Design of a Universal Logic Block for Fault-Tolerant Realization of any Logic Operation in Trapped-Ion Quantum Circuits

H. Goudarzi · **M. J. Dousti** ·
A. Shafaei Bejestan · **M. Pedram**

Abstract This paper presents a physical mapping tool for quantum circuits, which generates the optimal Universal Logic Block (ULB) that can, on average, perform any logical fault-tolerant (FT) quantum operations with the minimum latency. The operation scheduling, placement, and qubit routing problems tackled by the quantum physical mapper are highly dependent on one another. More precisely, the scheduling solution affects the quality of the achievable placement solution due to resource pressures that may be created as a result of operation scheduling whereas the operation placement and qubit routing solutions influence the scheduling solution due to resulting distances between predecessor and current operations, which in turn determines routing latencies. The proposed flow for the quantum physical mapper captures these dependencies by applying (i) a loose scheduling step, which transforms an initial quantum data flow graph into one that explicitly captures the no-cloning theorem of the quantum computing and then performs instruction scheduling based on a modified force-directed scheduling approach to minimize the resource contention and quantum circuit latency, (ii) a placement step, which uses timing-driven instruction placement to minimize the approximate routing latencies while making iterative calls to the aforesaid force-directed scheduler to correct scheduling levels of quantum operations as needed, and (iii) a routing step that finds dynamic values of routing latencies for the qubits. In addition to the quantum physical mapper, an approach is presented to determine the single best ULB size for a target quantum circuit by examining the latency of different FT quantum operations mapped onto different ULB sizes and using information about the occurrence frequency of operations on critical paths of the target quantum algorithm to weigh these latencies. Experimental results show an average latency reduction of about 40% compared to previous work.

Keywords Quantum computer aided design · Quantum physical mapping · Quantum instruction placement · Quantum Universal Logic Block

Hadi Goudarzi
University of Southern California, Department of Electrical Engineering
Tel.: +1-213-7409481
Fax: +1-213-7409803
E-mail: hgoudarz@usc.edu

1 Introduction

It has been speculated that quantum computers, which harness physical phenomenon unique to quantum mechanics (especially quantum interference and superposition) to realize a fundamentally new mode of information processing, can overcome the limitation of classical computers in efficiently solving hard computational problems. In particular, polynomial-time algorithms have been offered to solve several NP-intermediate problems by using quantum computers [1].

A key challenge in building large-scale quantum computers is the environmental noise - more precisely, loss of quantum information due to unwanted interactions between the quantum system and its environment. This phenomenon is called *quantum decoherence*. It has been shown that arbitrarily accurate quantum computation is possible, provided that the error per operation is below a *threshold* value [2] and [3]. In other words, quantum error correction works if the level of imperfections and noise in quantum gate operations and measurements are below a certain *error threshold* and if corrections can be applied repeatedly [4]. Based on the error threshold for each quantum circuit fabric technology, an appropriate encoding technique (such as m-level concatenated Steane code [5]) is selected to achieve fault tolerant realization of the quantum circuit.

A typical quantum circuit fabric consists of a two-dimensional array of identical primitive structures (called *primitive cells* in this paper), each structure containing some sites for generating/initializing qubits, measuring them, performing operations on one or two qubits, and channels for moving qubits or swapping their information. See fig. 3 for an example of such an array of primitive cells. Unfortunately, dealing directly with this primitive cell array is very cumbersome and unwieldy. So in practice another 2-D array of super-templates (which we call *tiles*) is built. Each tile comprises a number of primitive cells. Instead of mapping a quantum circuit to the quantum fabric, the quantum circuit is mapped to this tiled architecture (see below).

A quantum logic synthesis tool (surveyed in reference [6]) generates a quantum circuit. Every qubit in the output is called a *logical qubit*, which is subsequently encoded into several *physical qubits* to detect and correct potential errors. To prevent the propagation of errors in the quantum circuit, the (reversible) *logic gates* in the synthesized circuit (which are typically, NOT, CNOT and Toffoli gates) must be converted into FT *quantum operations*. A possible universal (but redundant) set of FT quantum operations includes CNOT, H (Hadamard), T (π/4 rotation), S (phase), X, Y and Z gates. Implementation of these FT quantum operations depend on the chosen error correction method. Note that the set {CNOT, H, T} constitutes a universal basis for quantum circuit realization–the other instructions are included to allow logic simplification during the process of converting the logic synthesis output to the FT quantum operation realization. A given quantum circuit fabric is natively capable of performing a universal set of one and two-qubit instructions (also called *native quantum instructions*). This set differs among various quantum technologies (e.g., ion-trap vs. superconducting qubit vs. quantum dots). Each FT quantum operation can be implemented by using a composition of these native quantum instructions. The transformation from logical gates (which are the result of the quantum logic synthesis) to the FT quantum operations and from the FT quantum operations to the native quantum instructions are called *FT quantum synthesis* and *native quantum synthesis*, respectively. FT and native quantum syntheses are outside of the scope of the present paper.

Each of the FT quantum operations performs a desired function on one or two logical qubits as the input producing one or two logical qubits as the output; each of the input qubits is encoded with some number of physical qubits. The output qubits will also be encoded. Moreover, each of these FT quantum operations requires subsequent *syndrome extraction* circuitry to detect and correct any errors (up to a certain number) that may have been introduced by the quantum operation itself.

Based on the adopted encoding scheme, implementation of each of the FT quantum operations may require hundreds to tens of thousands of native quantum instructions. Different works in the literature e.g., [7] and [8] have suggested using the *tiled quantum architecture* (TQA), comprised of a regular two-dimensional array of *Universal Logic Blocks* (ULBs) to avoid dealing with this complexity. Notice that each ULB in the TQA is capable of performing *any* FT quantum operation. ULBs are separated by routing channels, which are used to move logical qubits (or information about these qubits) from some source ULBs to a target ULB in the TQA. A pictorial representation of the TQA is shown in fig. 1. The quantum structures placed at the junctions of routing channels may be thought of as *quantum crossbars* (possibly with some *qubit purification* [9] capability). Routing channels and quantum crossbars are also built from primitive cells.

A ULB is analogous to a *Configurable Logic Block* (CLB) in a Field Programmable Logic Array (FPGA) device, in that it can implement any of a set of desired functions. Moreover, the same ULB (as identified by its unique row and column indices in the ULB array) can be configured to perform different FT quantum operations at different times as needed. This is analogous to an *on-the-fly-reconfigurable CLB*.

Trapped-ion technology is a very promising technology for implementing quantum circuits [10]. Hence, it is selected as the underlying technology for our quantum physical mapping tool. See section 3 for more details of this
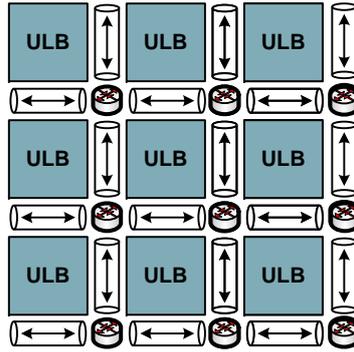
Fig. 1: Tile-based architecture

technology. A trapped-ion quantum circuit fabric comprises of small quantum cells that are organized in a two-dimensional grid, similar to the TQA. Although each primitive cell has the complete set of capabilities provided by the trapped-ion technology, it is not large enough to realize a complete FT operation with an acceptable latency. That is why we resort to the ULB abstraction. Notice that each ULB itself is simply a fixed-size 2-D array of primitive cells, which can realize any FT operation with an acceptable latency.

After appropriate high-level transformations, a quantum algorithm may be represented by a *quantum operation dependency graph* (QODG), where nodes represent FT quantum operations and edges capture data dependencies. Based on the target quantum fabric and error threshold, a particular quantum coding scheme is selected, and subsequently, a high-level tool maps the QODG into a TQA, where each ULB (tile) in this architecture can implement any desired one or two-input quantum operation in a fault-tolerant way. The latency of the quantum algorithm mapped to the TQA can be calculated as the length of the longest path in the QODG, where the length of an input-output path in the QODG is the summation of operation latencies plus routing latencies. The operation latencies are dependent on how precisely the required operation is realized within a ULB. This realization comprises static provisioning (setting the size of the ULB in terms of an array of primitive structures in the target quantum fabric) and mapping of the target operation into the ULB. On the other hand, the routing latencies are dependent on how long it takes for the logical qubits to move from source ULBs to a target ULB to perform the FT quantum operation. The routing latency, which depends on the actual move distances and the latency of moving a qubit through some link in the TQA, is highly sensitive to ULB size, operation placement, and qubit routing on the TQA.

The focus of this paper is on finding the optimal ULB size for the TQA in the trapped-ion technology. The most important part of ULB design is to minimize the operation latency for a fixed ULB size by optimally mapping the quantum instructions to the underlying quantum fabric. This action is called *quantum physical mapping*. The quantum physical mapping procedure is decomposed into three sub-procedures: (a) quantum instruction scheduling, (b) quantum instruction placement, and (c) dynamic qubit routing. Next, based on operation usage frequencies in the target quantum algorithm and the average movement of logical qubits per operation in the TQA, the optimal size of the ULB is determined.

Each quantum fault-tolerant operation and its required syndrome extraction circuit can be modeled by a *quantum instruction dependency graph* (or QIDG). A QIDG for an FT quantum operation is what a QODG is to a quantum algorithm. As an example, a CNOT operation between two physical qubits may be implemented by using more than one native quantum instructions in the trapped-ion technology. Moreover, this number of native quantum instructions is multiplied by a factor of seven if 7-qubit Steane code [[7,1,3]] is used to implement the CNOT in a fault-tolerant manner. Adding syndrome extraction circuit for logical qubits increases the number of native quantum instructions even more. These numbers are far larger in the case of more complex (i.e., non-transversal) FT quantum operations such as T operation in the [[7,1,3]] Steane code, or more complex quantum encoding schemes such as the *concatenated* 7-qubit Steane code.

Different solutions for quantum physical mapping are introduced in the literature – see for example, [7, 8, 11–16]. In this work, an iterative approach is proposed to schedule and place the instructions on quantum fabric to capture the relation between scheduling level of the instructions, instruction placement solution and qubit routing latencies. The proposed algorithms in each step are based on the state-of-the-art scheduling and placement solutions adding the unique characteristics of the quantum instructions scheduling and placement. To capture the quantum no-cloning theorem that forbids fan-out in quantum circuits, a pre-processing step is implemented on the QIDG to resolve the multiple read dependencies between instructions by adding auxiliary edges between them considering

the criticality factor of each instruction. A force-directed instruction scheduling approach is used as the scheduling algorithm to minimize the contention, i.e. the number of concurrent quantum instructions, and hence reduces the routing latency due to congestion among qubits. Quantum instructions can only be executed at specific locations on the quantum circuit fabric called *interaction wells*. The placement of the instructions in these interaction wells affects their start time due to the qubit routing latencies. Hence, we present a placement approach based on a state-of-the-art placement engine with adaptive re-levelization of the operations. More precisely, the placement algorithm calls the quantum instruction scheduler several times to update the scheduling levels of instructions based on the current quantum placement solution. These algorithms are implemented as part of a quantum CAD tool called *quantum ULB factory designer* (QUFD).

To evaluate the performance of the proposed quantum instruction scheduling and placement algorithms, a quantum fabric emulator including a greedy router is developed. The result of the proposed algorithms with respect to previous work is evaluated. On average, 40% improvement with respect to the state-of-the-art quantum mapper is observed. Moreover, considering Toffoli profiling information, the best aspect ratio for a ULB to achieve the best latency is calculated and explained in the experimental section.

The remainder of this paper is organized as follows. Reviews of the most relevant prior work are presented in section 2. Background on the quantum circuit representation and trapped-ion fabric is presented in section 3. The proposed flow for ULB design is presented in section 4. Quantum physical mapping tool is discussed in section 5. Quantum instruction scheduling and placement problem formulation and proposed algorithms are presented in sections 6 and 7. Experimental results are presented in section 8 and the paper is concluded in section 9.

2 Prior Work

QUFD is not the first tool of its kind, but it dramatically improves the performance of existing tools. The rest of this section gives a survey on previous quantum physical design tools and summarizes the contributions and unique features of QUFD compared to the prior art.

Balensiefer et al. [11] developed QUALE, a tool for designing microarchitectures for trapped-ion quantum computers. This tool employs as late as possible scheduling on QODG, greedy qubit placement, and Pathfinder [17] based routing. The key disadvantage of the proposed placement in [11] is that it is independent of the structure of the given QODG. Hence, two qubits that have a lot of interactions may be placed far from each other. This increases the routing cost of bringing the two qubits together to perform the required gate level operations. Moreover, it increases the congestion in the routing channels.

QPOS [12] uses a similar flow as QUALE, but distinguishes between the source and destination operands of a two-qubit instruction during the routing step. This tool employs a combination of list scheduling and as soon as possible scheduling methods for quantum instruction scheduling and a priority-based routing algorithm for qubit routing. Reference [13] tweaks QPOS by creating a priority queue for instructions based on the total delay of their dependent instructions. Reference [8] develops a quantum mapping tool and compares different architectures for quantum fabrics. The main focus of [8] is to improve the quality of high-level mapper. However, it lacks the aggressive optimization aspect of tile (ULB) design. We believe that this optimization dramatically affects the total latency of the final quantum circuit.

Reference [14] discusses the notion of the classical and quantum placement including differences and similarities. The authors propose heuristics to place the quantum instructions and route the qubits by applying the SWAP instruction to minimize the total latency of the quantum circuit. The proposed heuristics in [14] consider fixed instruction scheduling solution and does not consider the global instruction placement solution. Fixed instruction scheduling means the order of instructions are determined before determining the placement and routing solutions which limits the effect of scheduling algorithm solution on the circuit's latency. Moreover, considering the global placement solution reduces the possibility of being trapped in a local optimum. Reference [15] proposes a physical synthesis flow for ancillary qubit selection to reduce the total latency of the quantum physical mapping. This work is focused on changing the design of the fabric to decrease the circuit latency (similar to [13]).

A mapping tool for trapped-ion technology, called QSPR, is proposed in [16]. This tool supports ion multiplexing in the trapped-ion technology; it offers a more optimized global placement of the qubits on the quantum circuit fabric by using an iterative placement approach based on forward and backward computations in a QIDG; and it improves the routing solution by simultaneously moving the source and destination qubits toward a designated site. It provides about 41% improvement over QUALE. We consider this mapping tool the-state-of-the art technique and compare our results with it. The proposed greedy method for instruction placement in this approach cannot decrease the latency in large quantum circuits because the impact of initial optimized qubit placement fades after
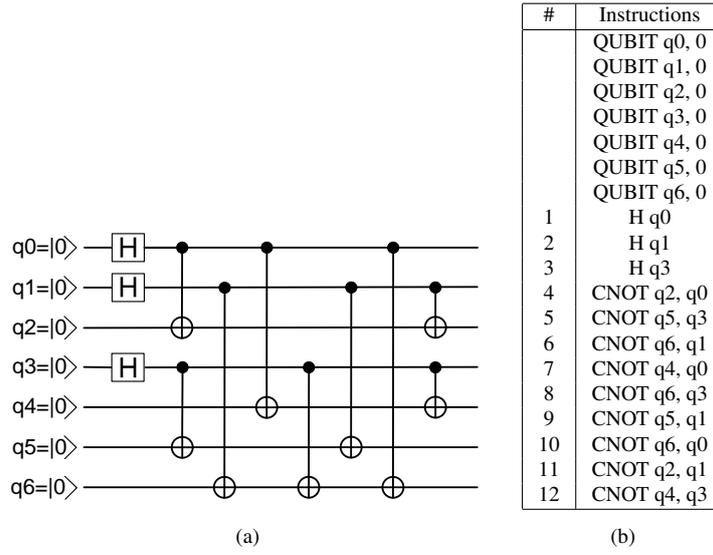
#	Instructions
	QUBIT q0, 0
	QUBIT q1, 0
	QUBIT q2, 0
	QUBIT q3, 0
	QUBIT q4, 0
	QUBIT q5, 0
	QUBIT q6, 0
1	H q0
2	H q1
3	H q3
4	CNOT q2, q0
5	CNOT q5, q3
6	CNOT q6, q1
7	CNOT q4, q0
8	CNOT q6, q3
9	CNOT q5, q1
10	CNOT q6, q0
11	CNOT q2, q1
12	CNOT q4, q3

(a)   (b)

Fig. 2: (a) Circuit for preparing the logical 0 using Steane Code [[7,1,3]], (b) QASM program description of the logical zero preparation circuit

running of few instructions; in other words, qubit placement can only enhance the routing delay for some of the first instructions and has no effect on later instructions.

The proposed quantum physical mapper and ULB designer in our work aim to automate ULB design in quantum mapping tools to decrease the total latency of the quantum circuit realization as much as possible. The proposed quantum physical mapper considers the effect of the routing time on quantum instruction scheduling and placement and presents a cross-layer optimization between these solutions to minimize the latency. In addition, the proposed offline method to optimize the ULB size considering the logical qubit routing times and operation repetition frequency on the critical path of the quantum circuit is not discussed in the prior art.

3 Quantum circuit representation and trapped-ion technology

3.1 Quantum circuit and its representation

Quantum circuits can be composed in Quantum Assembly (QASM) language [18], which provides a compact way of representing quantum circuits. QASM is a low-level quantum language that does not support complex control and data structures like arrays, loops, conditionals, etc.

As an example of a quantum circuit, fig. 2a shows a circuit for preparing logical zero using Steane Code [[7,1,3]] (the output is shown as $|\overline{0}\rangle$). Note that in this circuit, qubits are initialized to $|0\rangle$. Each qubit participates in several one- and two-qubit operations. Seven physical qubits at the output of the circuit represent a bundled logical qubit. Fig. 2b shows the QASM description of this encoding circuit. The first part of the QASM file describes the complete set of qubits involved in the circuit and their initial state, e.g. $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$. The second part of the QASM file describes the quantum operations based on their execution order. Operations having no dependency, e.g. H q0 and H q1, can be added to the file with any ordering. In each line, the operation type, e.g. CNOT, is followed by the qubit(s) involved in the operation. If two qubits are involved in the operation, the first mentioned qubit determines the *control qubit* and the second one determines the *target qubit*. The QIDG for the circuit is built from this QASM description after applying quantum fabric synthesis.

3.2 Trapped-ion technology

Trapped-ion technology is a promising technology for implementing quantum circuits [10]. Hence, it is selected as the underlying technology for our proposed quantum CAD tool. However, our tool can in principle handle other technologies such as the superconducting and solid state quantum fabrics. A recent specification of the trapped-ion technology emanates from a *government furnished information* (GFI) document released by the Sandia National
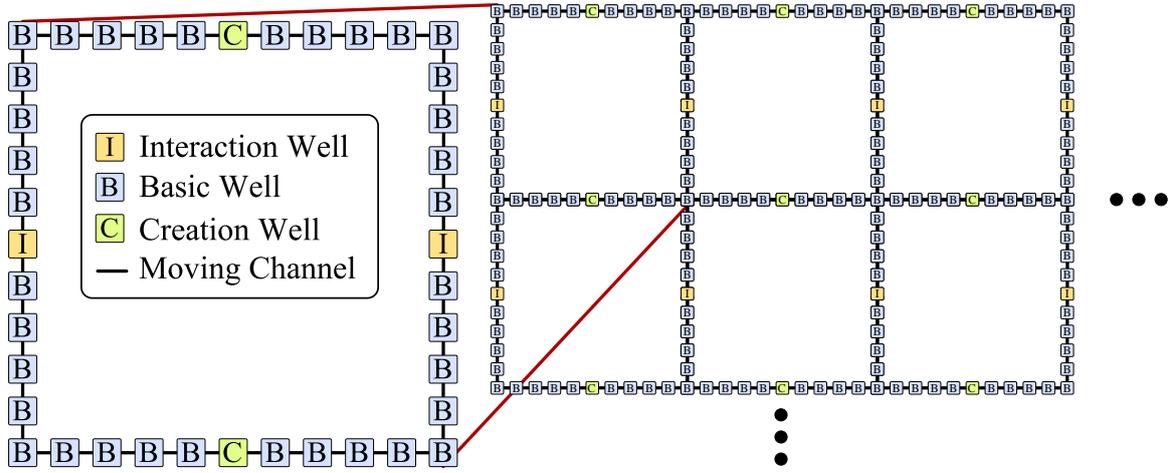
Fig. 3: A model of trapped-ion fabric

Laboratories. A synopsis of relevant details and the layout architecture for this new trapped-ion quantum fabric is included in the remainder of this section.

Qubits: Qubits are realized by ions. The ions can be moved by applying electrical field.

Wells and movement channels: Wells are connected together through movement channels. A well is called *free*, if no qubit occupies it. Two wells are called *adjacent*, if they are connected to each other through a movement channel. There are three types of wells:

1. **Basic Well:** This well only supports qubit movement.
2. **Creation Well:** The qubits are fed to the fabric through creation wells. Besides, it allows qubits to move through it.
3. **Interaction Well:** Quantum operations (also called instructions in this paper) are performed in interaction wells. These wells also allow qubit movement. During a quantum operation, the well is reserved and does not allow another operation or movement. The state-of-the-art trapped-ion technology supports quantum operations involving up to two qubits [19]. In a one-qubit operation, only the target qubit resides in a well. In a two-qubit operation, two qubits inhabit a well, control qubit remains untouched while the target qubit is affected by the operation.[1]

Fig. 3 shows the trapped-ion circuit fabric model. Wells are represented by squares while moving channels are drawn as line segments. White spaces in this figure represent empty locations on the fabric. As can be seen, this fabric consists of 11×11 templates. Each ULB shown in fig. 1 is an n×n grid of these templates. Determining how to optimize the size of ULB is discussed in the section 4.

Qubits travel through the wells and channels to reach interaction wells for quantum operations. There are three constraints for qubit relocations: 1) Movement is allowed only between two adjacent wells, 2) During qubit creation or interaction, the corresponding well is reserved and does not allow another operation or movement, and 3) Moving channels are *half-duplex*.

Well capacity can be defined as the maximum number of qubits that can concurrently reside inside a well. References [19], [20], and [21] discuss about the practical details of how more than one qubit inhabit a channel. Well capacity is a technology-specific parameter. In this paper, the well capacity is set to five according to the GFI document.

---

[1] There are cases in which both qubits are affected. For the sake of simplicity, we still call one qubit as control qubit and the other one as target qubit.
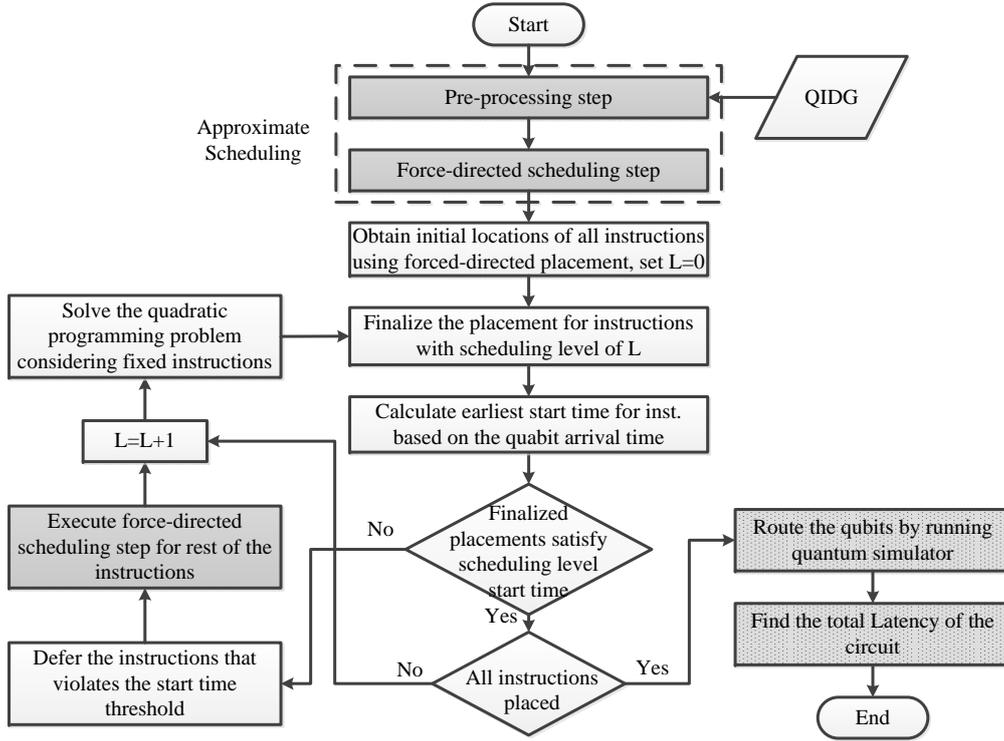
Fig. 4: Flow of the Quantum physical mapper

4 The CAD flow for quantum circuits in the trapped-ion circuit fabric

The input of QUFD is a QASM file describing the FT implementation of the quantum operation (including the required syndrome extraction circuit) in terms of native quantum instructions. The output of QUFD is a list of *low-level control commands*. These commands have three types:

1. Qubit movement command from a source well to a destination well
2. Qubit creation command at a creation well
3. Operation command on one or two qubits at an interaction well

The QUFD CAD flow for mapping the circuit description of a target FT quantum operation to a ULB is shown in fig. 4. The first step in the flow is to make a QIDG from the QASM file. QIDG is built based on the partial ordering of native instructions (which we shall again call quantum 'instructions' for simplicity). The scheduling, placement and routing steps of the quantum physical mapper are highly dependent to one another. More precisely, the scheduling solution affects the quality of the placement solution due to the resource contention, whereas placement solution changes the scheduling solution due to the routing latency. Similarly, the routing solution changes the scheduling solution and affects the placement solution indirectly. The proposed flow for the quantum physical mapper captures these dependencies by applying a scheduling step, a timing-driven instruction placement that minimizes the estimated routing latencies with iterative calls to the force-directed scheduling step to re-calculate scheduling levels for the instructions, and a routing step that determines the exact qubit routes based on the provided placement solution. The goal of the routing step is to minimize the additional routing latency on the critical path(s) of the circuit.

Routing is done in two steps: *static routing* and *dynamic routing*. In the static routing step, the shortest route for each qubit movement is determined. In the dynamic routing step, the actual control commands for routing the qubits are generated. In this step, the router may prohibit a qubit from entering a routing channel, if the channel is full. In this case, the qubit in question should wait till the channel usage drops below its capacity. Details of the different steps in the mapper flow (except for the router which is quite straightforward) are described in the following sections.
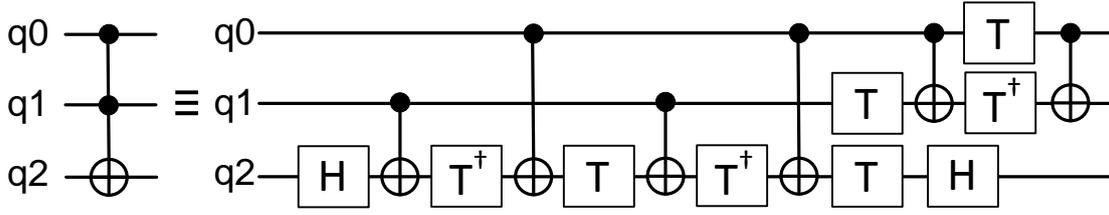
Fig. 5: Fault-tolerant implementation of Toffoli circuit

Table 1: Placement solution and cost for fault-tolerant operations in Toffoli circuit shown in fig. 5

Schedule	Instruction	Placement	Location at beg.			Location at end			Cost
			q0	q1	q2	q0	q1	q2	
1	H q2	3	1	2	3	1	2	3	$L^H(n)$
2	CNOT q1,q2	2	1	2	3	1	2	2	$L^{CNOT}(n) + 2nL^{r,1}$
3	$T^\dagger$ q2	4	1	2	2	1	2	4	$L^{T^\dagger}(n) + nL^{r,1}$
4	CNOT q0,q2	1	1	2	4	1	2	1	$L^{CNOT}(n) + 2nL^{r,1}$
5	T q2	3	1	2	1	1	2	3	$L^T(n) + nL^{r,1}$
6	CNOT q1,q2	2	1	2	3	1	2	2	$L^{CNOT}(n) + 2nL^{r,1}$
7.a	$T^\dagger$ q2	4	1	2	2	1	2	4	$L^{T^\dagger}(n) + nL^{r,1}$
7.b	T q1	2	1	2	2	1	2	4	$L^T(n)$
8	CNOT q0,q2	4	1	2	4	4	2	4	$L^{CNOT}(n) + nL^{r,1}$
9.a	T q2	3	4	2	4	2	2	3	$L^T(n) + nL^{r,1}$
9.b	CNOT q0,q1	4	4	2	4	2	2	3	$L^{CNOT}(n) + nL^{r,1}$
10.a	H q2	3	2	2	3	1	2	3	$L^H(n)$
10.b	T q0	1	2	2	3	1	2	3	$L^T(n) + nL^{r,1}$
10.c	$T^\dagger$ q1	2	2	2	3	1	2	3	$L^{T^\dagger}(n)$
11	CNOT q0,q1	1	1	2	3	1	1	3	$L^{CNOT}(n) + nL^{r,1}$

In addition to minimizing the latency of an FT quantum operation on a given ULB, QUFD minimizes the expected latency of the quantum algorithm by exploring different ULB sizes the determine the best size. The expected latency of the quantum algorithm is defined as a weighted sum of latencies of quantum circuit realizations of the FT quantum operations in the ULB plus the average inter-ULB routing latency. Analyzing the (timing) critical subgraph of the QODG, one can find the occurrence frequency of each FT operation (w^o). Latency of each FT operation as a function of ULB size is found by applying QUFD to their QIDG. These latencies are denoted by $L^o(n)$ in an n×n ULB. Moreover, the average inter-ULB routing distance ($D^{r,avg}$), which is the average qubit routing latency in the QODG critical path normalized to the routing latency between two adjacent ULBs, is found by mapping the QODG to the quantum circuit fabric. The routing latency between adjacent ULBs is denoted by $L^{r,1}$ for 1×1 ULB and is linearly proportional to the ULB size. The best ULB size for the target quantum algorithm is found by the following equation:

$$n^{best} = \underset{n}{\arg\min} \left(nL^{r,1}D^{r,avg} + \sum_{o \in FT \ oper. \ set} w^o L^o(n) \right) \qquad (1)$$

To illustrate the process of setting up equation (1) for a QODG, we consider a Toffoli gate with two control qubits. The Toffoli gate can be implemented using 15 FT operations [22] shown in fig. 5. Now, consider a 2×2 mesh TQA. ULBs in this architecture are numbered from 1 to 4 (1 for top-left , 2 for top-right, 3 for bottom-left and 4 for bottom-right ULB). A QODG mapping tool can map the Toffoli circuit to this architecture. An exemplary (high quality) mapping solution for this circuit is depicted in table 1.

Note that operations 7.b, 9.a, 10.a and 10.c are not part of the critical path of this circuit. As can be seen, the total latency of the critical path of the circuit is $2 \times L^T(n) + 2 \times L^{T^\dagger}(n) + 6 \times L^{CNOT}(n) + 1 \times L^H(n) + 13 \times nL^{r,1}$. In section 8, after finding the latency for each FT operation as a function of the ULB size, the value of the Toffoli gate latency is reported as a function n and the best ULB size is chosen.

Fig. 6 shows the QUFD ULB sizing flow. In this flow, the smallest ULB size is set by the size required for the most complex operation circuit that needs to be mapped to the quantum fabric (this is typically a non-transversal gate operation). As the ULB size increases, the resource contention decreases (i.e., there is less routing resource contention and thus potentially shorter waiting times to access routing channels during the dynamic routing step);
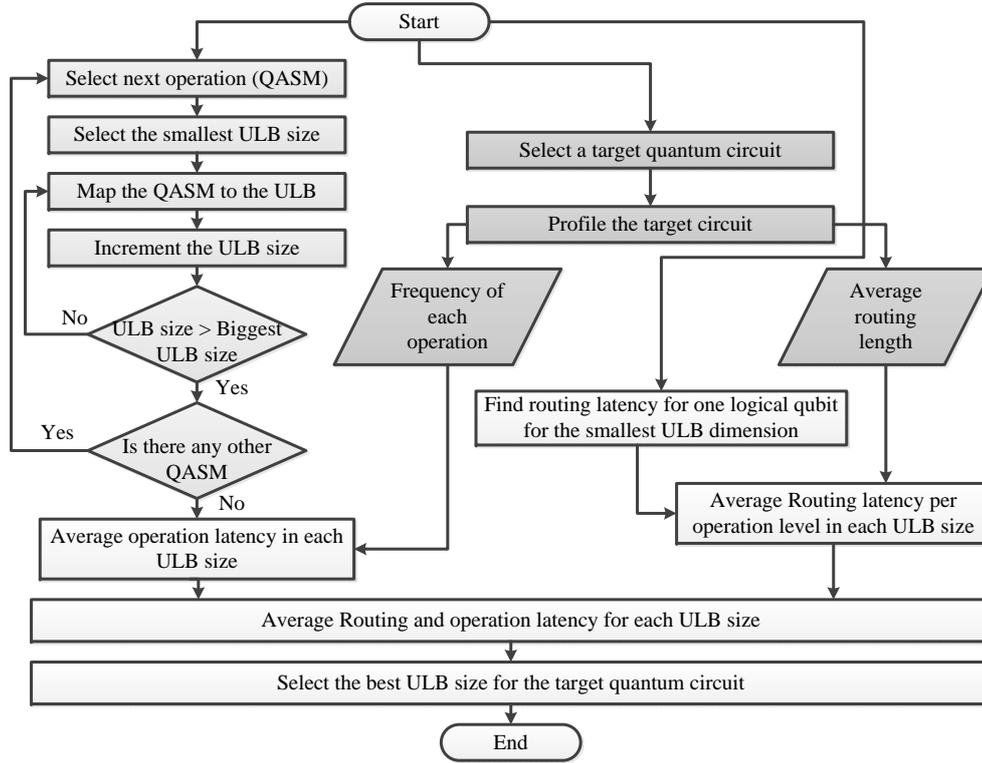
Fig. 6: ULB sizing flow

however, the intra-ULB routes become longer, and hence, routing latencies may actually become higher . From another perspective, increasing the ULB size increases the inter-ULB routing latency. In practice, there is an optimum ULB size that minimizes the aforesaid cost function in 1.

The proposed ULB sizing flow is an offline process to generate the tiles that comprise the TQA and the gate library that is used by the quantum physical mapper. Considering this fact, long run-time for this process does not raise a problem. Moreover, the number of ULB sizes explored to find the optimal n is finite considering the fact that beyond a certain ULB size, increasing the ULB size does not positively affect the mapped circuit latency.

5 Quantum physical mapping problem

The quantum physical mapper converts a quantum circuit to a list of low-level control commands by specifying the exact qubit movements and instruction execution times. Quantum physical mapping problem includes scheduling, placement and routing sub-problems. The scheduling step determines the order of instruction execution, the placement step determines locations of qubits at the beginning and end of each operation as well as the location for each instruction execution, while the routing step determines the qubit location at movement times.

Qubit movement latency is a big part of the quantum circuit latency. Therefore, the routing latency (which is a function of instruction placement and channel congestion) must be included in the decision making procedure for scheduling and placement problems. To account for routing latency in scheduling and placement steps, routing latency estimates based on the physical distance between interaction wells are used. Physical distance is different from Manhattan distance because it is the shortest path between two wells going through movement channels. Hence, it is equal or longer than Manhattan distance.

To explain the concurrent scheduling and placement problem in quantum physical mapper and the similarities and differences between this problem and scheduling and placement problem in the traditional VLSI CAD field, we consider two scenarios. In the first scenario, each instruction is executed in a unique interaction well and in the second scenario, more than one instruction can be executed in one interaction well as long as their execution times do not overlap.
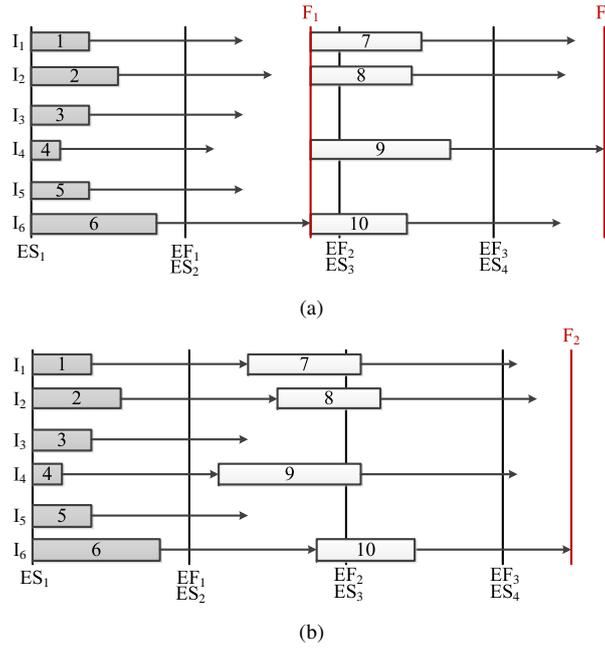
(a)

(b)

Fig. 7: Example of scheduling and placement solutions: (a) slotted time with fixed start time, (b) Slotted time with ASAP start time

First Scenario: Interaction wells in this scenario are similar to gates or transistors in VLSI circuits with the difference that there are shared connectors (channels) between interaction wells in the quantum fabric in place of dedicated connectors (point-to-point wires) in the VLSI circuits. Considering the large channel capacity in the quantum fabric, we can ignore the effect of routing resource contention in this scenario. Under this assumption, the best scheduling policy is the *as soon as possible schedule*. Moreover, the placement problem in this scenario is similar to VLSI timing-driven placement problems. The goal of the optimization is to minimize the total added latency to the critical path of the quantum circuit.

Second Scenario: Sharing interaction wells between instructions complicates the scheduling and placement problems. The scheduling and placement problems can thus be seen as a 3-dimensional placement problem with time as the third dimension. In addition to the constraint of no-overlap between execution times of instructions placed at the same interaction well, placing instructions along the time axis must satisfy the instruction dependency constraints. The resulting circuit latency and area in this scenario are much smaller than those of the first scenario.

To simplify the placement problem, we consider discrete start times for instruction executions. An example of the scheduling and placement solution for this scenario is shown in fig. 7a, where ES_t, EF_t and F_t denote the expected start time, expected finish time, and actual finish time of the instructions in time slot t, respectively. Moreover, I_1 to I_6 indicate the interaction wells on the quantum fabric; numbers in rectangles show the routing times of the operand qubits, and horizontal arrows represent the execution latencies of instructions. Instructions with dark color belong to the first scheduling level whereas instructions with white background belong to the second scheduling level.

As can be seen in fig. 7a, considering synchronized start time for qubit routing in each scheduling level (ES_1 for level 1 and F_1 for level 2) leads to inefficient use of resources and increases the circuit latency. To solve this problem, if instructions are started as soon as their qubits and the host interaction well become ready, the total latency of the circuit is lowered as shown in fig. 7b. This scheme has commonly been used in previous work such as [16]. Unfortunately, this modification decreases the predictability of start times for instruction executions. More precisely, this scheduling contradicts the assumption about separation of scheduling and placement steps because it makes it possible for two instructions that have statically been scheduled in two different time slots to overlap in the actual schedule of the circuit due to changes to routing latencies. Furthermore, placing instructions on the circuit fabric based on the instruction level has an underlying assumption about the stop time of the previous-level instructions. This assumption does not hold if the instructions are scheduled as soon as their qubits and interaction wells are ready. An example of this case is shown in fig. 8a. As can be seen, the second scheduling level instructions that are placed in the interaction wells hosting instructions 2 and 6 in the first scheduling level (I_2 and
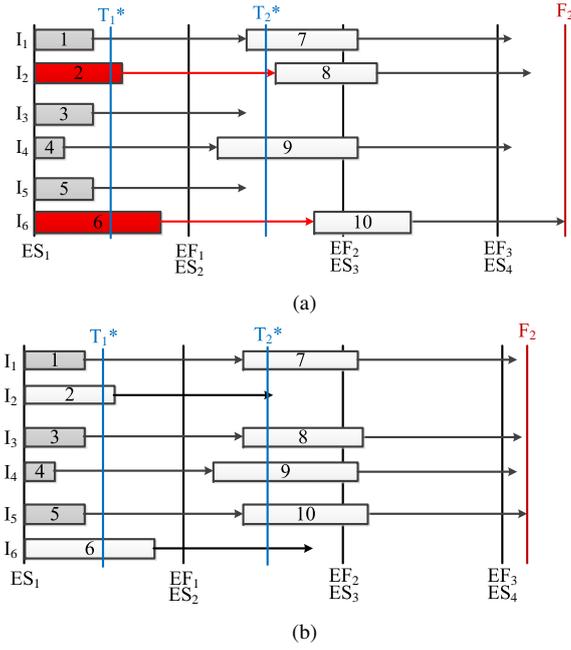
Fig. 8: Example of the proposed method for modifying the scheduling and placement solutions: (a) before modification, (b) after modification

I_6) are deferred. If the placement module had been aware of this situation, instructions 8 and 10 would have been placed on other interaction wells to lower their finish times.

To be able to predict the actual instruction execution times in the scheduling and placement step, our approach calculates the estimated start time of each instruction and if the estimated start time exceeds a start time threshold (shown by T_i^* in fig. 8a and calculated as the middle point of ES_i and EF_i), the instruction will be deferred and the scheduling and placement solutions are modified. An example of this modification is shown fig. 8b. As can be seen, this modification reduces the total latency of these ten instructions. Details of the proposed scheduling and placement solutions are presented in sections 6 and 7, respectively.

6 Instruction scheduling

The first step of the quantum physical mapper is *loose scheduling*. The goal of this step is to assign each instruction to a scheduling level (or time slot) to minimize the total latency of the circuit subject to dependency and resource availability constraints. All instructions assigned to one scheduling level can be executed simultaneously. This step is called loose because it does not consider the routing latencies when scheduling the instructions. To modify the scheduling solution based on placement decisions, a part of this scheduling step is iteratively called during the placement procedure.

The input to the scheduling procedure is a QIDG, which is a *directed acyclic graph*, capturing the dependencies between instructions specified in the QASM file. Each node of this graph is a native quantum instruction, i.e., it can be performed in one step in an interaction well in the trapped-ion quantum circuit fabric. Two instructions are dependent if any of these conditions hold:

1. The target qubit of instruction A is used as the control qubit of instruction B (Read After Write (RAW) dependency)
2. The target qubit of instruction A is used as the target qubit of instruction B (Write After Write (WAW) dependency)
3. The control qubit of instruction A is used as the target qubit of instruction B (Write After Read (WAR) dependency).

Each dependency is shown using a directed edge from the vertex representing instruction A to the vertex representing instruction B. Fig. 9 shows an example of QIDG for the quantum circuit presented in fig. 2a.
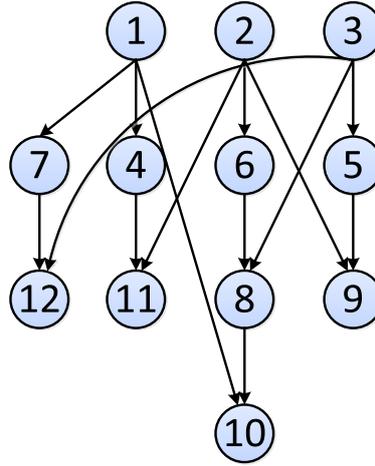
Fig. 9: Example of QIDG for the quantum circuit presented in fig. 2a

Each instruction and each qubit in the QIDG are identified by unique identifiers, represented by index i and q respectively. In the QIDG, nodes with the same parent instruction are called *sibling instructions*. These instructions are two-qubit instructions that share the same control qubit.

Due to the *no-cloning theorem* [23], only one copy of each qubit can exist in the circuit. More precisely, only one instruction can access a qubit at each point in time. So, if one qubit needs to be accessed by sibling instructions, e.g. qubit q0 in instructions 4 and 7 in fig. 9, those instructions cannot be executed at the same time. Based on the assumption that all instructions in one scheduling level can be executed simultaneously, at most one instruction from each sibling instruction set can be placed in one scheduling level. Therefore, an ordering should be imposed among sibling instructions to be able to schedule the instructions without additional restrictions.

To order the sibling instructions, we add *auxiliary edges* between them. Note that adding these auxiliary edges may increase the critical path length. Hence, this process should be done carefully. From another perspective, the number of instructions at each scheduling level is limited by the number of available interaction wells in the ULB.

The minimum latency QIDG scheduling problem (called *quantum instruction scheduling problem* or QISP) with the assumption of the same execution length for every instruction may be formulated as follows:

$$Min\ L$$

subject to:

$$t_i \leq L, \qquad\qquad\qquad \forall i \qquad (2)$$

$$t_i = \sum_l l.x_{il} \qquad\qquad\qquad (3)$$

$$\sum_l x_{il} = 1, \qquad\qquad\qquad x_{il} \in \{0,1\} \qquad (4)$$

$$t_i - t_j \geq 1, \qquad\qquad\qquad \forall j \in P_i \qquad (5)$$

$$x_{il} + \sum_{k \in S_i} x_{kl} \leq 1, \qquad\qquad\qquad \forall l,i \qquad (6)$$

$$\sum_i x_{il} \leq N^{max}, \qquad\qquad\qquad \forall l \qquad (7)$$

where P_i and S_i denote the sets of *parent* and *sibling instructions* for instruction i, respectively; t_i denotes the start time of instruction i; L, which stands for latency, is defined as the time when the last instruction starts its execution; N^{max} denotes the maximum number of concurrent instructions at each time. Parameter x_{il} is the scheduling parameter, which is equal to one if instruction i starts from level l; otherwise it is zero. Equation (3) relates the start time of an instruction to the scheduling parameter (x_{il}). Constraint (4) forces an instruction to have only one starting time. Constraint (5) captures the dependency constraints in the QIDG. Constraint (6) captures the fact that sibling instructions cannot be scheduled at the same time. Constraint (7) enforces the limit on the number of concurrent instructions at each scheduling level.

QISP is an integer linear programming (ILP) problem. In fact, QISP is a *resource-constrained scheduling* problem [24] with one important difference. In QISP, in addition to the limit on the total number of processing

elements in each scheduling level (cf. constraint (7)), the number of processing elements that can be used for sibling instructions is limited to one (cf. constraint (6)). Our solution to QISP decomposes the problem into two steps: (i) A pre-processing step based on the *list scheduling* algorithm [25] is applied to the QIDG to *totally order* the sibling instructions; (ii) An enhanced version of the well-known *force-directed scheduling* algorithm [26] is applied to the modified QIDG so as to minimize a cost function which is a function of the number of scheduling levels and the number of concurrent instructions per scheduling level. Details are provided below.

6.1 QIDG pre-processing step

The goal of the pre-processing step is to impose a total ordering on the sibling instructions to avoid any violations of constraint (6) during the scheduling process. A well-known approach for solving the resource-constrained scheduling problem is the list scheduling algorithm [25], which is reviewed in the next paragraph.

First, *as soon as possible* (ASAP) and *as late as possible* (ALAP) schedules for a given task dependency graph are constructed. The difference between levels of an instruction in the ASAP and ALAP schedules, which is called *mobility*, sets the priority of an instruction in scheduling with respect to the other instructions (that is, less mobile instructions have higher priority to be scheduled first). To schedule the instructions based on this approach, starting from the first level of instructions in the ASAP schedule, if a resource constraint is violated, an instruction with the highest mobility (least priority) will be picked and delayed by one scheduling level. While the resource constraint is violated, this process continues. This process is repeated for the next level and continues until all conflicts are resolved. The motivation behind this greedy heuristic is that the instruction with higher mobility can be delayed without adversely affecting the overall execution latency.

Algorithm 1 provides the pseudo-code for the pre-processing step of the scheduling solution, which is based on list scheduling algorithm. One of the difficulties in QISP is that the length of the critical path that determines the instruction mobilities is not known *a priori* because the sibling instructions have not been ordered yet. Therefore, new ASAP and ALAP schedules are determined in each step based on the modified QIDG (namely, current partial ordering of the sibling instructions). Let SL_i, $ASAP_i$, and $ALAP_i$ denote the scheduling level, ASAP level, and ALAP level of instruction i. ASAP and ALAP levels of each instruction along with the priority values (i.e., $1/(ALAP_i - ASAP_i)$) are calculated in the first iteration. Starting from the set of instructions with $ASAP_i = 0$, an instruction with the highest priority among siblings that have an ASAP level equal to zero is selected. This instruction is added as a parent for its sibling instructions with the same or higher ASAP level. Note that this selection fixes the instruction with the least mobility and moves its sibling instructions to the next levels in contrast to the list scheduling that repeatedly defers the instruction with the highest mobility until the resource constraint is satisfied. In this special problem, these approaches have the same results (with lower complexity for our algorithm), because the number of allowed sibling instructions in each scheduling level is only one. After these modifications, ASAP and ALAP and priority metrics are recalculated. If constraint (6) is no longer violated for the selected set of instructions, the set of instructions with $ASAP_i = 1$ will be selected and the process of adding edges will be repeated. This process is continued for all the ASAP levels or until all required edges are added to the QIDG.

Applying this pre-processing step to the QIDG of fig. 9 results in the QIDG of fig. 10a. Dashed (red) edges in this figure show the added auxiliary edges.

6.2 Quantum instruction scheduling algorithm

The number of quantum instructions being executed at the same time is upper bounded by the number of available interaction wells in the ULB. This bound is captured in constraint (7) of the QISP formulation. Although increasing the number of concurrent instructions in each scheduling level tends to decrease the number of scheduling levels, it may increase the total latency of the circuit as explained next. Recall that the number of concurrent instructions in each scheduling level greatly affects the placement solution, routing delays, and thus, the latency of the quantum circuit. An instruction cannot be executed until its qubit operands arrive at the host interaction well. The routing time of a qubit is defined as the difference between the arrival time of the qubit in the case of one-qubit instructions (or that of the last arriving qubit in the case of two-qubit instructions) and the time when the instruction was scheduled to be performed. The routing time is a function of the specific route selected for the operand qubit(s) and the wait (stall) time of the qubit during the routing due to the congestion (shared resource contention). Increasing the number of concurrent instructions forces some instructions to be placed in interaction wells that are far from the interaction wells that are hosting their parent instructions. Moreover, increasing the number of concurrent instructions results in an increase in the number of qubits that need to be simultaneously routed. In addition to the

---

Algorithm 1 Preprocessing of QIDG

 1: Compute $ASAP_i$, $ALAP_i$ and $m_i = 1/(ALAP_i - ASAP_i)$
 2: $S_i' = \{j | j \in S_i \text{ and } ASAP_j = ASAP_i\}$, $A = 0$ //target ASAP
 3: **while** $\cup S_i \neq \emptyset$ **do**
 4:    **repeat**
 5:       $flag = false$
 6:       **for** $i \in I$ **do**
 7:          **if** $ASAP_i = A$ and $S_i \neq \emptyset$ **then**
 8:             $i^* = argmax_{i \in S_i'}(m_i)$
 9:             $S_{i^*} = \emptyset$, $flag = true$
10:             **for** $j \in S_i$ **do**
11:                $P_i = P_i \cup \{i^*\}$
12:                $S_j = S_j - \{i^*\}$
13:             **end for**
14:             **break**
15:          **end if**
16:          **if** $flag$ **then**
17:             $A = A + 1;$
18:          **end if**
19:       **end for**
20:    **until** $A = \max(ASAP_i)$ and $flag$
21:    Compute $ASAP_i$, $ALAP_i$ and m_i
22:    $S_i' = \{j | j \in S_i \text{ and } ASAP_j = ASAP_i\}$
23: **end while**
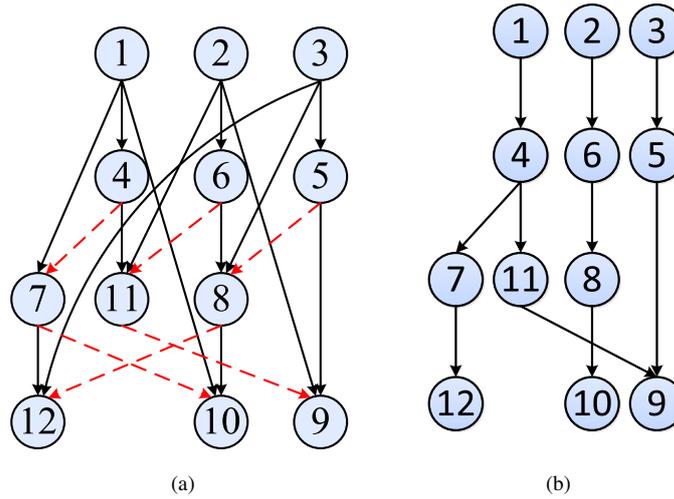
Fig. 10: (a) Example of QIDG in fig. 9 after preprocessing, (b) Example of scheduled instructions

effect of longer routes on the routing time, higher number of routed qubits increases congestion, and thus, increases the wait (stall) time. Evidently, increasing the number of concurrent instructions may delay the start of instruction execution times, and thereby, adversely affect the total latency of the quantum circuit. Therefore, to reduce the total latency of a quantum circuit, we must minimize the number of scheduling levels and evenly distribute concurrent instructions among all scheduling levels. This is a multi-objective optimization problem. To solve it, a modified version of the well-known force-directed scheduling (FDS) technique [26] is used as detailed below.

FDS is a technique used to schedule directed acyclic graphs so as to minimize the resource usage under a latency constraint in a shared resource system. The algorithm consists of three steps: (i) Determine the range of available time slots for scheduling an instruction; (ii) Create a distribution graph, which captures the resource pressures on each time slot, by assuming that an instruction is equally likely to be scheduled to start in any feasible time slot; (iii) A metric called *force* is defined to minimize the resource utilization. By repeatedly assigning instructions to all possible time slots and calculating the force associated with the assignment, several force values will be calculated. The instruction-to-time slot assignment with the lowest force value is chosen, which also balances the instruction concurrencies (i.e., reduces resource pressure).

The first goal of the quantum instruction scheduling is to minimize the number of concurrent instructions at each scheduling level. The second goal is to minimize the number of scheduling levels. To capture both of

these goals in our scheduling method, we use a latency-constrained FDS technique which tries to evenly distribute the instructions with fixed circuit latency with the addition of a fixed bound (N^m) on the number of concurrent instructions at each scheduling level. For this reason, after assigning an instruction to a scheduling level, the number of instructions assigned to that scheduling level is incremented by one. When this number reaches N^m, no new instruction is assigned to the scheduling level. Based this fact, assignment probabilities in the FDS technique are updated. If an instruction i is found to have no possible scheduling level assignment ($ALAP_i < ASAP_i$), the number of scheduling levels will be incremented by one and the assignment probabilities for the unassigned instructions will be re-calculated. This process continues until all instructions are scheduled.

Notice that there is a hard global limit on the number of concurrent instructions in each scheduling level, which is N^{max}. Our simulation results show that considering a limit less than this global value may decrease the total circuit latency due to a decrease in the routing latency in each scheduling level. Therefore, the scheduling process must be repeated for a few different $N^m \le N^{max}$ values and the best result (after final routing) chosen. In this paper, five different values for N^m are considered: $N^m \in \{N^{ma}, 0.8N^{ma}, 0.6N^{ma}, 0.4N^{ma}, 0.2N^{ma}\}$ where $N^{ma} = \min(N^{max}, \max_k |\{i|ASAP_i = k\}|)$.

7 Instruction placement

As mentioned in previous sections, the placement solution affects routing latencies, and hence, the start times of instructions. This fact motivates the development of a cross-layer optimization approach spanning the scheduling and placement steps in the quantum physical mapper. The proposed placement algorithm in this section is based on the state-of-the-art placement algorithm for placing logic gates in 2-D layouts. Modifications are made to account for the cross-later optimization issue. The placement solution is an iterative solution, which targets instructions in some scheduling level in each of its iterations. More precisely, at iteration L, considering placement solution for instructions in scheduling levels 0 to $L-1$ has already been obtained, a global instruction placement problem is formulated as a quadratic programming problem and solved by using a quadratic program solver. After this step, a host interaction well is determined for each instruction. Next, the *approximate start time* for each instruction in scheduling level L is calculated based on the fixed placement of its parent instructions and static routing latencies. If an instruction cannot satisfy the start time requirement of scheduling level L, it will be deferred by one scheduling level. Then, the force-directed scheduling algorithm is applied to determine the best scheduling solution based on the new deferrals. After applying the scheduling algorithm, the quadratic program solver is applied to the placement problem with new scheduling levels for instructions to obtain (and freeze) the placement solution for instructions in scheduling level L.

In this section, details of this procedure are presented. First, a mathematical formulation of the placement problem is given. Next, a solution without considering the effect of the placement on the scheduling solution is discussed, and, in the end, revisions to the algorithm in order to consider the effect of the routing latencies and placement solution on the scheduling solution are described.

7.1 Quantum instruction placement problem formulation

The ratio of the qubit movement latency to the instruction latency shows the importance of the routing latency in a quantum fabric realization. For example, in fig. 3 the minimum Manhattan distance between two interaction wells is 10. Considering 1/10 as the ratio of single move latency to the two-qubit instruction latency according to [8], the minimum routing latency is as large as the latency of performing one instruction, which underlines the importance of the routing latency.

Similar to routing steps, routing latency can be divided into two components: 1) *static routing latency*, and, 2) *dynamic routing latency*. The static routing latency is related to the Manhattan distance between two interaction wells whereas the dynamic routing latency is related to stall times of the qubit in channels due to transient saturation of the channel capacities.

The dynamic behavior of the routing latency is hard to predict in a fully analytical way. In other words, the quantum circuit should in fact be simulated in order to determine the dynamic routing latency. Based on this line of reasoning, only static routing latency is considered in the quantum instruction placement formulation.

There are two types of qubits in the circuit: 1) *I/O qubits*, and, 2) *ancillary qubits*. I/O qubits enter and exit ULBs from specified locations whereas ancillary qubits are created, consumed and then purified to be used again for other instructions. Let Q^a, Q^o and Q denote the set of ancillary, I/O and all qubits in the QIDG, respectively. Similarly, Q_i^a, Q_i^o and Q_i denote the set of operand ancillary, I/O and all qubits of instruction i, respectively.

The scheduling level and the parent set for each quantum instruction are determined in the quantum instruction scheduling step. In the quantum instruction placement step, each instruction should be assigned to some interaction well considering its scheduling level. Recall that the placement solution determines the routing latency for each qubit movement, which can change the scheduling level of instructions, and subsequently, affects the start time of the quantum instructions.

The *quantum instruction placement problem* (QIPP) can be formulated as follows:

$$min\left\{ \ \max_{q \in Q^o} \left\{ t_q \right\} \ \right\}$$

subject to:

$$t_i \geq t_j + d_j + \sum_{u \in W} \sum_{v \in W} y_{iu} y_{jv} R_{uv} \qquad \forall j \in P_i \qquad (8)$$

$$t_i \geq \sum_{c \in C \cup I} \sum_{w \in W} y_{iw} z_{qc} I_{cw} \qquad \forall q \in Q_i \qquad (9)$$

$$t_q^f \geq t_i + d_i + \sum_{c \in I} \sum_{w \in W} y_{iw} z_{qc} I_{cw} \ , \qquad \forall i | q \in Q_i^o \qquad (10)$$

$$t_i \geq (t_j + d_j) \sum_{w \in W} y_{iw} y_{jw} \qquad if \ (t_i \geq t_j) \qquad (11)$$

$$y_{iw} \in \{0,1\}, z_{qc} \in \{0,1\} \qquad (12)$$

$$\sum_{w \in W} y_{iw} = 1, \sum_{c \in C} z_{qc} = 1, \forall q \in Q^a \qquad (13)$$

where t_q denotes the time when qubit q arrives at its output location after finishing the last instruction, W, C and I denote the set of locations of the interaction, creation wells and input/output ports in the circuit, respectively, d_i is the latency of the ith instruction, R_{uv} is the static routing latency between the interaction well u and v, R_{cw} denotes the static routing time from the creation well (or input port) c to the interaction well w, and y_{iw} is an assignment parameter that determines if the instruction is assigned to an interaction well (when it is equal to 1) or not (when 0). Similarly, z_{qc} denotes an assignment parameter that determines if the qubit is assigned to a creation well (or input port) or not. For input qubits, this parameter is fixed. Parameters y_{iw}, z_{qc} and t_i are the optimization parameters in the QIPP. Note that the result of the quantum instruction scheduling cannot determine the exact timing of the instructions and can only determine the order of dependent instructions.

The gaol in QIPP is to minimize the total latency in the quantum circuit. Constraint (8) determines the earliest start time of each instruction based on the start time, static routing time and latency of its parent instructions. Constraint (9) determines the earliest start time of the quantum instruction that accesses qubit(s) for the first time. Constraint (10) determines the time when qubits arrive at their output locations after finishing the last instructions. Constraint (11) limits the number of concurrent instructions that can be executed at each interaction well to one. Constraints (12) and (13) define the binary assignment parameters.

QIPP is a non-linear, mixed-integer problem. From another perspective, QIPP is a timing-driven 3D placement problem. The first two dimensions correspond to the X and Y dimensions of the quantum fabric whereas the third dimension represents time. In a regular placement problem, the overlap between different placement objects is not acceptable, but in QIPP, the physical overlap is acceptable as long as the execution times of different instructions do not coincide. A sample instruction placement solution for the QIDG shown in fig. 10b is depicted in fig. 11. Each rectangular cube represents an instruction placed at an interaction well on the ULB. Longer rectangular cubes represent two instructions placed at an interaction well in two consecutive scheduling times.

In this paper, to manage the problem complexity, time is discretized. The duration of each time slot (D) is considered to be the latency of the fastest instruction such as a rotation operation. Also to solve the timing-driven placement, a net-weighting technique is employed [27]. Details of the problem formulation and the algorithm are presented next.

7.2 Timing-driven force-directed placement based on fixed instruction levels

Considering fixed instruction levels based on the result of the quantum instruction scheduling step, the QIPP can be simplified to placing instructions on interaction wells such that there are no overlaps among instructions with the same scheduling level. Note that the routing latency of instructions may delay instructions from their original scheduled time. Thus, minimizing the routing latencies can decrease the time shifts and eventually decreases the total latency of the circuit. However, not all routing latencies have the same importance. An increase in the routing latency of qubits related to an instruction that lies in the critical path of the QIDG will increase the total latency
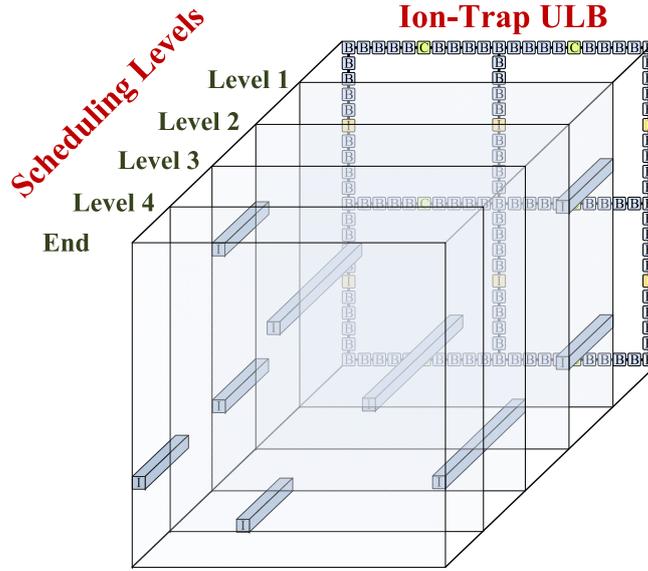
Fig. 11: Sample instruction placement solution for QIDG of fig. 9

of the circuit, while adding to the routing latency of qubits related to an instruction in a non-critical path may only cause a time shift on the start times of the remaining instructions in that path without changing the total latency of the circuit. To account for this fact, the goal of the placement solution is simplified to minimizing the weighted sum of the routing latencies. The weight for the qubit routing latency between the ith (child) and the jth (parent) instructions is called m_{ij} and is found from equation (14).

$$m_{ij} = min(m^{max}, 1/(ALAP_i - ASAP_i + SL_i - SL_j - 1))$$ (14)

where, m^{max} denotes the maximum weight for cases with infinity weight.

This weight is in inverse proportion to the difference between ALAP and ASAP levels of the child instruction and also the difference between levels of the child and parent instructions. The higher the difference between the child and parent instruction levels, the more time the qubit has to be routed without delaying the child instruction. Moreover, the difference between ALAP and ASAP levels of the child instruction shows the criticality of this instruction to the rest of the circuit.

To model the routing latency in this problem, we use the Manhattan distance between the parent and child instruction locations on the circuit fabric. Using the aforementioned simplifications, the quantum instruction placement problem with *fixed instruction level* (QIPP-FIL) is to minimize the weighted summation of the routing latencies in the quantum circuit. Moreover, no more than one instruction with a specific level can be placed on any interaction well. The effect of initial qubit routing latencies (between the initial qubit locations and the location of the first instructions that use them) and final routing latencies (between the last operation that uses I/O qubits and their final locations) is considered with weights calculated from equation (14) when $SL_i = SL_j + 1$.

QIPP-FIL consists of an instruction placement problem with per-level overlap avoidance constraints plus a qubit placement problem. To solve this problem, an instruction placement technique is used, which is followed by placing the ancillary qubits on the creation sites based on the instruction placement solution.

To place the instructions on the quantum circuit fabric, a modified version of the state-of-the-art force-directed placement tool, SimPL [28], is used. SimPL works best when placing 10^4 to 10^7 objects on a circuit fabric. This range complies with the number of instructions that we have for the placement step considering multi-level concatenated error correction scheme for transversal and non-transversal operations. SimPL is reviewed next.

To minimize the half perimeter wire length in VLSI circuits, without any overlap avoidance constraints, an iterative optimization procedure is used. Starting from an initial solution, a quadratic optimization problem is formulated and solved. Distances between placed components are used to formulate a new quadratic optimization problem to be solved. After convergence of the solution, the placement result is the optimal solution. To remove the overlaps, an algorithm called *rough legalization*, which uses a *non-linear scaling* method is proposed in [28]. Based on this approach, over-utilized areas in the fabric are examined and components that are placed in those areas are distributed evenly in the surrounding area. To avoid components from reverting back to their original locations,

a *pseudo-net* is added in the new location of any moved component and a wire is added between that pseudo-node and the original component. Weight of the added wire is increased as the number of iterations increases. Note that adding pseudo-nets does not change the problem size because it does not add to the number of variables or equations and only changes the objective function related to the original node. After this modification, the half perimeter wire length minimization problem is formed and solved again. This process continues until all overlaps are removed. Finally, a simple approach is used for final legalization. SimPL improves the speed of prior placement tools and has similar quality to the analytical placement solutions.

Due to the difference between the quantum instruction placement problem and the VLSI placement problem, some modifications must be made to SimPL as explained next. Instead of minimizing the total routing latencies, a weighted summation of the routing latencies is minimized. Moreover, the rough legalization step in this solution is applied to the instructions in each scheduling level. In particular, only overlaps between instructions on the same scheduling level are removed with the non-linear scaling approach. Note that changing the locations of instructions during the rough legalization in one level affects the placement solution for other scheduling levels. Therefore, to consider this effect, the quadratic optimization solution is repeated after finalizing the rough legalization for each scheduling level. To find the final placement solution (assignment of instructions to interaction wells), similar to rough legalization, each scheduling level is processed separately. For each scheduling level, the interaction wells are sorted based on their distance to the instructions, and then the best interaction well for each instruction is chosen.

After finalizing the placement solution for all instructions, the best creation well for each ancillary qubit is determined. Each ancillary qubit is assigned to the free creation well that is closest to the location of the instruction that uses it for the first time.

7.3 Timing-driven force-directed placement with variable instruction levels

The proposed method based on classical placement tool plus net-weighting relies on a fixed scheduling level for each instruction. This assumption is violated with the addition of the routing latencies. More precisely, adding the routing latencies may significantly vary the start time of each instruction from its nominal start time. Considering fixed levels for instructions in the placement problem has two negative effects on the placement solution. First, there may be a large number of instructions at some scheduling level, and hence, distributing them on the circuit fabric may cause some of the instructions to be placed far from the interaction wells hosting their parent instructions. However, if the scheduling level of those instructions is changed, they can be placed near the interaction wells hosting their parent instructions. This may even reduce the finish time for the deferred instructions. Second, placing instructions on the circuit fabric based on the fixed scheduling level relies on an assumption about the start and stop times of the previous level instructions. This assumption typically does not, which results in the deferral of every instruction that is placed on the same interaction well in the next scheduling levels, cf. fig. 8b.

In order to avoid big differences between nominal and actual start times for an instruction, a rule can be added to the placement solution as follows. The start time of an instruction assigned to scheduling level L should be less than a threshold time (T_L^{th}). This threshold start time can be the middle time in time slot L. This rule results in higher accuracy of the execution time prediction for instructions in different scheduling levels. To modify the placement solution based on this rule, we propose an iterative approach. In each iteration, instructions in one scheduling level are fixed. In iteration L: (i) considering fixed placement solution for instructions in scheduling level 0 to $L-1$, the proposed solution for QIPP-FIL is applied; (ii) an approximate start time for each instruction (T_i^{min}) in scheduling level L is calculated based on the distance between the location of the interaction well assigned to it and locations of the interaction wells hosting the parent instructions. If an instruction cannot satisfy the start time condition ($T_i^{min} <= T_L^{th}$), it will be deferred by one scheduling level ($SL_i^{min} = L+1$); (iii) All of the finalized instructions that do not satisfy the timing constraints are removed from the quantum circuit fabric; (iv) The force-directed scheduling step is applied to determine the best scheduling solution for all 'floating' instructions based on the new deferrals. The instruction scheduling step is augmented to support the new parameter (SL_i^{min}); (v) The proposed solution for QIPP-FIL is applied to the placement problem with new scheduling levels for instructions to fix the placement solution for the remaining instructions in the scheduling level L.

The proposed approach may increase the number of scheduling levels, but it significantly decreases the overall latency of the circuit due to considering a more realistic scheduling solution. The flowchart for finalizing the placement for each scheduling level is provided in fig. 12.

Notice that repeating the scheduling and global placement after each scheduling level with at least one deferred instruction is costly in case of large circuits. Instead of this approach, the algorithm can be changed to repeat the scheduling and global placement algorithms after the number of deferred instructions reaches a threshold value or
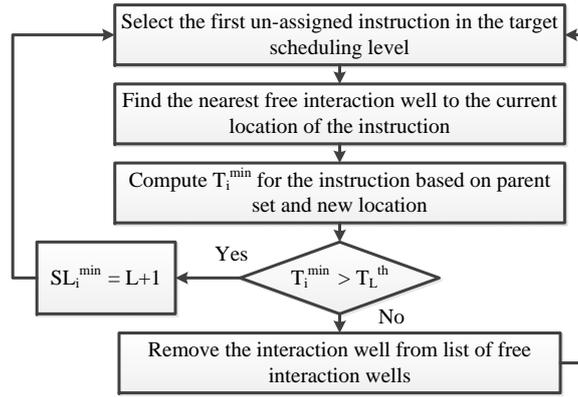
Fig. 12: Finalization step for instruction placement

the summation of $(T_i^{min} - T_L^{th})$ for deferred instructions reaches a limit. Using this mechanism, the run time of the proposed solution can be shortened at the cost of a slight increase in the resulting circuit latency.

8 Experimental Results

8.1 Experimental Setup

QUFD is implemented in Java. Trapped-ion parameters used for the simulations are as follows: move delay=$10\mu s$; one-qubit primitive instruction delay=$50\mu s$; and two-qubit primitive instruction delay=$100\mu s$. A workstation with Intel Core i7-2600 3.4GHz CPU and 8GB RAM was used to run the simulations.

Determining the concatenation level for the encoding, which is sufficient to get a correct result with high fidelity from the quantum circuit is a process based on the characteristics of the underlying quantum circuit technology and the implemented quantum circuit. This problem is outside the scope of this paper. In this experiment, we focus on one-level Steane code [[7,1,3]] for quantum error correction and the Steane syndrome extraction scheme. Results for some other combinations of encoding and syndrome extraction schemes will also be presented.

Different ULB sizes are enumerated. For each ULB size, the latency of executing each of the FT operations (CNOT, T, S, X, Y, and Z) encoded with Steane code [[7,1,3]] are found by applying QUFD. Based on the results of the quantum physical mapper and target quantum circuit profiling results, the best ULB size is determined.

8.2 Experimental results for different FT operations and different ULB sizes

The most important part of QUFD is the quantum physical mapper for trapped-ion technology. There are different quantum mapping tools in the literature such as [8], QUALE [11], QPOS [12], and QSPR [16]. In contrast to our work, all of these approaches rely on ASAP or ALAP scheduling and greedy placement of the qubits and instructions, which in turn results in long circuit latency. QSPR improved on these approaches by repeating the scheduling and placement solution with random initial qubit placement solutions to reach a better initial qubit placement solution, which is critical in case of applying a greedy instruction placement solution. QSPR algorithm provides about 41% improvement with respect to QUALE. In the following we provide a detailed comparison between QUFD and QSPR [16], which is the most advanced mapping tool in the literature.

Latencies of various FT operations implemented with different ULB sizes by using QUFD and QSPR are reported in fig. 13. Assuming simultaneous creation of ancillary qubits, there exist no feasible mapping solutions for the T instruction (gate) on 6×6 and 8×8 ULBs. This is because the number of the ancillary qubits for this instruction is more than the number of creation wells in the aforesaid ULBs. QSPR fails to find any mapping solution for the 10×10 and 12×12 ULBs while QUFD succeeds in these cases.

The FT operation latencies obtained by QUFD are from 35% (for CNOT) to 41% (for T instruction) lower than those produced by QSPR. Although QUFD uses larger computation time compared to QSPR to produce its solutions, this is not a major concern, because we are only designing ULBs for seven FT operations and this procedure is done only once for the FT operation library design for each quantum circuit fabric.
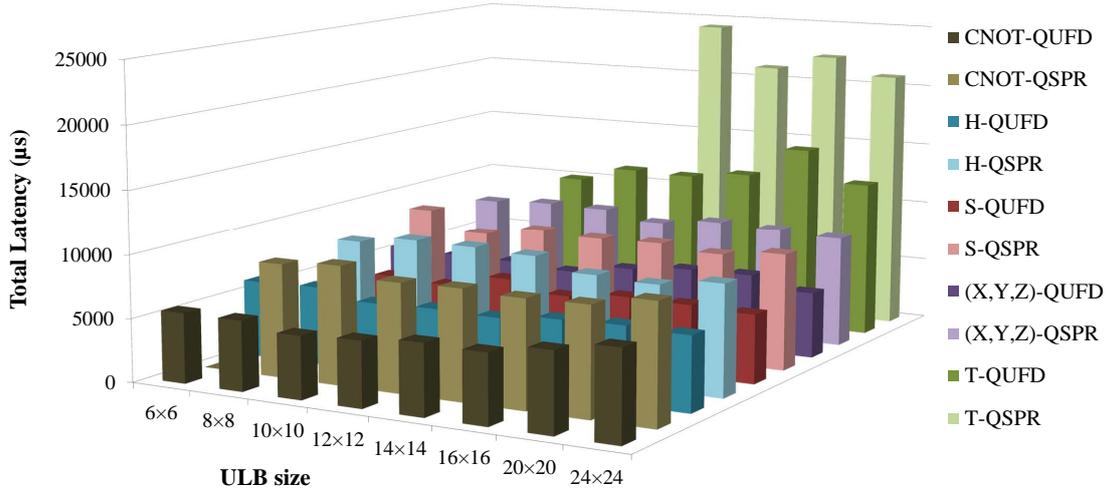
Fig. 13: Latencies of different FT operations as a function of ULB sizes

As mentioned in section 6, tightening the upper-bound on the number of instructions being scheduled in each scheduling level may decrease the total latency of the circuit by decreasing the routing time even though it may increase the number of scheduling levels at the end. This effect is examined for the CNOT instruction mapped to a 10×10 ULB in fig. 14. As can be seen in this figure, if the number of concurrent instructions is limited to 30% of the physical interaction wells on the fabric, the total latency is decreased by 18%. It is also seen that decreasing the maximum number of simultaneously active interaction wells does not change the circuit latency until this limit exceeds the maximum number of concurrent instruction in the QIDG ($0.4 < \alpha < 1$ in fig. 14). Decreasing the maximum number of simultaneously active interaction wells after this value changes the scheduling, placement and routing solution by decreasing the resource contention in each scheduling level for placement and routing problems (positive effect) and possible QIDG depth increase (negative effect). The positive effect is dominant for a part of this process and after that the negative effect starts to dominate.
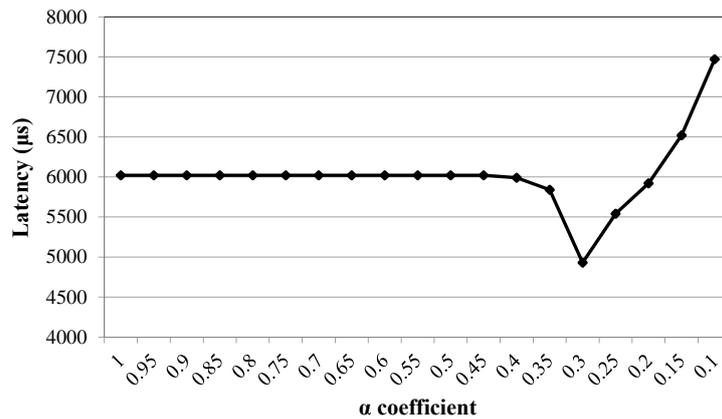

Fig. 14: Effect of changing α as the coefficient of N^{max}

To show the effect of enumerating different N^m for the FDS method, (normalized) total latency of the circuit with and without considering this factor for a complete set of FT operations (CNOT, H and T) plus one of the Pauli operations in a 10×10 ULB is reported in fig. 15. As can be seen, for H and X operations, enumeration of different N^m in the scheduling algorithm does not change the total latency but for CNOT and T operations, this enumeration greatly reduces the total latency.

Fig. 16 shows an exemplary case for the maximum scheduling level in different placement steps for the FT CNOT instruction mapping on a 6×6 ULB. Clearly, as we go further in the placement iterations, the number of scheduling levels is increased when the deferred instruction(s) is on the critical path of the circuit. The actual
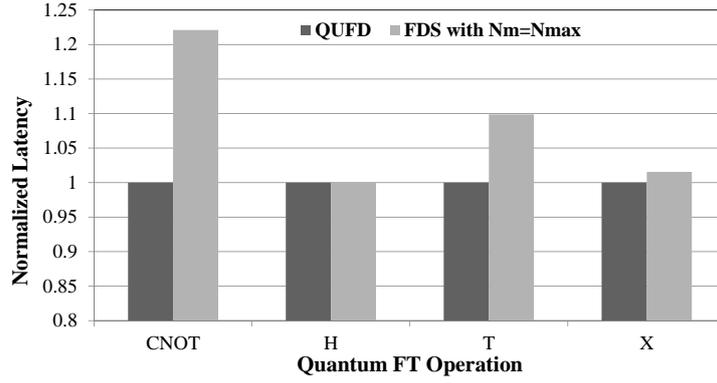
Fig. 15: Effect of enumeration on N^m in the scheduling algorithm on the circuit latency

number of scheduling levels for this case is equal to 51. The predicted number of scheduling levels after scheduling and placement solutions is increased from 26 (without considering the routing latencies) to 43 (after iterative correction of the scheduling levels based on the placement solution).
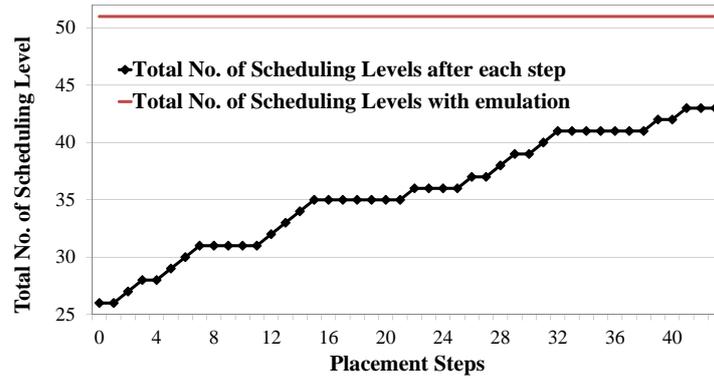

Fig. 16: Effect of the placement iterations on the number scheduling levels

To show the effectiveness of the placement method that utilizes force-directed scheduling to modify the scheduling solution after each placement step, we implemented two other placement methods in the VLSI area. The first algorithm is based on the force-directed placement solution [29]. In this approach, each instruction is considered to be a mass and data dependencies in the circuits are modeled by springs. The constant for each spring is set by the criticality factor of the source and the destination qubits. To avoid overlap between instructions with the same scheduling level, repulsive forces are used. The second algorithm is based on SimPL algorithm [28] with fixed scheduling levels for instructions. Normalized latencies (w.r.t. QUFD results) for a complete set of FT operations (CNOT, H and T) plus one of the Pauli operations in a 12×12 ULB for different placement mechanisms are presented in fig. 17.

As can be seen, the proposed placement method that modifies the scheduling level of instructions based on the placement solution generates on average 27% better results than the same placement method without using adaptive scheduling levels. Moreover, the placement method based on the force-directed placement generates on average 72% worse results compared to the proposed placement solution.

In this experiment, we focus on one-level Steane encoding [[7,1,3]] and Steane syndrome extraction scheme but to show the difference between encoding and syndrome extraction schemes, results of another concatenated code, that is, Bacon-Shor with the lowest concatenation level and different syndrome extraction schemes (Steane and Knill ancilla) are reported in fig. 18.

In this figure, latencies of CNOT, H and X operations are measured in a 12×12 ULB. For the T operation, a 20×20 ULB is used because the number of required ancillary qubits for the T gate for some of the schemes is higher than the number of available creation wells on a 12×12 ULB. Note, however, that these encoding/syndrome
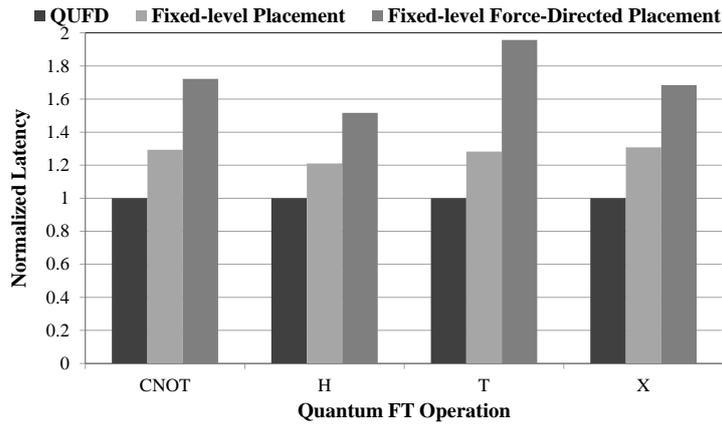
Fig. 17: Normalized latency of different operations utilizing different placement solutions
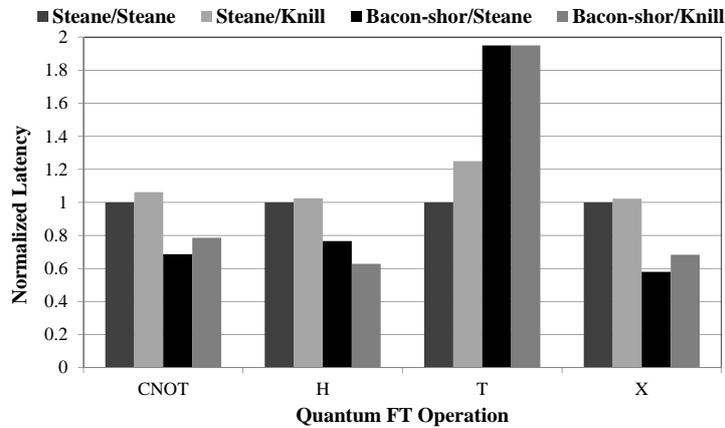

Fig. 18: Normalized latency of different operations in different encoding/syndrome extraction schemes

extraction schemes provide different levels of error detection/correction capabilities for the quantum circuit and cannot be compared solely based on these latencies.

8.3 Selecting the best ULB size

Based on the found latencies for FT operations, the total latency of the mapped Toffoli circuit to a 2×2 mesh TQA, which is calculated in section 4, is reported in fig. 19. Note that results for the 10×10 and 12×12 ULB sizes are not reported for QSPR because this tool fails to find the total latency of the T instruction for these ULB sizes.
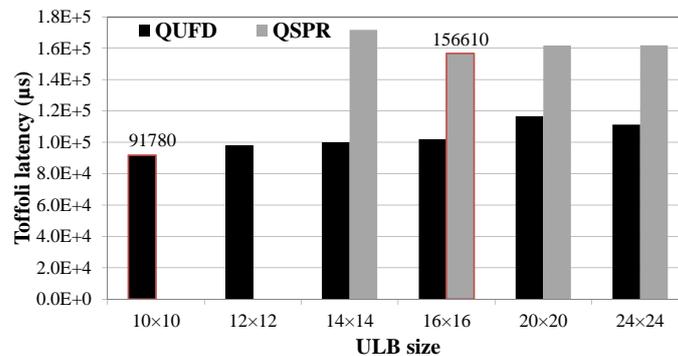

Fig. 19: Latency of Toffoli circuit for different ULB sizes

As can be seen, the best ULB size for QUFD is 10×10 and the best ULB size for QSPR is 16×16. Moreover, the best total latency for the proposed QUFD is 41% less than the best total latency of QSPR.

9 Conclusion

This paper presented quantum scheduling and placement algorithms for mapping fault-tolerant quantum operations to a universal logic block (ULB). Analytical formulations for the scheduling and placement problems were also presented. Effective and state-of-the-art algorithms used in standard CAD tools for classical circuits were modified to be applicable to the quantum mapping problem (a quantum CAD problem). In particular, modified list scheduling and force-directed scheduling were applied to the quantum instruction scheduling problem to capture the no-cloning dependencies among quantum instructions and determine the scheduling levels to minimize the number of instruction levels and reduce the resource contention. To consider the large impact of the qubit routing on the total latency of the quantum circuit, a net-weighting timing-driven placement solution based on SimPL package was employed. This technique adaptively modifies the scheduling level of the instructions. Moreover, a technique was described to determine the best ULB size for implementing FT quantum operations. All of the aforementioned algorithms were implemented in a CAD tool called QUFD. Experimental results showed that QUFD dominated the prior art quantum mappers by as much as 41% in terms of the average latency reduction.

With some modifications, QUFD can be applied to large QODG's to produce a mapping solution for any tile-based quantum architectures. Moreover, the proposed flow is applicable to quantum mapping and ULB design in other quantum fabric technologies with small changes due to different qubit/information movement cost.

References

1. M.A. Nielsen, I.L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, 10th edn. (Cambridge University Press, 2011)
2. J. Preskill, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **454**(1969), 385 (1998). DOI 10.1098/rspa.1998.0167
3. E. Knill, R. Laflamme, W.H. Zurek, Science **279**(5349), 342 (1998). DOI 10.1126/science.279.5349.342
4. P. Schindler, J.T. Barreiro, T. Monz, V. Nebendahl, D. Nigg, M. Chwalla, M. Hennrich, R. Blatt, Science **332**(6033), 1059 (2011). DOI 10.1126/science.1203329
5. A. Steane, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences **452**(1954), 2551 (1996). DOI 10.1098/rspa.1996.0136
6. M. Saeedi, I.L. Markov, ACM Computing Surveys (2011). URL http://arxiv.org/abs/1110.2574
7. T.S. Metodi, D.D. Thaker, A.W. Cross, F.T. Chong, I.L. Chuang, in *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture* (IEEE Computer Society, Washington, DC, USA, 2005), MICRO 38, pp. 305–318. DOI 10.1109/MICRO.2005.9. URL http://dx.doi.org/10.1109/MICRO.2005.9
8. M.G. Whitney, N. Isailovic, Y. Patel, J. Kubiatowicz, in *Proceedings of the 36th annual international symposium on Computer architecture* (ACM, New York, NY, USA, 2009), ISCA '09, pp. 383–394. DOI 10.1145/1555754.1555802. URL http://doi.acm.org/10.1145/1555754.1555802
9. J.I. Cirac, A.K. Ekert, C. Macchiavello, Physical Review Letters **82**(21), 4344 (1999). DOI 10.1103/PhysRevLett.82.4344
10. T.D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, J.L. O'Brien, Nature **464**(7285), 45 (2010). DOI 10.1038/nature08812
11. S. Balensiefer, L. Kreger-Stickles, M. Oskin, Proceedings of SPIE **5815**(1), 103 (2005). DOI doi:10.1117/12.604073
12. T.S. Metodi, D.D. Thaker, A.W. Cross, F.T. Chong, I.L. Chuang, Proceedings of SPIE **6244**(1), 62440T (2006). DOI doi:10.1117/12.666419
13. M. Whitney, N. Isailovic, Y. Patel, J. Kubiatowicz, in *Proceedings of the 4th international conference on Computing frontiers* (ACM, New York, NY, USA, 2007), CF '07, pp. 83–94. DOI 10.1145/1242531.1242546. URL http://doi.acm.org/10.1145/1242531.1242546
14. D. Maslov, S. Falconer, M. Mosca, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **27**(4), 752 (2008). DOI 10.1109/TCAD.2008.917562
15. N. Mohammadzadeh, M. Zamani, M. Sedighi, Quantum Information Processing **10**(2), 139 (2011). DOI 10.1007/s11128-010-0183-0
16. M.J. Dousti, M. Pedram, in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* (2012), pp. 840–843
17. L. McMurchie, C. Ebeling, in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays* (ACM, New York, NY, USA, 1995), FPGA '95, pp. 111–117. DOI 10.1145/201310.201328. URL http://doi.acm.org/10.1145/201310.201328
18. S. Balensiefer, L. Kregor-Stickles, M. Oskin, SIGARCH Comput. Archit. News **33**(2), 186196 (2005). DOI 10.1145/1080695.1069986
19. M.A. Rowe, A. Ben-Kish, B. DeMarco, D. Leibfried, V. Meyer, J. Beall, J. Britton, J. Hughes, W.M. Itano, B. Jelenkovic, C. Langer, T. Rosenband, D.J. Wineland, arXiv:quant-ph/0205094 (2002). URL http://arxiv.org/abs/quant-ph/0205094
20. W.K. Hensinger, S. Olmschenk, D. Stick, D. Hucul, M. Yeo, M. Acton, L. Deslauriers, C. Monroe, J. Rabchuk, Applied Physics Letters **88**(3), 034101 (2006). DOI doi:10.1063/1.2164910

21. D.R. Leibrandt, J. Labaziewicz, R.J. Clark, I.L. Chuang, R.J. Epstein, C. Ospelkaus, J.H. Wesenberg, J.J. Bollinger, D. Leibfried, D.J. Wineland, D. Stick, J. Sterk, C. Monroe, C.S. Pai, Y. Low, R. Frahm, R.E. Slusher, arXiv:0904.2599 (2009). URL http://arxiv.org/abs/0904.2599

22. V.V. Shende, I.L. Markov, Quantum Info. Comput. **9**(5), 461486 (2009)

23. W.K. Wootters, W.H. Zurek, Nature **299**(5886), 802 (1982). DOI 10.1038/299802a0

24. G. De Micheli, *Synthesis and Optimization of Digital Circuits*, 1st edn. (McGraw-Hill Science/Engineering/Math, 1994)

25. S. Davidson, D. Landskov, B. Shriver, P. Mallett, Computers, IEEE Transactions on **C-30**(7), 460 (1981). DOI 10.1109/TC.1981.1675826

26. P. Paulin, J. Knight, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **8**(6), 661 (1989). DOI 10.1109/43.31522

27. D.Z. Pan, H. B., R. H., in *Handbook of Algorithms for VLSI Physical Automation*, ed. by A.C. J., M.D. P., S.S. S. (CRC Press, 2007), pp. 223–233

28. M.C. Kim, D.J. Lee, I. Markov, in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on* (2010), pp. 649 –656. DOI 10.1109/ICCAD.2010.5654229

29. N. Quinn, M. Breuer, Circuits and Systems, IEEE Transactions on **26**(6), 377 (1979). DOI 10.1109/TCS.1979.1084652