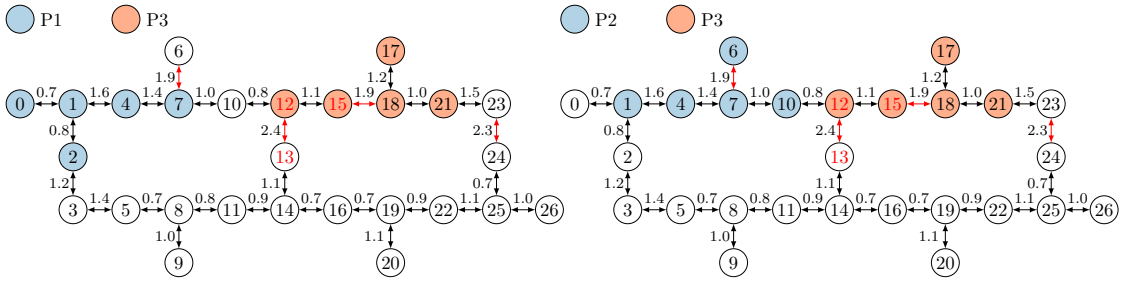


Figure 4: A motivational example of qubit partition problem. (a) No crosstalk between partition P1 and partition P3. (b) Crosstalk exists between partition P2 and partition P3.

Fig. 3(a).

Suppose we have a list of n circuit workloads with n_i qubits for each of them, that are expected to be executed on N-qubit hardware. We define the circuit density metric as the number of `CNOTs` divided by the qubit number of the circuit, $\#CNOTs/n_i$, and the circuit with higher density is considered to be more subject to errors. Firstly, the circuits are ordered by their "density" metric. Note that, the users can also customize the order of circuits if certain circuits are preferred to have higher fidelities. Then, we pick K circuits as the maximum number of circuits that can be executed on the hardware at the same time, $\sum_{n=1}^{K} n_i \leq N$. If K is equal to one, then all the circuits should be executed independently. Otherwise, these circuits are passed to the hardware-aware multi-programming compiler. It works together with the parallelism manager to decide an optimal number of simultaneous circuits to be executed.

5 Hardware-aware multi-programming compiler

The hardware-aware multi-programming compiler contains two steps. First, perform qubit partitioning algorithm to allocate reliable partitions to multiple circuits. Second, compute the fidelity metric during post qubit partition process and work with parallelism manager to determine the number of simultaneous circuits.

5.1 Qubit partition

We develop two qubit partition algorithms by accounting for the crosstalk, hardware topology, and calibration data. In this section, we first introduce a motivational example for qubit partition. Second, we explain the approach for crosstalk characterization. Finally, we present two qubit partition algorithms, one greedy and one heuristic.

5.1.1 Motivational example

We consider two constraints when executing multiple circuits concurrently. First, each circuit should be allocated to a partition containing reliable physical qubits. Allocated physical qubits (qubits used in hardware) can not be shared among quantum circuits. Second, qubits can be moved only inside of their circuit partition during the routing process, in other words, qubits can be swapped within the same partition only. Note that, in this work, we performed routing inside of the reliable partition but other approaches can be

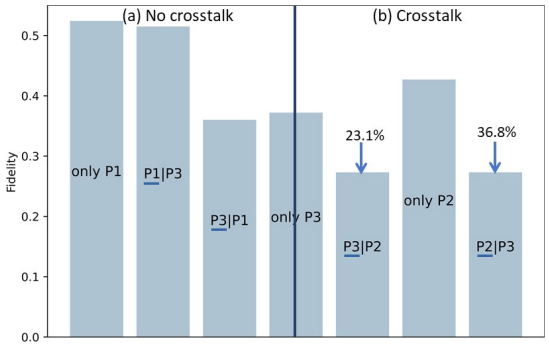


Figure 5: Results of the motivational example. (a) No crosstalk corresponds to Fig. 4(a) where no crosstalk exists between P1 and P3. (b) Crosstalk corresponds to Fig. 4(b) where crosstalk exists between P2 and P3. Note that "only P1" means the fidelity of the circuit when it is executed independently on P1, whereas "P1|P3" means the fidelity of circuit on P1 when two circuits are executed on P1 and P3 simultaneously.

applied as well such as to route to other neighboring qubits that are outside of the reliable partition.

Finding reliable partitions for multiple circuits is an important step in the multi-programming problem. In order to illustrate the impact of partitions with different error sources on the output fidelity, first, we execute a small circuit `alu-v0_27` (the information of this circuit can be found in Table 3) on three different partitions independently to show the impact of operational error (including `CNOT` error and readout error): (1) Partition P1 with reliable qubits and links. (2) Partition P2 with unreliable links. (3) Partition P3 with unreliable links and qubits with high readout error rate. Note that, the `CNOT` error rate of each link is shown in Fig. 4 and the unreliable links with high `CNOT` error rates and qubits with high readout error rates are highlighted in red. Second, we execute two of the same circuits simultaneously to show the crosstalk effect: (1) P1 and P3 without crosstalk (Fig. 4(a)). (2) P2 and P3 with crosstalk (Fig. 4(b)). For the sake of fairness, each partition has the same topology. It is important to note that if we have different topologies, the circuit output fidelity will also be different since the number of additional gates is strongly related to the hardware topology.

The result of the motivational example is shown in Fig. 5. The fidelity is calculated using PST metric explained in Section 7.1.1 and higher is better. For independent execution, we have P1 $>$ P2 $>$ P3 in terms of fidelity, which shows the influence of operational error on output fidelity. For simultaneous execution, the circuit fidelities are approximately the same for the two partitions P1 and P3 compared with the independent execution in the case of no crosstalk. Whereas, the fidelities are decreased by 36.8% and 23.1% respectively for P2 and P3 when the two circuits are executed simultaneously due to the crosstalk. This example demonstrates the importance of considering crosstalk effect in the multi-programming mechanism.

5.1.2 Crosstalk effect characterization.

Crosstalk is one of the major noise sources in NISQ devices, which can corrupt a quantum state due to quantum operations on other qubits [35]. There are two types of crosstalk. The first one is quantum crosstalk, which is caused by the always-on-ZZ interaction [24, 42]. The second one is classical crosstalk caused by the incorrect control of the qubits. The calibration data provided by IBM do not include the crosstalk error. To consider

the crosstalk effect in partition algorithms, we must first characterize it in the hardware. There are several protocols presented in [3, 12, 14, 33] to benchmark the crosstalk effect in quantum devices. In this paper, we choose the mostly used protocol – Simultaneous Randomized Benchmarking (SRB) [14] to detect and quantify the crosstalk between CNOT pairs when executing them in parallel.

We characterize the crosstalk effect followed by the optimization methods presented in [26]. On IBM quantum devices, the crosstalk effect is significant only at one hop distance between CNOT pairs [26], such as $(CX_{0,1}|CX_{2,3})$ shown in Fig. 6(a), when the control pulse of one qubit propagates an unwanted drive to the nearby qubits that have similar resonate frequencies. Therefore, we perform SRB only on CNOT pairs that are separated by one-hop distance. For those pairs whose distance is greater than one hop, the crosstalk effects are very weak and we ignore them. It allows us to parallelize SRB experiments of multiple CNOT pairs when they are separated by two or more hops. For example, in IBM Q 27 Toronto, the pairs $(CX_{0,1}|CX_{4,7})$, $(CX_{12,15}|CX_{17,18})$, $(CX_{5,8}|CX_{11,14})$ can be characterized in parallel.

Previous works [1, 26, 27] show that, although the absolute gate errors vary every day, the pairs that have strong crosstalk effect remain the same across days. We confirm that validation by performing the crosstalk characterization on IBM Q 27 Toronto twice and we observe the similar behavior. The SRB experiment on CNOT pairs $(g_i|g_j)$ gives error rate $E(g_i|g_j)$ and $E(g_j|g_i)$. Here, $E(g_i|g_j)$ represents the correlated CNOT error rate of g_i when g_i and g_j are executed in parallel. If there is a crosstalk effect between the two pairs, it will lead to $E(g_i|g_j) > E(g_i)$ or $E(g_j|g_i) > E(g_j)$. The crosstalk effect characterization is expensive and time costly. Some of the pairs do not have crosstalk effect whereas the correlated CNOT error affected the most by crosstalk effect is increased by more than five times. Therefore, we extract the pairs with significant crosstalk effect, i.e., $E(g_i|g_j) > 3 \times E(g_i)$ and only characterize these pairs when crosstalk properties are needed. We choose the same factor 3 to quantify the pairs with strong crosstalk error like [26]. The result of crosstalk effect characterization on IBM Q 27 Toronto is shown in Fig. 6(b).

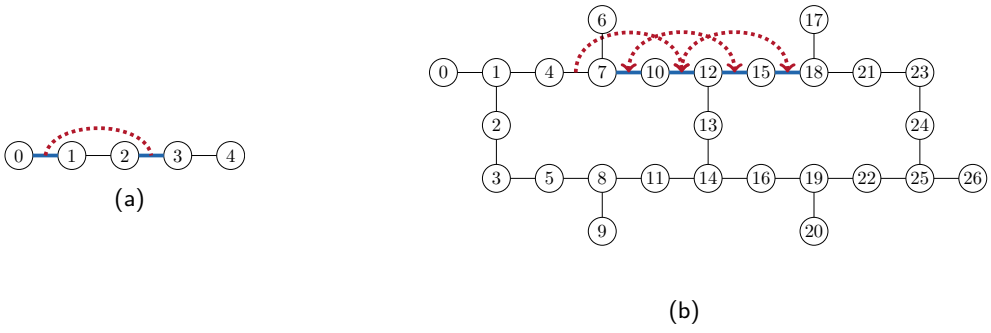


Figure 6: Characterization of crosstalk effect. (a) Crosstalk pairs separated by one-hop distance. The crosstalk pairs should be able to be executed at the same time. Therefore, they cannot share the same qubit. One-hop is the minimum distance between crosstalk pairs. (b) Crosstalk effect results of IBM Q 27 Toronto using SRB. The arrow of the red dash line points to the CNOT pair that is affected significantly by crosstalk effect, e.g., $CX_{7,10}$ and $CX_{12,15}$ affect each other when they are executed simultaneously. In our experiments, $E(CX_{10,12}|CX_{4,7}) > 3 \times E(CX_{10,12})$, whereas $E(CX_{4,7}|CX_{10,12}) \approx 1.5 \times E(CX_{4,7})$. As we choose 3 as the factor to pick up pairs with strong crosstalk effect, there is no arrow at pair $CX_{4,7}$.

5.1.3 Greedy sub-graph partition algorithm.

We develop a Greedy Sub-graph Partition algorithm (GSP) for qubit partition process which is able to provide the optimal partitions for different quantum circuits. The first step of the GSP algorithm is to traverse the overall hardware to find all the possible partitions for a given circuit. For example, suppose we have a five-qubit circuit, we find all the subgraphs of the hardware topology (also called coupling graph) containing five qubits as the partition candidates. Each candidate has a score to represent its fidelity depending on the topology and calibration data. The partition with the best fidelity is selected and all the qubits inside of the partition are marked as used qubits so they cannot be assigned to other circuits. For the next circuit, a subgraph with the required number of qubits is assigned and we check if there is an overlap on this partition to partitions of previous circuits. If not, the subgraph is a partition candidate for the given circuit and the same process is applied to each subsequent circuit. To account for crosstalk, we check if any pairs in a subgraph have strong crosstalk effect caused by the allocated partitions of other circuits. If so, the score of the subgraph is adjusted to take crosstalk error into account.

In order to evaluate the reliability of a partition, three factors need to be considered: partition topology, error rates of two-qubit links, and readout error of each qubit. One-qubit gates are ignored for simplicity and because of their relatively low error rates compared to the other quantum operations. If there is a qubit pair in a partition that has strong crosstalk affected by other partitions, the `CNOT` error of this pair is replaced by the correlated `CNOT` error which takes crosstalk into account. Note that the most recent calibration data should be retrieved through the IBM Quantum Experience before each usage to ensure that the algorithm has access to the most accurate and up-to-date information. To evaluate the partition topology, we determine the longest shortest path (also called graph diameter) of the partition, denoted L. The smaller the longest shortest path is, the better the partition is connected. Eventually, fewer additional gates would be needed to connect two qubits in a well-connected partition.

Algorithm 1 GSP algorithm

Input: Quantum circuit QC , Coupling graph G, Calibration data C, Crosstalk properties crosstalk_props, Used_qubits q_{used}

Output: A list of candidate partitions sub_graph_list

```
qubit_num ← QC.qubit_num
Set sub_graph_list to empty list
for sub_graph ∈ combinations(G, qubit_num) do
    if sub_graph is connected then
        if q_used is empty then
            sub_graph.Set_Partition_Score(G, C, QC)
            sub_graph_list.append(sub_graph)
        end if
        if no qubit in sub_graph is in q_used then
            crosstalk_pairs ← Find_Crosstalk_pairs(sub_graph,
                  crosstalk_props, q_used)
            sub_graph.Set_Partition_Score(G, C, QC, crosstalk_pairs)
            sub_graph_list.append(sub_graph)
        end if
    end if
end for
return sub_graph_list
```

We devise a fidelity score metric for a partition that is the sum of the graph diameter L, average `CNOT` error rate of the links times the number of `CNOTs` of the circuit, and the sum of the readout error rate of each qubit in a partition (shown in (1)). Note that the `CNOT` error rate includes the crosstalk effect if it exists.

$$Score_g = L + Avg_{CNOT} \times \#CNOTs + \sum_{Q_i \in P} R_{Q_i} \tag{1}$$

The graph diameter L is always prioritized in this equation, since it is more than one order of magnitude larger than the other two factors. The partition with the smallest fidelity score is selected. It is supposed to have the best connectivity and the lowest error rate. Moreover, the partition algorithm prioritizes the quantum circuit with a large density because the input circuits are ordered by their densities during the parallelism manager process. The partition algorithm is then called for each circuit in order. However, GSP algorithm is expensive and time costly. For small circuits, the GSP algorithm gives the best choice of partition. It is also useful to use it as a baseline to compare with other partition algorithms. For beyond NISQ, a better approach should be explored to overcome the complexity overhead.

5.1.4 Qubit fidelity degree-based heuristic sub-graph partition algorithm.

In order to reduce the overhead of GSP, we propose a Qubit fidelity degree-based Heuristic Sub-graph Partition algorithm (QHSP). It performs as well as GSP but without the large runtime overhead.

In QHSP, when allocating partitions, we favor qubits with high fidelity. We define the fidelity degree of a qubit based on the `CNOT` and readout fidelities of this qubit as in (2).

$$F\_Degree_{Q_i} = \sum_{Q_j \in N(Q_i)} \lambda \times (1 - E(Q_i, Q_j) + (1 - R_{Q_i}) \tag{2}$$

Q_j are the neighbour qubits connected to Q_i, E is the CNOT error matrix which is constructed by applying the Floyd-Warshall algorithm to the hardware coupling graph with CNOT error rate as edge weights, and R is the readout error rate. λ is a user defined parameter to weight between the CNOT error rate and readout error rate. Such parameter is useful for two reasons: (1) Typically, in a quantum circuit, the number of CNOT operations is different from the number of measurement operations. Hence, the user can decide λ based on the relative number of operations. (2) For some qubits, the readout error rate is one or more orders of magnitude larger than the CNOT error rate. Thus, it is reasonable to add a weight parameter.

The fidelity degree metric reveals two aspects of a qubit. The first one is the connectivity of the qubit. The more neighbours a qubit has, the larger its fidelity degree is. The second one is the reliability of the qubit accounting CNOT and readout error rates. Thus, the metric allows us to select a reliable qubit with good connectivity. Instead of trying all the possible subgraph combinations (as in the GSP algorithm), we propose a QHSP algorithm to build partitions that contain qubits with high fidelity degree while significantly reducing runtime.

To further improve the algorithm, we construct a list of qubits with good connectivity as starting points. We sort all physical qubits by their physical node degree, which is defined as the number of links in a physical qubit. Note that, the physical node degree is different from the fidelity degree. Similarly, we also obtain the largest logical node degree of the logical qubit (qubits used in the quantum circuit) by checking the number of different qubits that are connected to a qubit through CNOT operations. Next, we compare these two metrics.

Suppose the largest physical node degree is less than the largest logical node degree. In that case, it means that we cannot find a suitable physical qubit to map the logical qubit with the largest logical node degree that satisfies all the connections. In this case, we only collect the physical qubits with the largest physical node degree. Otherwise, the physical qubits whose physical node degree is greater than or equal to the largest logical node degree are collected as starting points. By limiting the starting points, this heuristic partition algorithm becomes even faster.

Algorithm 2 QHSP algorithm

Input: Quantum circuit QC , Coupling graph G, Calibration data C, Crosstalk properties crosstalk_props, Used_qubits q_{used}, Starting points starting_points
Output: A list of candidate partitions sub_graph_list

```
circ_qubit_num ← QC.qubit_num
Set sub_graph_list to empty list
for i ∈ starting_points do
    Set sub_graph to empty list
    qubit_num ← 0
    while qubit_num < circ_qubit_num do
        if sub_graph is empty then
            sub_graph.append(i)
            qubit_num ← qubit_num + 1
            continue
        end if
        best_qubit ← find_best_qubit(sub_graph, G, C)
        if best_qubit ≠ None then
            sub_graph.append(best_qubit)
            qubit_num ← qubit_num + 1
            continue
        end if
    end while
    if len(sub_graph) = circ_qubit_num then
        if q_used is empty then
            sub_graph.Set_Partition_Score(G, C, QC)
            sub_graph_list.append(sub_graph)
        end if
        if no qubit in sub_graph is in q_used then
            crosstalk_pairs ← Find_Crosstalk_pairs(sub_graph,
                  crosstalk_props, q_used)
            sub_graph.Set_Partition_Score(G, C, QC, crosstalk_pairs)
            sub_graph_list.append(sub_graph)
        end if
    end if
end for
return sub_graph_list
```

For each qubit in the starting points list, the algorithm explores its neighbours and finds the neighbour qubit with the highest fidelity degree calculated in (2), and merges it into the sub-partition. Then, the qubit inside of the sub-partition with the highest fidelity degree explores its neighbour qubits and merges the best one. The process is repeated until the number of qubits inside of the sub-partition is equal to the number of qubits needed. This sub-partition is considered as a subgraph and is added to the partition candidates.

After obtaining all the partition candidates, we compute the fidelity score for each of them. As we start from a qubit with a high physical node degree and merge to neighbour qubits with a high fidelity degree, the constructed partition is supposed to be well-connected, hence, we do not need to check the connectivity of the partition using the longest shortest path L as in (1), GSP algorithm. We can only compare the error rates. The fidelity score metric is simplified by only calculating the `CNOT` and readout error rates

as in (3) (crosstalk is included if it exists). It is calculated for each partition candidate and the best one is selected.

$$Score_h = Avg_{CNOT} \times \#CNOTs + \sum_{Q_i \in P} R_{Q_i} \tag{3}$$

Figure 7: Example of qubit partition on IBM Q 5 Valencia for a four-qubit circuit using QHSP. Suppose the largest logical node degree of the target circuit is three. (a) The topology and calibration data of IBM Q 5 Valencia. The value inside of the node represents the readout error rate (in%), and the value above the link represents the `CNOT` error rate (in%). (b) Process of constructing a partition candidate using QHSP.

Table 1: The physical node degree and the fidelity degree of each qubit on IBM Q 5 Valencia.

Qubit	Q_0	Q_1	Q_2	Q_3	Q_4
Fidelity degree	1.96	3.93	1.95	2.94	1.97
Physical node degree	1	3	1	2	1

Fig. 7 shows an example of applying QHSP on IBM Q 5 Valencia (ibmq_valencia) for a four-qubit circuit. The calibration data of IBM Q 5 Valencia, including readout error rate and `CNOT` error rate are shown in Fig. 7(a). We set λ to two and the physical node degree and the fidelity degree of qubit calculated by (2) are shown in Table 1. Suppose the largest logical node degree is three. Therefore, Q_1 is selected as the starting point since it is the only physical qubit that has the same physical node degree as the largest logical node degree. It has three neighbour qubits: Q_0, Q_2, and Q_3. Q_3 is merged into the sub-partition because it has the highest fidelity degree among neighbour qubits. The sub-partition becomes $\{Q_1, Q_3\}$. As the fidelity degree of Q_1 is larger than Q_3, the algorithm will again select the left neighbour qubit with the largest fidelity degree of Q_1, which is Q_0. The sub-partition becomes $\{Q_1, Q_3, Q_0\}$. Q_1 is still the qubit with the largest fidelity degree in the current sub-partition, its neighbour qubit – Q_2 is merged. The final sub-partition is $\{Q_1, Q_3, Q_0, Q_2\}$ and it can be considered as a partition candidate. The merging process is shown in Fig. 7(b).

5.1.5 Runtime analysis

Let n be the number of hardware qubits (physical qubits) and k the number of circuit qubits (logical qubits) to be allocated a partition. The GSP algorithm selects all the combinations of k subgraphs from n-qubit hardware and takes $O(C(n,k))$ time, which is $O(n\,choose\,k)$. For each subgraph, it computes its fidelity score including calculating the longest shortest path, which scales at $O(k^3)$. It ends up being equivalent to $O(k^3 min(n^k, n^{n-k}))$. In most

cases, the number of circuit qubits is less than the number of hardware qubits, thus the time complexity becomes $O(k^3 n^k)$. It increases exponentially as the number of circuit qubits augments.

The QHSP algorithm starts by collecting a list of m starting points where $m \leq n$. To get the starting points, we sort the n physical qubits by their physical node degree, which takes $O(nlog(n))$. Then, we iterate over all the gates of the circuit (e.g., circuit has g gates) and sort the k logical qubits according to the logical node degree, which takes $O(g+klog(k))$. Next, for each starting point, it iteratively merges the best neighbour qubit until each sub-partition contains k qubits. To find the best neighbour qubit, the algorithm finds the best qubit in a sub-partition and traverses all its neighbours to select the one with the highest fidelity degree. Finding the best qubit in the sub-partition is $O(p)$ where p is the number of qubits in a sub-partition. The average number of qubits p is $k/2$, so this process takes $O(k)$ time on average. Finding the best neighbour qubit is $O(1)$ because of the nearest-neighbor connectivity of superconducting devices. Overall, the QHSP takes $O(mk^2 + nlog(n) + g + klog(k))$ time, and it can be truncated to $O(mk^2 + nlog(n) + g)$, which is polynomial.

5.2 Post qubit partition

By default the multi-programming mechanism reduces circuit fidelity compared to standalone circuit execution mode. If the fidelity reduction is significant, circuits should be executed independently or the number of simultaneous circuits should be reduced even though the hardware throughput can be decreased as well. Therefore, we consistently check the circuit fidelity difference between independent versus concurrent execution.

We start with the qubit partition process for each circuit independently and obtain the fidelity score of the partition. Next, this qubit partition process is applied to these circuits to compute the fidelity score when executing them simultaneously. The difference between the fidelity scores is denoted ΔS, which is the fidelity metric. If ΔS is less than a specific threshold δ, it means simultaneous circuit execution does not significantly detriment the fidelity score, thus circuits can be executed concurrently, otherwise, independently or reduce the number of simultaneous circuits. The fidelity metric and the parallelism manager help determine the optimal number of simultaneous circuits to be executed.

6 Scheduler

The scheduler includes the mapping algorithm to make circuits executable on real quantum hardware.

6.1 Mapping transition algorithm

Two steps are needed to make circuits hardware-compliant: initial mapping and mapping transition. The initial mapping of each circuit is created while taking into account swap error rate and swap distance, and the initial mapping of the simultaneous mapping transition process is obtained by merging the initial mapping of each circuit according to its partition. We improve the mapping transition algorithm proposed in [28] by modifying the heuristic cost function to better select the inserted gate. We also introduce the `Bridge` gate to the simultaneous mapping transition process for multi-programming.

First, each quantum circuit is transformed into a more convenient format – Directed Acyclic Graph (DAG) circuit, which represents the operation dependencies of the circuit without considering the connectivity constraints. Then, the compiler traverses the DAG

circuit and goes through each quantum gate sequentially. The gate that does not depend on other gates (i.e., all the gates before execution) is allocated to the first layer, denoted F. The compiler checks if the gates on the first layer are hardware-compliant. The hardware-compliant gates can be executed on the hardware directly without modification. They are added to the scheduler, removed from the first layer and marked as executed. If the first layer is not empty, which means some gates are non-executable on hardware, a `SWAP` or `Bridge` gate is needed. We collect all the possible `SWAPs` and `Bridges`, and use the cost function H (see (5)) to find the best candidate. The process is repeated until all the gates are marked as executed.

A `SWAP` gate requires three `CNOTs` and inserting a `SWAP` gate can change the current mapping. Whereas a `Bridge` gate requires four `CNOTs` and inserting a `Bridge` gate does not change the current mapping. It can only be used to execute a `CNOT` when the distance between the control and the target qubits is exactly two. Both gates need three supplementary `CNOTs`. A `SWAP` gate is preferred when it has a positive impact on the following gates, allocated in the extended layer E, i.e., it makes these gates executable or reduces the distance between control and target qubits. Otherwise, a `Bridge` gate is preferred.

A cost function H is introduced to evaluate the cost of inserting a `SWAP` or `Bridge`. We use the following distance matrix (see (4)) as in [28] to quantify the impact of the `SWAP` or `Bridge` gate,

$$D = \alpha_1 \times S + \alpha_2 \times \mathcal{E} \tag{4}$$

where S is the swap distance matrix and $\mathcal{E}$ is the swap error matrix. We set α_1 and α_2 to 0.5 to equally consider the swap distance and swap error rate. In [28], only the impact of a `SWAP` and `Bridge` on other gates (first and extended layer) was considered without considering their impact on the gate itself. As each of them is composed of either three or four `CNOTs`, their impact cannot be ignored. Hence, in our simultaneous mapping transition algorithm, we take self impact into account and create a list of both `SWAP` and `Bridge` candidates, labeled as "tentative gates". The heuristic cost function is as:

$$H = \frac{1}{|F + N_{Tent}|}\Big(\sum_{g \in F} D[\pi(g.q_1)][\pi(g.q_2)] + \sum_{g \in Tent} D[\pi(g.q_1)][\pi(g.q_2)]\Big) + W \times \frac{1}{|E|} \sum_{g \in E} D[\pi(g.q_1)][\pi(g.q_2)] \tag{5}$$

where W is the parameter that weights the impact of the extended layer, N_{Tent} is the number of gates of the tentative gate, $Tent$ represents a `SWAP` or `Bridge` gate, and π represents the mapping. `SWAP` gate has three `CNOTs`, thus N_{Tent} is three and we consider the impact of three `CNOTs` on the first layer. The mapping is the new mapping after inserting a `SWAP`. For `Bridge` gate, N_{Tent} is four and we consider four `CNOTs` on the first layer, and the mapping is the current mapping as `Bridge` gate does not change the current mapping. We weight the impact on the extended layer to prioritize the first layer. This cost function can help the compiler select the best gate to insert between a `SWAP` and `Bridge` gate.

Our simultaneous mapping transition algorithm outperforms HA [28] thanks to the modifications of the cost function while not changing its asymptotic complexity. Let n be the number of hardware qubits, g the `CNOT` gates in the circuit. The simultaneous mapping transition algorithm takes $O(gn^{2.5})$ assuming nearest-neighbor chip connectivity and an extended layer E with at most $O(n)$ `CNOT` gates. The detailed explanation about the complexity can be found in [28].

Algorithm 3 Simultaneous mapping transition algorithm

Input: Circuits $DAGs$, Coupling graph G, Distance matrices Ds, Initial mapping π_i, First layers Fs

Output: Final schedule

$\pi_c \leftarrow \pi_i$
while not all gates are executed **do**
    Set swap_bridge_lists to empty list
    for F_i in Fs **do**
        for gate in F_i **do**
            if gate is hardware-compliant **then**
                schedule.append(gate)
                Remove gate from F_i
            end if
        end for
        if F_i is not empty **then**
            swap_bridge_candidate_list ← FindSwapBridgePairs(F_i, G)
            swap_bridge_lists.append(swap_bridge_candidate_list)
        end if
    end for
    for swap_bridge_candidate_list ∈ swap_bridge_lists **do**
        for g_{tmp} ∈ swap_bridge_candidate_list **do**
            $\pi_{\text{tmp}} \leftarrow$ Map_Update(g_{tmp}, π_c)
            $H_{\text{basic}} \leftarrow 0$
            for gate ∈ F_i **do**
                $H_{\text{basic}} \leftarrow H_{\text{basic}} + D_i(\text{gate}, \pi_{\text{tmp}})$
            end for
            $H_{\text{tentative}} \leftarrow g_{\text{tmp}}.cost(G, D_i, \pi_{\text{tmp}})$
            Update the extended layer E
            $H_{\text{extend}} \leftarrow 0$
            for gate ∈ E **do**
                $H_{\text{extend}} \leftarrow H_{\text{extend}} + D_i(\text{gate}, \pi_{\text{tmp}})$
            end for
            $H \leftarrow \frac{1}{|F+H_{\text{tentative}}|}(H_{\text{basic}} + H_{\text{tentative}}) + \frac{W}{|E|}H_{\text{extend}}$
        end for
        Choose the best gate g_n according to H
        $\pi_c \leftarrow$ Map_Update(g_n, π_c)
    end for
    Update Fs
end while
return schedule

7 Evaluation

In this section, we compare our QuMC method with the state of the art and showcase its different applications.

7.1 Methodology

7.1.1 Metrics

Here are the explanations of the metrics we use to evaluate the algorithms.

1. Probability of a Successful Trial (PST) [38]. This metric is used to represent the circuit output fidelity and is defined by the number of trials that give the expected result divided by the total number of trials. The expected result is obtained by executing the quantum circuit on the simulator. To precisely estimate the PST, we execute each quantum circuit on the quantum hardware for a large number of trials (8192).

2. Number of additional `CNOT` gates. This metric is related to the number of `SWAP` or `Bridge` gates inserted. This metric can show the ability of the algorithm to reduce the number of additional gates.

3. Trial Reduction Factor (TRF). This metric is introduced in [10] to evaluate the improvement of the throughput thanks to the multi-programming mechanism. It is defined as the ratio of the number of trials/shots needed when quantum circuits are executed independently to the number of trials/shots needed when they are executed simultaneously.

7.1.2 Comparison

Several published qubit mapping algorithms [16, 18, 22, 25, 28, 40] and multi-programming mapping algorithms [10, 23] are available. We choose HA [28] as the baseline for independent execution, a qubit mapping algorithm taking hardware topology and calibration data into consideration to achieve high circuit fidelity with a reduced number of additional gates. Due to the different hardware access and code unavailability of the state-of-the-art multi-programming algorithms, we only compare our QuMC with independent executions to show the impact of the multi-programming mechanism. Moreover, our qubit partition algorithms can also be applied to the qubit mapping algorithm for independent executions if running a program on a relatively large quantum device.

To summarize, the following comparisons are performed:

- For independent executions, we compare the partition + improved mapping transition algorithm based on HA (labeled as PHA) versus HA to show the impact of partition on large quantum hardware for a small circuit.

- For simultaneous executions, we compare our QuMC framework, 1) GSP + improved mapping transition (labeled as GSP) and 2) QHSP + improved mapping transition (labeled as QHSP), with independent executions, HA and PHA, to report the fidelity loss due to simultaneous executions of multiple circuits.

A detailed summary of the comparisons for independent and simultaneous executions is shown in Table 2. Note that, PHA allows each quantum circuit to be executed on the best partition selected according to the partition fidelity score metric.

7.1.3 Benchmarks

We evaluate our QuMC framework by executing a list of different-size benchmarks at the same time on two quantum devices, IBM Q 27 Toronto (ibmq_toronto) and IBM Q 65

Table 2: A summary of comparisons for independent and simultaneous executions.

Comparison	Independent		Simultaneous	
Methods	HA	PHA	GSP	QHSP
Partition	N/A	Algorithm. 2	Algorithm. 1	Algorithm. 2
Mapping	[28]	Algorithm. 3		

HA method does not include partition process.

Table 3: Information of benchmarks.

Type	ID	Name	Qubits	Num_g	Num_CNOT	Depth
Small	1	3_17_13	3	36	17	22
Small	2	4mod5-v1_22	5	21	11	12
Small	3	mod5mils_65	5	35	16	21
Small	4	alu-v0_27	5	36	17	21
Small	5	decod24-v2_43	4	52	22	30
Medium	6	qaoa_6	6	49	24	26
Medium	7	qaoa_8	8	80	42	38
Medium	8	qaoa_10	10	102	54	38
Medium	9	qft_6	6	81	39	40
Medium	10	qft_8	8	147	68	56
Medium	11	qft_10	10	233	105	72
Medium	12	ising_5	5	91	40	48
Medium	13	ising_10	10	481	90	70
Large	14	adr4_197	13	3439	1498	1839
Large	15	radd_250	13	3213	1405	1781
Large	16	z4_268	11	3073	1343	1644
Large	17	rd73_252	10	5321	2319	2867
Large	18	cycle10_2_110	12	6050	2648	3386
Large	19	sqn_258	10	10223	4459	5458
Large	20	16QBT_10CYC_TFL_4	16	73	29	10
Large	21	16QBT_15CYC_TFL_3	16	109	44	15
Large	22	16QBT_100CYC_QSE_4	16	1136	320	100
Large	23	16QBT_200CYC_QSE_1	16	2272	640	200

Qubits: number of qubits. Num_g: number of gates. Num_`CNOT`: number of `CNOTs`. Depth: circuit depth.

Manhattan (ibmq_manhattan). The benchmarks are collected from QUEKO circuits [37], application-specific benchmarks, and RevLib [39]. These benchmarks are widely used in the quantum community and their details are shown in Table 3. We execute small quantum circuits with shallow-depth on the selected two quantum devices since only they can obtain reliable results. For medium and large quantum circuits, we compile them on the chips without hardware execution.

7.1.4 Algorithm configurations

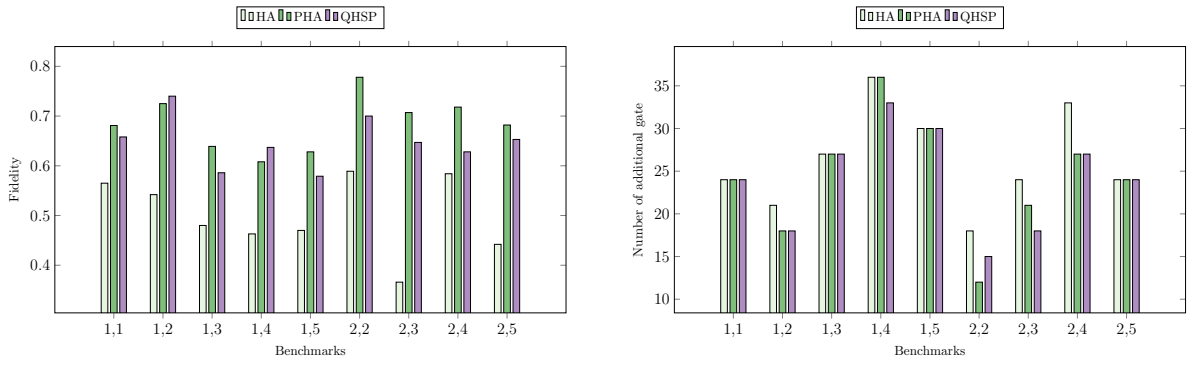
Here, we consider the algorithm configurations of different multi-programming and standalone mapping approaches. We select the best initial mapping out of ten attempts for HA, PHA, GSP, and QHSP. Weight parameter W in the cost function (see (5)) is set to 0.5 and the size of the extended layer is set to 20. Parameters α_1 and α_2 are set to 0.5 respectively to consider equally the swap distance and swap error rate.

For the experiments of simultaneous executions of multiple different-size circuits (Section 7.2), the weight parameter λ of QHSP (see (2)) is set to 2 because of the relatively large number of `CNOT` gates in benchmarks, whereas for the deuteron experiment (Section 7.3), λ is set to 1 because of the small number of `CNOTs` of the parameterized circuit. The threshold δ for post qubit partition is set to 0.1 to ensure the multi-programming reliability. Due to the expensive cost of SRB, we perform SRB only on IBM Q 27 Toronto and collect the pairs with significant crosstalk effect. Only the collected pairs are characterized and their crosstalk properties are provided to the partition process. The experimental results on IBM Q 65 Manhattan do not consider the crosstalk effect. For each algorithm, we only evaluate the mapping transition process, which means no optimisation methods like gate commutation or cancellation are applied.

The algorithm is implemented in Python and evaluated on a PC with 1 Intel i5-5300U CPU and 8 GB memory. Operating System is Ubuntu 18.04. All the experiments were performed on the IBM quantum information science kit (Qiskit) [13] and the version used is 0.21.0.

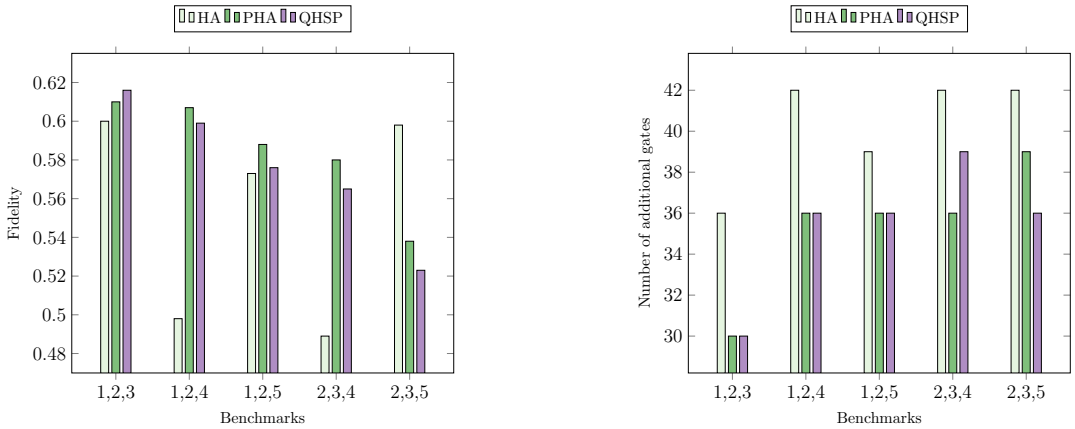
7.2 Application: simultaneous executions of multiple circuits of different sizes

7.2.1 Experimental results



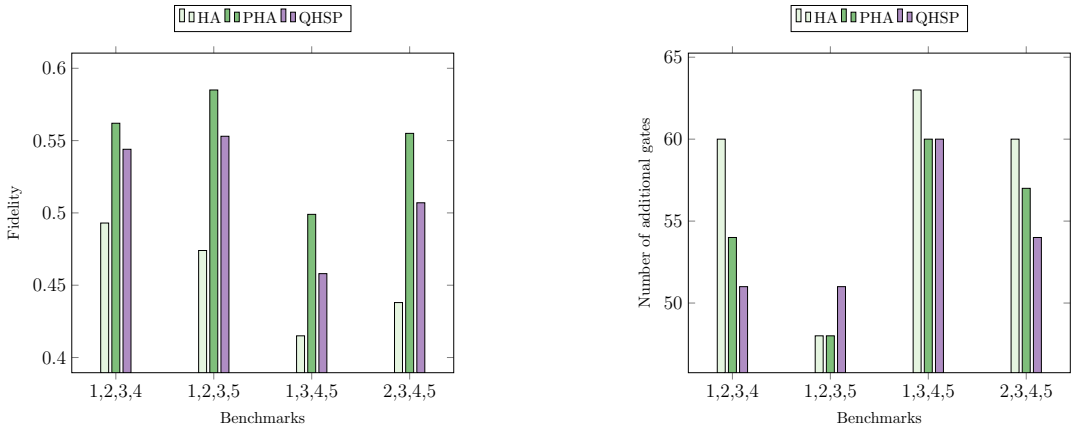
(a) (b)

Figure 8: Comparison of average fidelity and total number of additional gates on IBM Q 27 Toronto when executing two small circuits independently and simultaneously. TRF=2. (a) Fidelity. (b) Number of additional gates.



(a) (b)

Figure 9: Comparison of average fidelity and total number of additional gates on IBM Q 65 Manhattan when executing three small circuits independently and simultaneously. TRF=3. (a) Fidelity. (b) Number of additional gates.

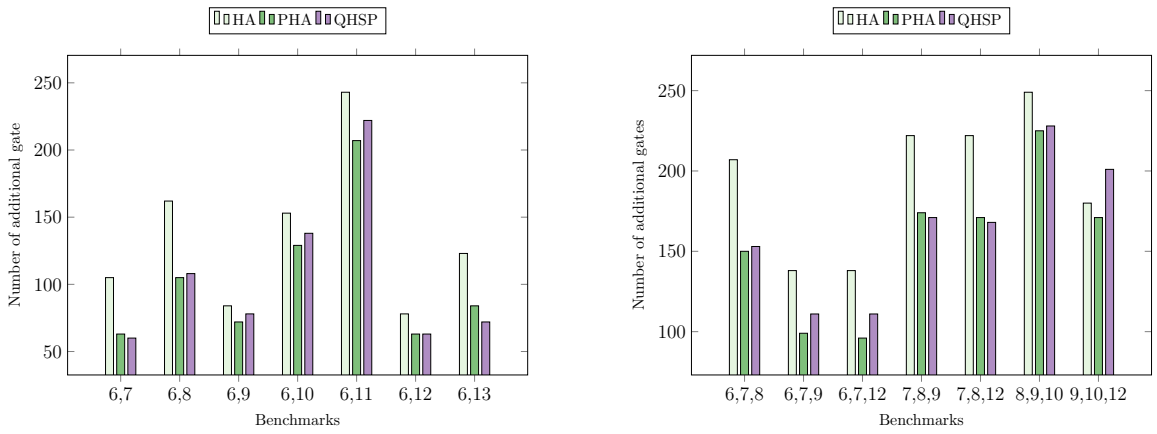


(a) (b)

Figure 10: Comparison of average fidelity and total number of additional gates on IBM Q 65 Manhattan when executing four small circuits independently and simultaneously. TRF=4. (a) Fidelity. (b) Number of additional gates.

We first run two small quantum circuits on IBM Q 27 Toronto independently and simultaneously. Results on average output state fidelity and the total number of additional gates are shown in Fig. 8. Note that, all the circuit output fidelities are calculated by PST metric explained in Section 7.1.1.

For independent executions, the fidelity is improved by 46.8% and the number of additional gates is reduced by 8.7% comparing PHA to HA. For simultaneous executions, QHSP and GSP allocate the same partitions except for the first experiment – (ID1, ID1). In this experiment, GSP improves the fidelity by 6% compared to QHSP. Note that partition results might be different due to the various calibration data and the choice of λ, but the difference of the partition fidelity score between the two algorithms is small. The results show that QHSP is able to allocate nearly optimal partitions while reducing runtime significantly (from exponential to polynomial complexity). Therefore, for the rest experiments, we only evaluate QHSP algorithm. Comparing QHSP (simultaneous executions) versus



(a) (b)

Figure 11: Comparison of total number of additional gates for medium benchmarks when (a) compiling two benchmarks on IBM Q 27 Toronto (TRF=2). (b) compiling three benchmarks on IBM Q 65 Manhattan (TRF=3).

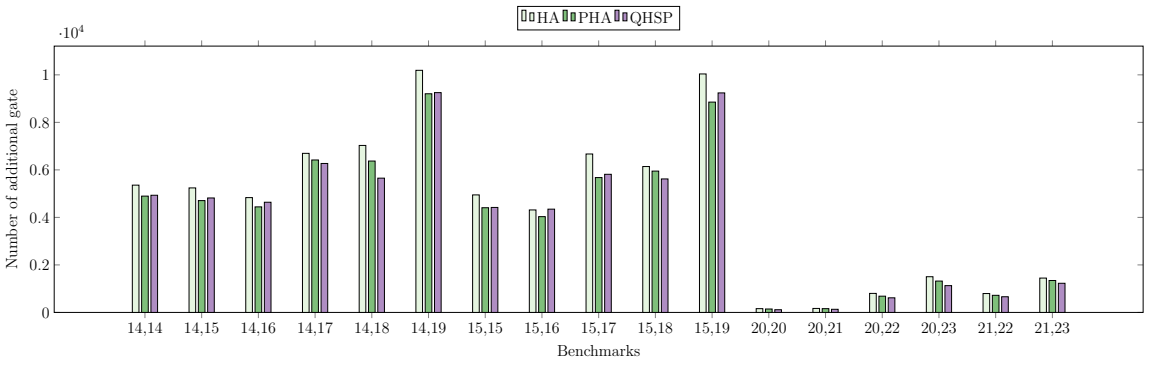


Figure 12: Comparison of total number of additional gates for large benchmarks when compiling two benchmarks on IBM Q 65 Manhattan (TRF=2).

HA (independent executions), the fidelity is even improved by 31.8% and the number of additional gates is reduced by 9.2%. Whereas comparing QHSP with PHA, the fidelity is decreased by 5.4% and the gate number is almost the same, with only 0.3% increase. During the post-partition process, ΔS does not pass the threshold for all the combinations of benchmarks so that TRF is two, which means that the hardware throughput is improved by two times.

Next, we execute on IBM Q 65 Manhattan three and four simultaneous quantum circuits and compare the results with the independent executions. Fig. 9 and Fig. 10 show the comparison of fidelity and the number of additional gates. PHA outperforms HA for independent executions in most of the cases. Comparing QHSP with HA, the fidelity is improved by 5.3% and 13.3% for three and four simultaneous executions, and the inserted gate number is always reduced. Whereas the fidelities decrease by 1.5% and 6.4% respectively for the two cases when comparing QHSP versus PHA, and the additional gate number is always almost the same. The threshold is still not passed for each experiment and TRF becomes three and four.

Then, to evaluate the hardware limitations of executing multiple circuits in parallel, we set the threshold δ to 0.2. All the five small benchmarks are able to be executed simultaneously on IBM Q 65 Manhattan. Partition fidelity difference is 0.18. The average fidelity of simultaneous executions (QHSP) and independent executions (PHA) is 0.493 and 0.54, respectively, corresponding to a fidelity loss of 9.5%.

Finally, to illustrate our QHSP algorithm's performance on medium and large benchmarks, we compile two medium-size circuits on IBM Q 27 Toronto, two medium-size circuits and three large-size circuits on IBM Q 65 Manhattan, simultaneously. We compare the results with HA and PHA for independent compilation. Since these benchmarks are not able to obtain meaningful results due to the noise, we do not execute them on the real hardware and only use the number of additional gates as the comparison metric. The results are shown in Fig. 11 and Fig. 12. The additional gate number is reduced by 23.2%, 15.6%, and 13.2% respectively comparing QHSP with HA. When compared with PHA, the additional gate number is increased by 0.9% and 6.4%, and is reduced by 4.5% respectively. All the program-wise experimental results are listed in Appendix A.

7.2.2 Result analysis

PHA is always better than HA for independent executions for two reasons: (1) The initial mapping of the two algorithms is based on a random process. During the experiment, we perform the initial mapping generation process ten times and select the best one. However, for PHA, we first limit the random process into a reliable and well-connected small partition space rather than the overall hardware space used by HA. Therefore, with only ten trials, PHA finds a better initial mapping. (2) We improve the mapping transition process of PHA, which can make a better selection between `SWAP` and `Bridge` gate. HA is shown to be sufficient for hardware with a small number of qubits, for example a 5-qubit quantum chip. If we want to map a circuit on large hardware, it is better to first limit the search space into a reliable small partition and then find the initial mapping. This qubit partition approach can be applied to general qubit mapping problems for search space limitation when large hardware is selected to map.

Comparing simultaneous process QHSP to independent process HA, QHSP is able to outperform HA with higher fidelity and a reduced number of additional gates. The improvement is also due to the partition allocation and the enhancement of the mapping transition process as explained before. When comparing QHSP with PHA (where independent circuit is executed on the best partition), QHSP uses almost the same number of

additional gates whereas fidelity is decreased less than 10% if the threshold is set to 0.1. However, the hardware throughput increases by two and four times respectively for the two devices. Note that, it also corresponds to a huge reduction of total runtime of these circuits (waiting time + circuit execution time).

7.3 Application: estimate the ground state energy of deuteron

In order to demonstrate the potential interest to apply the multi-programming mechanism to existing quantum algorithms, we investigate it on VQE algorithm. To do this, we perform the same experiment as [11, 15] on IBM Q 65 Manhattan, estimating the ground state energy of deuteron, which is the nucleus of a deuterium atom, an isotope of hydrogen.

Deuteron can be modeled using a 2-qubit Hamiltonian spanning four Pauli strings: ZI, IZ, XX, and YY [11, 15]. If we use the naive measurement to calculate the state energy, one ansatz corresponds to four different measurements. Pauli operator grouping (labeled as PG) has been proposed to reduce this overhead by utilizing simultaneous measurement [7, 15, 19]. For example, the Pauli strings can be partitioned into two commuting families: $\{ZI, IZ\}$ and $\{XX, YY\}$ using the approach proposed in [15]. It allows one parameterized ansatz to be measured twice instead of four measurements in naive method.

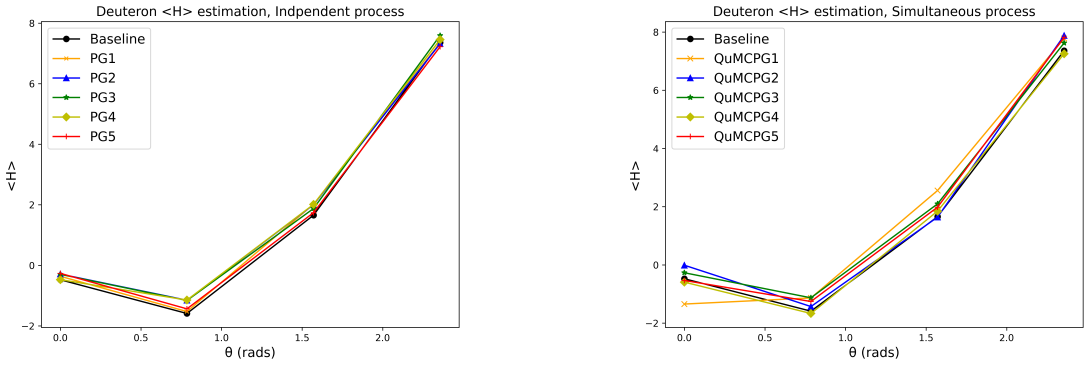
We use a simplified Unitary Coupled Cluster ansatz with a single parameter and three gates, as described in [11, 15]. We apply our QuMC method on the top of the Pauli operator grouping approach (labeled as QuMCPG) to estimate the ground state energy of deuteron and compare the results with PG.

In our QuMC method, the parallelism manager works with the hardware-aware multi-programming compiler to determine the number of circuits for simultaneous execution. Eight circuits are selected in order not to pass the fidelity threshold, which correspond to four parameterized circuits with four different parameters since one parameterized circuit requires two measurement circuits using PG. It is also equivalent to perform four times of optimizations. These circuits can be executed simultaneously using QuMCPG, which reduces the total circuit runtime by eight times compared with PG for independent execution. We perform this experiment five times across days with different calibration data. Note that, if we use the naive measurement, the number of measurement circuits needed will be reduced by a factor of 16. The results of the five experiments using PG (independent process) and QuMCPG (simultaneous process) are shown in Fig. 13. We use simulator to perform the same experiment and set the result as baseline. The sum of the difference between the obtained result (independent or simultaneous process) and baseline (using simulator) is represented by the error rate. All the partition fidelity differences ΔS of the five experiments (on average ΔS=0.06) are less than the threshold δ (set to 0.1). Compared to the baseline, the average error rates are 9% and 13.3% for PG and QuMCPG, respectively. Despite the augmented errors, the hardware throughput is improved by eight times. Note that, the users can tune the threshold δ according to the tolerance of the increase of error rate while using multi-programming. More information about the experimental results can be found in Table 4.

Table 4: The information of the five experiments.

Experiments	n_c [1]	Error rate(%)	Hardware throughput
PG	1	9	0.03
QuMCPG	8	13.3	0.25

[1] the number of simultaneous circuit number.



(a) (b)

Figure 13: The estimation of the ground state energy of deuteron under PG and QuMCPG with four optimisations. (a) PG result (independent process) with eight measurements. (b) QuMCPG result (simultaneous process) with one measurement. TRF=8.

8 Discussion

8.1 Multi-programming mechanism and fidelity loss

The aforementioned experimental results have shown that, the multi-programming mechanism can improve the hardware utilization and reduce the total circuit runtime, but with a cost of slightly losing circuit fidelity. However, the multi-programming mechanism is not always detrimental to circuit fidelity. Especially for large quantum hardware, the partition that has high fidelity is not limited to one region. We choose the largest superconducting quantum hardware, IBM 127 Q Washington (ibmq_washington) to demonstrate it. We pick the two partitions with the highest fidelities according to our partition score metric (3) and execute two of the same circuits on the two partitions simultaneously. The score difference between the two partitions is around 0.01 and they are not adjacent to each other, so that no additional crosstalk. The benchmarks are taken from Table 3 and represented by their IDs. We repeat this experiment five times and the results are shown in Fig. 14. P1 is the partition with the highest score and P2 with the second highest score. From the experimental results, the fidelity of the circuit on P1 cannot always outperform P2 (see benchmarks 2 and 5). It might be due to the following reasons: (1) The calibration data are not 100% precise. Since the partitions have almost the same fidelity scores, the circuits executed on the two partitions should also have similar results. (2) The calibration data are not constant. If the circuits are waiting for a long time in the queue, the calibration data might get updated, so that the partition with the highest fidelity score when submitted the circuit might not be the best one when the circuit is executed.

8.2 Multi-programming on circuits with varying depths

The depths of benchmarks that we have executed on quantum hardware in Section 7.2 are not dramatically different, i.e., the longest circuit depth (decod24-v2_43) is 2.5 times longer than the shortest one (4mod5-v1_22). The program-wise results from Table 5, Table 7, and Table 9 have shown that the fidelity of the circuit with slightly shorter depth is not influenced by the parallel execution.

In this section, we further discuss the impact of multi-programming on circuits with

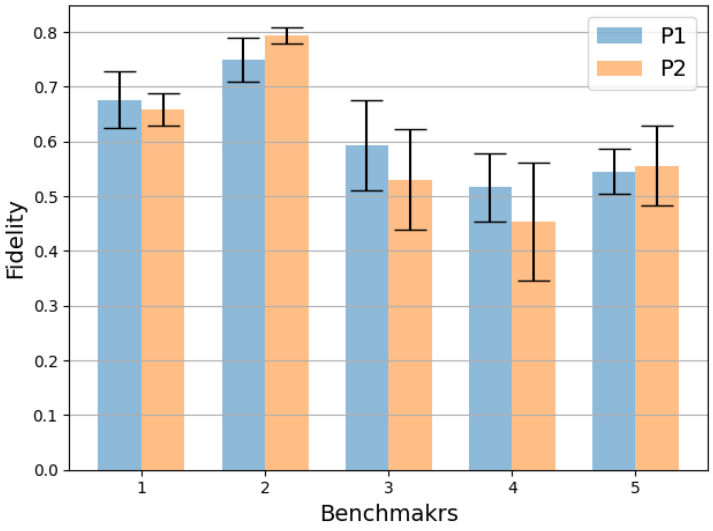


Figure 14: Comparison of fidelity on IBM Q 127 Washington when executing two of the same circuits on partitions P1 and P2 simultaneously.

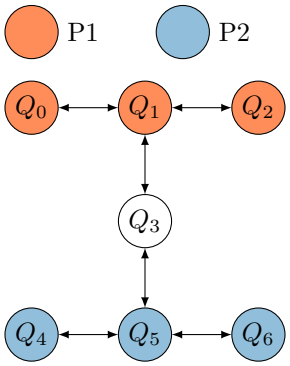


Figure 15: IBM Q 7 Nairobi hardware topology. P1 and P2 are selected to execute circuits with varying depths.

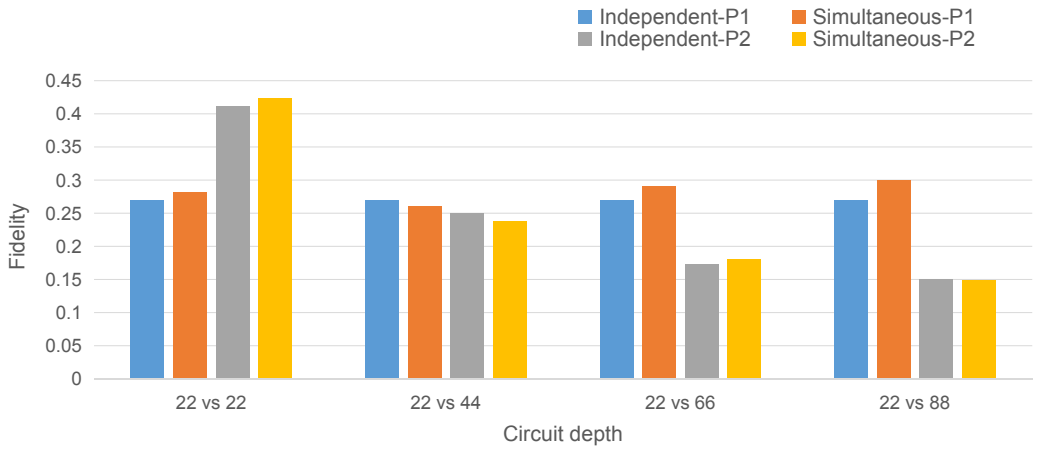


Figure 16: Comparison of fidelity on IBM Q 7 Nairobi when executing two circuits with varying depths on P1 and P2 independently and simultaneously. The blue and red columns represent the fidelities of the original circuit (fixed depth 22). The gray and yellow columns represent the fidelities of the modified circuit (depth from 22 to 88).

comparable or dramatically different depths. We use the largest IBM public chip IBM Q 7 Nairobi (ibm_nairobi) to perform the experiment[1], and its hardware topology is shown in Fig. 15. We choose two partitions P1 and P2, which are more than one hop distance so that no additional crosstalk impact exists. First, we execute a three-qubit small circuit 3_17_13 with circuit depth of 22 (the information of this circuit can be found in Table 3) in P1 individually. Second, we repeat the circuit to increase its depth from one to four times and execute the circuit with varying depths in P2 individually. Finally, we execute the original circuit (depth 22) and the modified circuit (depth from 22 to 88) on P1 and P2 simultaneously. Based on the results shown in Fig. 16, the fidelities of circuits with varying depths are not influenced by parallel executions, since all the circuit operations are scheduled "as late as possible".

9 Conclusion

In this article, we presented QuMC, a multi-programming approach that allows to execute multiple circuits on a quantum chip simultaneously without losing fidelity. We introduced the parallelism manager and fidelity metric to select optimally the number of circuits to be executed at the same time. Moreover, we proposed a hardware-aware multi-programming compiler which contains two qubit partition algorithms taking hardware topology, calibration data, and crosstalk effect into account to allocate reliable partitions to different quantum circuits. We also demonstrated an improved simultaneous mapping transition algorithm which helps to transpile the circuits on quantum hardware with a reduced number of inserted gates.

We first executed a list of circuits of different sizes simultaneously and compared our algorithm with the state of the art. Experimental results showed that our QuMC can even outperform the independent executions using state of the art qubit mapping approach. Then, we investigated our QuMC approach on VQE algorithm to estimate the ground state energy of deuteron, showing the added value of applying our approach to existing

[1]During the preparation of the manuscript, we do not have access to IBM private chips any more due to the end of the contract.

quantum algorithms. The QuMC approach is evaluated on IBM hardware, but it is general enough to be adapted to other quantum hardware.

Based on the experimental result, we found that the main concern with multi-programming mechanism is a trade-off between output fidelity and the hardware throughput. For example, how one can decide which programs to execute simultaneously and how many of them to execute without losing fidelity. Here, we list several guidelines to help the user to utilize our QuMC approach.

- Check the target hardware topology and calibration data. The multi-programming mechanism is more suitable for a relatively large quantum chip compared to the quantum circuit and with low error rate.
- Choose appropriate fidelity threshold for the post qubit partition process. A high threshold can improve the hardware throughput but lead to the reduction of output fidelity. It should be set carefully depending on the size of the benchmark. For benchmarks of small size that we used in experiments, it is reasonable to set the threshold to 0.1.
- The number of circuits that can be executed simultaneously will mainly depend on the fidelity threshold and the calibration data of the hardware.
- The QHSP algorithm is suggested for the partition process due to efficiency and GSP is recommended to evaluate the quality of the partition algorithms. Using both algorithms, one can explore which circuits can be executed simultaneously and how many of them within the given fidelity threshold.

Quantum hardware development with more and more qubits will enable execution of multiple quantum programs simultaneously and possibly a linchpin for quantum algorithms requiring parallel sub-problem executions. The Variational Quantum Algorithm is becoming a leading strategy to demonstrate quantum advantages for practical applications. In such algorithms, the preparation of parameterized quantum state and the measurement of expectation value are realized on shallow circuits [41]. Taking VQE as an example, the Hamiltonian can be decomposed into several Pauli operators and simultaneous measurement by grouping Pauli operators have been proposed in [7, 15, 19] to reduce the overhead of the algorithm. Based on our experiment, we have shown that the overhead of VQE can be further improved by executing several sets of Pauli operators simultaneously using a multi-programming mechanism. For future work, we would like to apply our QuMC to other variational quantum algorithms such as VQLS or VQC to prepare states in parallel and reduce the overhead of these algorithms. Moreover, in our qubit partition algorithms, we take the crosstalk effects into consideration by characterizing them and adding them to the fidelity score of the partition, which is able to avoid the crosstalk error in a high level. There are some other approaches of eliminating the crosstalk error, for example inserting barriers between simultaneous `CNOTs` to avoid crosstalk in a gate-level [26]. However, it has some challenges of trading-off between crosstalk and decoherence. More interesting tricks for crosstalk mitigation need to be targeted for simultaneous executions.

Supplementary material

The source code of the algorithms used in this paper is available on the Github repository `https://github.com/peachnuts/Multiprogramming`.

Acknowledgment

This work is funded by the QuantUM Initiative of the Region Occitanie, University of Montpellier and IBM Montpellier. The authors are very grateful to Adrien Suau for the helpful suggestions and feedback on an early version of this manuscript. We acknowledge use of the IBM Q for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team.

References

[1] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. Analysis of crosstalk in nisq devices and security implications in multi-programming regime. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 25–30, 2020. DOI: https://doi.org/10.1145/3370748.3406570.

[2] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. Experimental characterization, modeling, and analysis of crosstalk in a quantum computer. *IEEE Transactions on Quantum Engineering*, 2020. DOI: https://doi.org/10.1109/TQE.2020.3023338.

[3] Radoslaw C Bialczak, Markus Ansmann, Max Hofheinz, Erik Lucero, Matthew Neeley, AD O'Connell, Daniel Sank, Haohua Wang, James Wenner, Matthias Steffen, et al. Quantum process tomography of a universal entangling gate implemented with josephson phase qubits. *Nature Physics*, 6(6):409–413, 2010. DOI: https://doi.org/10.1038/nphys1639.

[4] Carlos Bravo-Prieto, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick Coles. Variational quantum linear solver: A hybrid algorithm for linear systems. *Bulletin of the American Physical Society*, 65, 2020.

[5] A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996. DOI: https://doi.org/10.1103/PhysRevA.54.1098.

[6] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021. DOI: https://doi.org/10.1038/s42254-021-00348-9.

[7] Ophelia Crawford, Barnaby van Straaten, Daochen Wang, Thomas Parks, Earl Campbell, and Stephen Brierley. Efficient quantum measurement of pauli operators in the presence of finite sampling error. *Quantum*, 5:385, 2021. DOI: https://doi.org/10.22331/q-2021-01-20-385.

[8] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[9] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3):032328, 2019. DOI: https://doi.org/10.1103/PhysRevA.100.032328.

[10] 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. DOI: https://doi.org/10.1145/3352460.3358287.

[11] Eugene F Dumitrescu, Alex J McCaskey, Gaute Hagen, Gustav R Jansen, Titus D Morris, T Papenbrock, Raphael C Pooser, David Jarvis Dean, and Pavel Lougovski. Cloud quantum computing of an atomic nucleus. *Physical review letters*, 120(21): 210501, 2018. DOI: https://doi.org/10.1103/PhysRevLett.120.210501.

[12] Alexander Erhard, Joel J Wallman, Lukas Postler, Michael Meth, Roman Stricker, Esteban A Martinez, Philipp Schindler, Thomas Monz, Joseph Emerson, and Rainer Blatt. Characterizing large-scale quantum computers via cycle benchmarking. *Nature communications*, 10(1):1–7, 2019. DOI: https://doi.org/10.1038/s41467-019-13068-7.

[13] Héctor Abraham et al. Qiskit: An open-source framework for quantum computing. https://qiskit.org/, 2019.

[14] Jay M Gambetta, AD Córcoles, Seth T Merkel, Blake R Johnson, John A Smolin, Jerry M Chow, Colm A Ryan, Chad Rigetti, S Poletto, Thomas A Ohki, et al. Characterization of addressability by simultaneous randomized benchmarking. *Physical review letters*, 109(24):240504, 2012. DOI: https://doi.org/10.1103/PhysRevLett.109.240504.

[15] Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. Optimization of simultaneous measurement for variational quantum eigensolver applications. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 379–390. IEEE, 2020. DOI: https://doi.org/10.1109/QCE49297.2020.00054.

[16] Gian Giacomo Guerreschi and Jongsoo Park. Two-step approach to scheduling quantum circuits. *Quantum Science and Technology*, 3(4):045003, 2018. DOI: https://doi.org/10.1088/2058-9565/aacf0b.

[17] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019. DOI: https://doi.org/10.1038/s41586-019-0980-2.

[18] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70:43–50, 2020. DOI: 10.1016/j.vlsi.2019.10.004.

[19] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017. DOI: https://doi.org/10.1038/nature23879.

[20] Iordanis Kerenidis and Anupam Prakash. Quantum gradient descent for linear systems and least squares. *Physical Review A*, 101(2):022316, 2020. DOI: 10.1103/PhysRevA.101.022316.

[21] Benjamin P Lanyon, James D Whitfield, Geoff G Gillett, Michael E Goggin, Marcelo P Almeida, Ivan Kassal, Jacob D Biamonte, Masoud Mohseni, Ben J Powell, Marco Barbieri, et al. Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106–111, 2010. DOI: https://doi.org/10.1038/nchem.483.

[22] 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. DOI: 10.1145/3297858.3304023.

[23] 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. DOI: https://doi.org/10.1109/HPCA51647.2021.00024.

[24] Pranav Mundada, Gengyan Zhang, Thomas Hazard, and Andrew Houck. Suppression of qubit crosstalk in a tunable coupling superconducting circuit. *Physical Review Applied*, 12(5):054023, 2019. DOI: https://doi.org/10.1103/PhysRevApplied.12.054023.

[25] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, 2019. DOI: 10.1145/3297858.3304075.

[26] 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. DOI: https://doi.org/10.1145/3373376.3378477.

[27] Siyuan Niu and Aida Todri-Sanial. Analyzing crosstalk error in the nisq era. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 428–430, 2021. DOI: https://doi.org/10.1109/ISVLSI51109.2021.00084.

[28] 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. DOI: 10.1109/TQE.2020.3026544.

[29] Yasuhiro Ohkura, Takahiko Satoh, and Rodney Van Meter. Simultaneous quantum circuits execution on current and near-future nisq systems. *arXiv preprint arXiv:2112.07091*, 2021.

[30] Elijah Pelofske, Georg Hahn, and Hristo N Djidjev. Parallel quantum annealing. *Scientific Reports*, 12(1):1–11, 2022. DOI: https://doi.org/10.1038/s41598-022-08394-8.

[31] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014. DOI: https://doi.org/10.1038/ncomms5213 (2014).

[32] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018. ISSN 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[33] Timothy J Proctor, Arnaud Carignan-Dugas, Kenneth Rudinger, Erik Nielsen, Robin Blume-Kohout, and Kevin Young. Direct randomized benchmarking for multiqubit devices. *Physical review letters*, 123(3):030503, 2019. DOI: https://doi.org/10.1103/PhysRevLett.123.030503.

[34] Salonik Resch, Anthony Gutierrez, Joon Suk Huh, Srikant Bharadwaj, Yasuko Eckert, Gabriel Loh, Mark Oskin, and Swamit Tannu. Accelerating variational quantum algorithms using circuit concurrency. *arXiv preprint arXiv:2109.01714*, 2021.

[35] Mohan Sarovar, Timothy Proctor, Kenneth Rudinger, Kevin Young, Erik Nielsen, and Robin Blume-Kohout. Detecting crosstalk errors in quantum information processors. *Quantum*, 4:321, 2020. DOI: https://doi.org/10.22331/q-2020-09-11-321.

[36] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. DOI: 10.1137/S0097539795293172.

[37] Bochen Tan and Jason Cong. Optimality study of existing quantum computing layout synthesis tools. *IEEE Transactions on Computers*, 70(9):1363–1373, 2021. DOI: https://doi.org/10.1109/TC.2020.3009140.

[38] 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. DOI: https://doi.org/10.1145/3297858.3304007.

[39] 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. URL http://www.revlib.org.

[40] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019. DOI: https://doi.org/10.1145/3316781.3317859.

[41] Feng Zhang, Niladri Gomes, Noah F Berthusen, Peter P Orth, Cai-Zhuang Wang, Kai-Ming Ho, and Yong-Xin Yao. Shallow-circuit variational quantum eigensolver based on symmetry-inspired hilbert space partitioning for quantum chemical calculations. *Physical Review Research*, 3(1):013039, 2021. DOI: https://doi.org/10.1103/PhysRevResearch.3.013039.

[42] Peng Zhao, Peng Xu, Dong Lan, Ji Chu, Xinsheng Tan, Haifeng Yu, and Yang Yu. High-contrast z z interaction using superconducting qubits with opposite-sign anharmonicity. *Physical Review Letters*, 125(20):200503, 2020. DOI: https://doi.org/10.1103/PhysRevLett.125.200503.

Table 5: Comparison of fidelity when executing two small circuits simultaneously on IBM Q 27 Toronto.

Benchmarks		Independent						Correlated								Comparison	
ID		HA			PHA			QHSP				GSP				Δ_{PST}%	
ID1	ID2	PST1	PST2	Avg	PST1	PST2	Avg	PST1	PST2	Avg	t	PST1	PST2	Avg	t	HA	PHA
1	1	0.571	0.558	0.565	0.686	0.676	0.681	0.675	0.641	0.658	0.009	0.641	0.682	0.662	0.4	16.5	-3.4
1	2	0.334	0.75	0.542	0.661	0.789	0.725	0.69	0.789	0.74	0.012	0.69	0.789	0.74	7.4	36.5	2.1
1	3	0.547	0.412	0.48	0.687	0.591	0.639	0.619	0.552	0.586	0.007	0.619	0.552	0.586	7.4	22.1	-8.3
1	4	0.476	0.45	0.463	0.574	0.642	0.608	0.626	0.647	0.637	0.016	0.626	0.647	0.637	7.4	37.6	4.8
1	5	0.495	0.445	0.47	0.673	0.582	0.628	0.647	0.511	0.579	0.012	0.647	0.511	0.579	1.6	23.2	-7.8
2	2	0.647	0.53	0.589	0.78	0.775	0.778	0.808	0.591	0.7	0.006	0.808	0.591	0.7	14.4	18.8	-10
2	3	0.428	0.304	0.366	0.787	0.626	0.707	0.764	0.529	0.647	0.013	0.764	0.529	0.647	15	76.8	-8.5
2	4	0.561	0.607	0.584	0.791	0.645	0.718	0.788	0.467	0.628	0.008	0.788	0.467	0.628	14.7	7.5	-12.5
2	5	0.573	0.311	0.442	0.796	0.568	0.682	0.774	0.531	0.653	0.006	0.774	0.531	0.653	8.7	47.7	-4.3

Avg: average of PSTs. **t**: runtime in seconds of the partition process. $\mathbf{\Delta_{PST}}$: comparison of average fidelity.

Table 6: Comparison of number of additional gates when executing two small circuits simultaneously on IBM Q 27 Toronto.

Benchmarks		Independent						Correlated			Comparison	
ID		HA			PHA			QHSP			Δ_g%	
ID1	ID2	g_1	g_2	Sum	g_1	g_2	Sum	g_1	g_2	Sum	HA	PHA
1	1	12	12	24	12	12	24	12	12	24	0	0
1	2	12	9	21	12	6	18	12	6	18	14.3	0
1	3	12	15	27	12	15	27	12	15	27	0	0
1	4	12	24	36	12	24	36	12	21	33	8.3	8.3
1	5	12	18	30	12	18	30	12	18	30	0	0
2	2	6	12	18	6	6	12	6	9	15	16.7	-25
2	3	9	15	24	6	15	21	6	12	18	25	14.3
2	4	9	24	33	6	21	27	6	21	27	18.2	0
2	5	6	18	24	6	18	24	6	18	24	0	0

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.

A Supplementary experimental results

The program-wise experimental results of executing two small circuits simultaneously on IBM Q 27 Toronto (Table 5, Table 6), three small circuits (Table 7, Table 8) and four small circuits (Table 9, Table 10) on IBM Q 65 Manhattan, medium and large circuits on the two devices are listed (Table 11, Table 12, Table 13).

Table 7: Comparison of fidelity when executing three small circuits simultaneously on IBM Q 65 Manhattan.

Benchmarks			Independent								Correlated					Comparison	
ID			HA				PHA				QHSP					Δ_{PST}%	
ID1	ID2	ID3	PST1	PST2	PST3	Avg	PST1	PST2	PST3	Avg	PST1	PST2	PST3	Avg	t	HA	PHA
1	2	3	0.61	0.566	0.624	0.6	0.651	0.624	0.555	0.61	0.609	0.526	0.714	0.616	0.047	2.7	1
1	2	4	0.521	0.683	0.289	0.5	0.637	0.703	0.48	0.607	0.559	0.708	0.531	0.599	0.048	19.8	-1.3
1	2	5	0.627	0.725	0.368	0.573	0.623	0.653	0.487	0.588	0.609	0.592	0.528	0.576	0.047	0.5	-2
2	3	4	0.644	0.434	0.389	0.489	0.631	0.566	0.544	0.58	0.633	0.565	0.498	0.565	0.04	15.5	-2.6
2	3	5	0.689	0.617	0.488	0.598	0.585	0.542	0.486	0.538	0.7	0.528	0.34	0.523	0.04	-12.5	-2.8

Avg: average of PSTs. **t**: runtime in seconds of the partition process. $\mathbf{\Delta_{PST}}$: comparison of average fidelity.

Table 8: Comparison of number of additional gates when executing three small circuits simultaneously on IBM Q 65 Manhattan.

Benchmarks			Independent								Correlated				Comparison	
ID			HA				PHA				QHSP				Δ_g%	
ID1	ID2	ID3	g_1	g_2	g_3	Sum	g_1	g_2	g_3	Sum	g_1	g_2	g_3	Sum	HA	PHA
1	2	3	12	12	12	36	12	6	12	30	12	6	12	30	16.7	0
1	2	4	12	9	21	42	12	6	18	36	12	6	18	36	14.3	0
1	2	5	12	9	18	39	12	6	18	36	12	6	18	36	7.7	0
2	3	4	9	15	18	42	6	12	18	36	6	15	18	39	7.1	-8.3
2	3	5	9	15	18	42	9	12	18	39	6	12	18	36	14.3	7.7

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.

Table 9: Comparison of fidelity when executing four small circuits simultaneously on IBM Q 65 Manhattan.

Benchmarks				Independent										Correlated						Comparison	
ID				HA					PHA					QHSP						Δ_{PST}%	
ID1	ID2	ID3	ID4	PST1	PST2	PST3	PST4	Avg	PST1	PST2	PST3	PST4	Avg	PST1	PST2	PST3	PST4	Avg	t	HA	PHA
1	2	3	4	0.512	0.622	0.486	0.35	0.493	0.588	0.644	0.572	0.443	0.562	0.443	0.747	0.542	0.443	0.544	0.06	10.3	-3.2
1	2	3	5	0.44	0.644	0.608	0.203	0.474	0.648	0.638	0.561	0.491	0.585	0.612	0.645	0.581	0.373	0.553	0.058	16.7	-5.5
1	3	4	5	0.6	0.542	0.228	0.289	0.415	0.592	0.504	0.497	0.404	0.499	0.557	0.53	0.32	0.426	0.458	0.058	10.4	-8.2
2	3	4	5	0.643	0.544	0.287	0.278	0.438	0.699	0.53	0.525	0.465	0.555	0.691	0.477	0.492	0.369	0.507	0.048	15.8	-8.6

Avg: average of PSTs. **t**: runtime in seconds of the partition process. $\mathbf{\Delta_{PST}}$: comparison of average fidelity.

Table 10: Comparison of number of additional gates when executing four small circuits simultaneously on IBM Q 65 Manhattan.

Benchmarks				Independent									Correlated					Comparison	
ID				HA					PHA				QHSP					Δ_g%	
ID1	ID2	ID3	ID4	g_1	g_2	g_3	g_4	Sum	g_1	g_2	g_3	g_4	Sum	g_1	g_2	g_3	g_4	Sum	HA	PHA
1	2	3	4	12	9	15	24	60	12	9	15	18	54	12	6	15	18	51	15	5.6
1	2	3	5	12	9	15	12	48	12	6	12	18	48	12	6	15	18	51	-6.3	-6.3
1	3	4	5	12	15	18	18	63	12	12	18	18	60	12	12	18	18	60	4.8	0
2	3	4	5	6	15	21	18	60	6	15	18	18	57	6	12	18	18	54	10	5.3

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.

Table 11: Comparison of number of additional gates when executing two medium benchmarks on IBM Q 27 Toronto.

Benchmarks		Independent						Correlated			Comparison	
ID		HA			PHA			QHSP			Δ_g%	
ID1	ID2	g_1	g_2	Sum	g_1	g_2	Sum	g_1	g_2	Sum	HA	PHA
6	7	33	72	105	24	39	63	24	36	60	42.9	4.8
6	8	33	129	162	24	81	105	24	84	108	33.3	-2.9
6	9	33	51	84	24	48	72	24	54	78	7.1	-8.3
6	10	33	120	153	24	105	129	24	117	138	9.8	-7
6	11	33	210	243	24	183	207	24	198	222	8.6	-7.2
6	12	33	45	78	24	39	63	24	39	63	19.2	0
6	13	33	90	123	24	60	84	24	48	72	41.5	14.3

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.

Table 12: Comparison of number of additional gates when executing three medium benchmarks on IBM Q 65 Manhattan.

Benchmarks			Independent								Correlated				Comparison	
ID			HA				PHA				QHSP				Δ_g%	
ID1	ID2	ID3	g_1	g_2	g_3	Sum	g_1	g_2	g_3	Sum	g_1	g_2	g_3	Sum	HA	PHA
6	7	8	27	69	111	207	18	39	93	150	18	42	93	153	26.1	-2
6	7	9	27	69	42	138	18	39	42	99	18	42	51	111	19.6	-12.1
6	7	12	27	69	42	138	18	39	39	96	18	45	48	111	19.6	-15.6
7	8	9	69	111	42	222	39	93	42	174	42	78	51	171	23	1.7
7	8	12	69	111	42	222	39	93	39	171	42	78	48	168	24.3	1.8
8	9	10	111	42	96	249	93	42	90	225	90	45	93	228	8.4	-1.3
9	10	12	42	96	42	180	42	90	39	171	42	117	42	201	-11.7	-17.5

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.

Table 13: Comparison of number of additional gates when executing two large benchmarks on IBM Q 65 Manhattan.

Benchmarks		Independent						Correlated			Comparison	
ID		HA			PHA			QHSP			Δ_g%	
ID1	ID2	g_1	g_2	Sum	g_1	g_2	Sum	g_1	g_2	Sum	HA	PHA
14	14	2676	2682	5358	2400	2496	4896	2463	2469	4932	7.9	-0.7
14	15	2766	2475	5241	2382	2325	4707	2529	2289	4818	8.1	-2.4
14	16	2556	2277	4833	2388	2055	4443	2472	2166	4638	4	-4.4
14	17	2670	4026	6696	2502	3915	6417	2481	3789	6270	6.4	2.3
14	18	2685	4344	7029	2430	3942	6372	2403	3249	5652	19.6	11.3
14	19	2733	7458	10191	2445	6759	9204	2457	6795	9252	9.2	-0.5
15	15	2409	2538	4947	2214	2193	4407	2226	2193	4419	10.7	-0.3
15	16	2328	1986	4314	2049	1983	4032	2295	2052	4347	-0.8	-7.8
15	17	2454	4215	6669	2121	3555	5676	2058	3756	5814	12.8	-2.4
15	18	2448	3693	6141	2157	3792	5949	2202	3417	5619	8.5	5.5
15	19	2643	7395	10038	2112	6741	8853	2325	6915	9240	7.9	-4.4
20	20	75	84	159	69	75	144	60	54	114	28.3	20.8
20	21	81	87	168	81	78	159	51	81	132	21.4	16.9
20	22	78	723	801	69	615	684	63	552	615	23.2	10.1
20	23	87	1416	1503	45	1275	1320	78	1050	1128	25	14.5
21	22	102	693	795	72	648	720	105	555	660	17	8.3
21	23	120	1326	1446	78	1266	1344	75	1152	1227	15.1	8.7

g: number of additional gates. **Sum**: sum of number of additional gates. $\mathbf{\Delta_g}$: comparison of sum of number of additional gates.