QRANE: Lifting QASM Programs to an Affine IR

Blake Gerard
The University of Oklahoma
Norman, Oklahoma, USA
blakegerard@ou.edu

Tobias Grosser
The University of Edinburgh
Edinburgh, Scotland, UK
tobias.grosser@ed.ac.uk

Martin Kong
The University of Oklahoma
Norman, Oklahoma, USA
mkong@ou.edu

Abstract

This paper introduces QRANE, a tool that produces the affine intermediate representation (IR) from a quantum program expressed in Quantum Assembly language such as OpenQASM. QRANE finds subsets of quantum gates prescribed by the same operation type and linear relationships, and constructs a structured program representation expressed with polyhedral iteration domains and access relations, all while preserving the original semantics of the quantum program. We explore various policies for deciding amongst different delinearization strategies and discuss their effect on the quality of the reconstruction. Our evaluation demonstrates the high coverage and efficiency obtained with QRANE while enabling research on the benefits of affine transformations for large quantum circuits. Specifically, QRANE reconstructs affine iteration domains of up to 6 dimensions and up to 184 points per domain.

CCS Concepts: • **Computer systems organization** → **Quantum computing**; • **Software and its engineering** → **Translator writing systems and compiler generators**.

Keywords: quantum assembly, polyhedral model, delinearization, quantum affine computing.

ACM Reference Format:
Blake Gerard, Tobias Grosser, and Martin Kong. 2022. QRANE: Lifting QASM Programs to an Affine IR. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction (CC '22), April 02–03, 2022, Seoul, South Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3497776.3517775

1 Introduction

Quantum Computing (QC) has gained considerable traction in the last decade, owing to several novel high-level programming languages being introduced (QUIPPER [20], SILQ [5]) as well as many efforts to improve the overall compilation pipeline in problems such as qubit allocation and layout synthesis [42, 54, 56, 68], improving the reliability of quantum programs via noise-adaptive and noise-aware schemes [45, 58], and breaking abstractions [28, 52].

Regardless of the programming language used to specify a quantum program, ultimately, it has to be compiled down to a low-level representation such as OpenQASM [14], a quantum assembly language. At this point, the program effectively consists of long sequences of single- and two-qubit quantum gates. Lowering the representation to this level of abstraction results in significant loss of structured information that can be beneficial and heavily impactful in later compilation stages, such as qubit mapping [54], where the compiler typically constructs a dependence graph from the QASM representation, proceeding to extract information from only the first few network layers, and mapping qubits that participate in long paths (or cycles) found in the dependence graph [54, 68].

Recent works have proposed the use of loop-based optimization frameworks to analyze and optimize quantum programs [24, 36]. In addition, the strong impact that polyhedral compilation techniques and affine frameworks have had on high-performance computing and domain specific languages [3, 6, 8, 15, 22, 34, 59, 60, 62] leads to the next natural question: **can polyhedral and affine techniques contribute to the same extent to the field of quantum computing?**. This work is a significant step forward to answer this question. We introduce `QRANE`, a tool for automatically lifting quantum programs, written in quantum assembly, that exhibit static control (i.e. no dynamic control conditions) to the intermediate representation (IR) used in polyhedral frameworks. Our work is complementary to previous studies that explored the applicability and potential of affine transformations and abstractions on QC [18, 39].

`QRANE` fully automates the *delinearization* process [63], the detection and reconstruction of linear relationships among qubit indices, producing the well-known polyhedral abstractions: iteration domains, access relations and the program schedule [19, 21]. We discuss the delinearization mechanism, the challenges faced and the meta-parameters that control the quality and behavior of the reconstruction. Our goal, is to provide the entry point by which affine analyses and transformations can contribute in the same way that they have to classical computing.

In summary, our contributions are the following: i) The design and implementation of the first quantum assembly delinearizer that produces the IR abstractions used in affine

CC '22, April 02–03, 2022, Seoul, South Korea

ACM ISBN 978-1-4503-9183-2/22/04.
https://doi.org/10.1145/3497776.3517775

compilation frameworks (Sec. 4 and Sec. 5); ii) A comprehensive evaluation to demonstrate the quality and coverage product of the delinearization process, the efficiency of the reconstruction implementation, and the effect of the meta-parameters when lifting quantum assembly (Sec. 6); iii) A tool serving as the entry point to exploit affine transformations in quantum compilation, and a preliminary study demonstrating the current potential.

2 Motivation and Related Work

Limitations of Current Approaches. Quantum compilers compute space-time mappings of the quantum operations (Q-OPS) in a program to a quantum processor, modeled as a (coupling) graph. Most scheduling and mapping techniques fall into one of the following categories: i) graph traversals and peephole optimizations [9, 12, 30, 35, 42, 43, 52–55, 66, 67], ii) Precise, instance-wise formulations using Integer Linear Programming and/or Satisfiability Modulo Theory solvers [4, 45, 46, 56], iii) Exhaustive enumeration and search techniques [54, 68]. The above techniques aim to tackle variations of the qubit mapping problem, which has been proven to be NP-complete [43]. Hence, scalable and yet precise methods are necessary. The three classes of techniques span the spectrum of fast but approximate results (type i), and slow (and computationally expensive) but precise solutions (types ii and iii). While some graph-based techniques performing multiple passes over the DAG could scale to millions of Q-OPS, this is not the common case. Lack of scalability hinders several of these methods, especially when they utilize expensive graph operations such as computing (Hamiltonian) cycles, paths with properties, and isomorphisms. Methods of the class ii, rely on precise formulations for each quantum operation of the program, and often producing $O(g^2)$ (or more) constraints on the number of Q-OPS. These techniques are only applicable to extremely small quantum programs, owing to ILP solving being double-exponential in their dimensionality.

Benefits of using QRANE. QRANE produces regular, well-structured representations of QASM programs. Its benefits are two-fold: First, grouping gates that have parallel dependencies enables us to collapse them into a single structure, ultimately reducing the dimensionality of the ILP problem we solve and, consequently, its complexity (solving ILPs is double-exponential in the dimensionality). Second, a good grouping maintains scheduling freedom and flexibility by exposing a structured representation that makes the dependence patterns in the original circuit explicit. The complexity of polyhedral scheduling is (mostly) independent of the number of points in an iteration domain, as for each iteration domain exactly one shared affine scheduling function is computed.

QRANE's Long Term Impact. Being the first known tool to extract a regular structure from QASM programs, we see QRANE as the foundation of a larger ecosystem of polyhedral tool-chains for QC. Analyses and optimizations over regular structures are often computationally cheaper and easier to understand. By exposing such regular structure, QRANE enables the development of more aggressive optimizations, e.g. scheduling in the polyhedral model or even applying machine learning over the affine IR, both of which have already brought enormous benefits in classical computing. In fact, lifting with QRANE already reduces the size of the program representation asymptotically. As more concrete future benefits, using our tool, the community can now focus on high-level compiler scheduling heuristics that exploit quantum domain knowledge, e.g. commutative, canceling and identity properties [52].

Scalable Compiler Techniques. One of our driving motivations is to show that, soon enough, current quantum compilation techniques will hit a scalability wall. Recent qubit allocator approaches such as Zulehner et al.'s [68] and Siraichi et al.[54], which leverage A* search techniques [26] and dynamic programming, respectively, have proven that even relatively small circuits can incur high complexity execution times and memory footprints. For example, Fig.1 shows the compilation times obtained with the TKET compiler [55] on circuits of the IBM-QX circuit set [33] that have more than 100,000 gates. We can see that compilation time will continue to increase as a function of the number of QASM operations in the near term. In Section 6.3 we demonstrate that QRANE scales effectively to very large circuits.

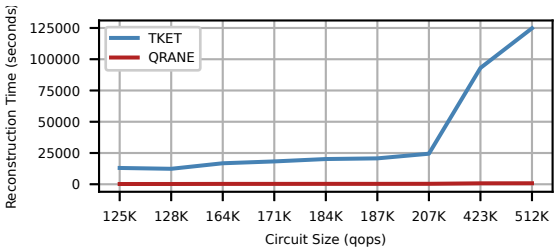


Figure 1. QRANE and TKET Processing Time for the Largest IBMQX Circuits

Delinearization and Affine Transformations. QRANE lifts quantum assembly representations into an affine IR with high efficiency and efficacy, as supported by Table 1, where we present the coverage obtained for the largest circuit of the benchmark which contains 0.5M quantum operations. QRANE achieves a compression factor of 4.2x, relative to the number of domains produced for the circuit, by aggregating quantum operations into expressive, multi-dimensional affine structures.

Table 1. Domain Dimensionality Profile for IBMQX's Largest (0.5M) Circuit

Dimension	Domain Count	Point Count
1D	943	959
2D	70374	185905
3D	43155	226161
4D	7943	82753
5D	896	16286
Total	123311	512064

Table 2 shows an excerpt of our results that demonstrate the potential impact of affine transformations on lifted quantum assembly programs. We apply two well-known loop transformation heuristics, PLuTo Min/Max-fuse, to select circuits of the QUEKO-BIGD dataset [57]. The "TKET" column data are the circuit depths (critical path) achieved by TKET with the original circuit, and the PLuTo Min/Max-fuse column data are the same metric achieved by TKET on the transformed circuits.

Table 2. Circuit Depth Achieved by TKET on Original and Transformed Circuits.

QASM	No. Input gates	TKET	PLuTo Min	PLuTo Max	Improvement
0D1_.8D2_1	360	524	480	504	1.09x, 1.04x
1D1_.7D2_9	405	524	381	381	1.51x
2D1_.5D2_8	405	400	348	348	1.15x
3D1_.3D2_4	405	280	240	240	1.16x
4D1_.3D2_8	495	256	176	176	1.45x
5D1_.2D2_2	540	182	160	160	1.13x

Though the applied transformations are designed for the optimization of classical computations and have yet to be tuned to the characteristics of quantum programs and devices, we still observe significant circuit depth reductions relative to circuit sizes. Scheduling QASM operations on a lifted representation of the program permits the reduction of circuit depth by exposing more relevant structures and parallelism opportunities to the back-end compiler.

Lifting to Intermediate Representations. Lifting and delinearization has been applied to several domains and for a variety of purposes. Ketterlin and Clauss [38] proposed the reconstruction of loop nests from data traces using a stack of terms to achieve compression, a similar goal to QRANE's. Verdoolaege and Grosser tackled the problem of recovering polyhedral abstractions from a LLVM's IR, while Holewinski et al. [29] devised methods to detect hidden vectorization potential in dynamic program traces. Grosser et al. [23] addressed the issue delinearizing parametric arrays to obtain their shape and dimensions in legacy codes and high-level programming languages like Julia. Nearly at the same time, Mendis et al. proposed a *lifting* scheme to extract Halide kernels [48] from x86 binaries [44]. Rodriguez et al. [50] leveraged an affine framework based on traversal of a tree search space. Kamil et al. [37] used program synthesis to lift stencil computations expressed in Fortran to a higher-level predicate language. Hasabnis and Sekar [27] automated the extraction of semantics required for lifting assembly to higher IR by learning from the implicit semantics of instruction sets and code generators. Augustine et al. [2] tackled the reconstruction of dense affine codelets from small sparse structures, yielding code customized to the underlying sparsity pattern, while Selva et al. [51] focused on extracting the affine IR of non-fully affine dynamic traces via a *folding-based analysis*.

3 Background

In this section we briefly recap the main concepts pertaining to quantum computing, their representation and the polyhedral abstractions we aim to reconstruct.

3.1 Quantum Computing (QC)

Quantum Bits and Gates. The basic unit of information in QC is the *qubit*. A qubit is commonly represented as a two-vector of complex probability amplitudes related to the probability that the qubit will collapse to the computational basis state of $|0\rangle$ or $|1\rangle$ upon measurement. Multi-qubit states are represented as the tensor product of all constituent qubit state vectors, so the resultant state vector $|\psi\rangle$ has dimension 2^n, where n is the number of qubits [47].

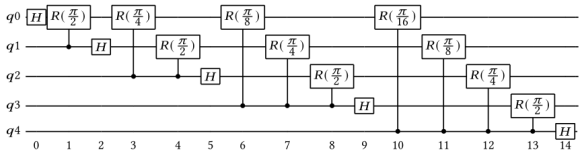
Quantum gates, unitary operations modeled as $2^m \times 2^m$ matrices, where m is the number of qubits the gate acts upon. While theoretical quantum gates can be designed to act on any number of qubits, in practice, all gates are decomposed into sets of "basis" gates that are physically implemented in hardware.

Quantum Circuits. The quantum circuit formulation is a popular way of modeling quantum computations. A quantum circuit allocates qubits into one or more one-dimensional *register* accessible by index expressions. The states of individual qubits are modeled as horizontal lanes, with gates applied to one or more qubits appearing either on the affected lane or spanning several of them. Figure 2 shows the Quantum Fourier Transform (QFT) [32] circuit and its accompanying OpenQASM code. We note that, while using multiple registers is possible in a quantum program, our work aims to produce a single one-dimensional register.

Multi-qubit gates are typically extensions of single-qubit gates in that they take a *control* argument and a *target* argument. If the control qubit is in the $|1\rangle$ state, then the corresponding single-qubit gate is applied. For example, the first "cp" gate shown in Figure 2 inverts the state of the target qubit "q[0]" if the control qubit "q[1]" is in state $|1\rangle$. From here on we assume the input to QRANE is given in the OpenQASM 2.0 format [14] format, and that it's free of dynamic

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[5];
...
h q[2];
cp(pi/8) q[3], q[0];
cp(pi/4) q[3], q[1];
cp(pi/2) q[3], q[2];
h q[3];
...
```

(a) OpenQASM Code Snippet



(b) Circuit Diagram

Figure 2. Quantum Fourier Transform (QFT) on Five Qubits with OpenQASM Code

control dependencies. For a more complete background, we refer the reader to Nielsen and Chuang's work [47].

3.2 Affine Abstractions

We represent quantum circuits in a polyhedral IR by grouping quantum operations (gates) into *iteration domains* and *access relations*. We don't attempt to delinearize the *program schedule* and defer this step to later scheduling steps. Further, in line with common practice, *dependence polyhedra* are later extracted using these three abstractions.

Iteration Domains are sets in $\mathbb{Z}^n$ whose points represent the dynamic instances of multi-dimensional loops. Loop bounds are affine functions of outer loop iterators and symbolic (constant) parameters. In the case of quantum gate streams, we assume an underlying stream generator takes an affine loop nest form, wherein quantum register accesses are affine functions of the loop iteration variables. Thus, we can group together multiple OpenQASM operations whose corresponding register accesses appear to be governed by the same affine functions. In the QFT example above, our proposed tool groups all "cp" operations into a single integer set defined by an outer loop variable i ranging from zero to three, and inner loop variable j ranging from zero to i. Thus, each (i, j) pair represents one of the ten "cp" operations.

Access Relations are functions of the iteration domain, $F^S : \mathbb{Z}^d \mapsto \mathbb{N}$, used here to model quantum register access expressions. Since control qubits are unaltered by multi-qubit gates and target qubits can undergo state change, we define *read* relations and *write* relations as a mapping of iteration domain points to control and target qubit accesses, respectively. The control and target access sequences of the "cp" operations could be generated by affine expressions *i+1* and *j*, respectively, given the previously described iteration domain on the *i* and *j* loop variables.

Schedules are affine relations that assign logical multi-dimensional time-stamps to iteration domain points. By default QRANE generates a single one-dimensional linear schedule mapping each point of an iteration domain to the global ordering of the operation it represents, thus preserving the original semantics. For example, the first "cx" operation in the QFT circuit would be mapped to the one-tuple "[1]", as it is the second operation in the circuit. Furthermore, schedules may be transformed from one time-space to another to achieve, for example, loop fusion, thereby changing the order of quantum operations. For a more complete background discussion, we refer the reader to Bastoul's work [3] and to Girbal et al. [19].

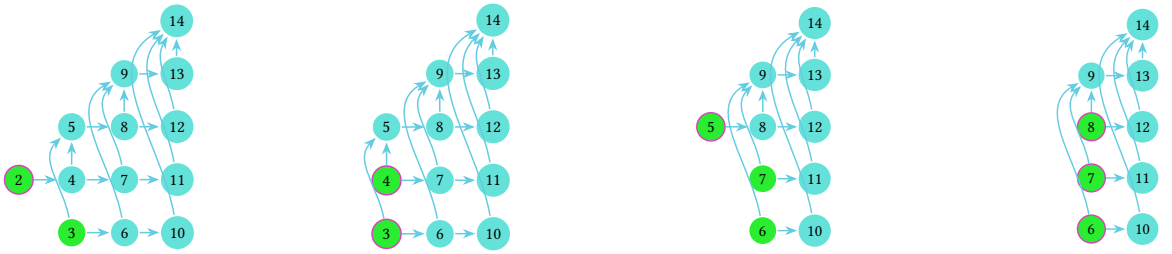
4 Overview

We first provide a walk-through of our approach to the delinearization of quantum gate streams. The Quantum Fourier Transform (QFT) circuit on five qubits [10], shown in Figure 2, serves as our motivating example. Walking through each step of QRANE's reconstruction process, we outline the techniques used in the one-dimensional and N-dimensional reconstruction phases, describe the formation of access relations, and discuss potential scheduling options.

4.1 1D Domains and Access Relations

The first step is to generate one-dimensional iteration domains. That is, QRANE will first look for gate sequences whose qubit register accesses *could* have been generated by a single affine loop. The immediate observation is that the number of points in any one-dimensional iteration domain must be less than or equal to the size of the qubit register. Thus, QRANE (mostly) limits its view of the circuit to the set of gates that most recently accessed each qubit in the register. We refer to this set of gates as the circuit "frontier", which is equivalent to the set of vertices in the time-space dependence graph of in-degree zero. At each time-step in this phase, QRANE examines the frontier of the circuit and attempts to find an affine function that generates the largest gate access sequence possible. The gates covered by the chosen function are then removed from the circuit, and this process is repeated until the circuit is empty.

Figure 3 shows the evolution of the 1D reconstruction over the dependence graph of QFT-5. The first and second time-steps are elided, because the dependence frontier is occupied by only one gate. QRANE considers these points as "stray points" as they don't follow the greater dependence pattern evident in the circuit; singleton domains are created to house these points. The vertices representing gates two and three enter the frontier in time-step three. Since domains may only cover one gate type, a decision must be made to draw a domain from gate two (H) or three (CP).



(a) Time-step 3 **(b)** Time-step 4 **(c)** Time-step 5 **(d)** Time-step 6

Figure 3. Evolution of 1D Reconstruction. Green represents Frontier, Red represents selected Domain.

QRANE computes the ratio of one to two-qubit gates neighboring the vertices of the frontier, and finding that ratio to be one, defaults to build a domain for one-qubit gates, putting gate two in a singleton domain.

```
       Domains                Read/Write Relations
# Structures repeat with unique
# SX identifier for H gates
SX[i] : 0 <= i <= 0;      SX[i] -> q[i];

# Structures representing each grouping of R gates.
 S1[i] : 0 <= i <= 0;      S1[i] -> q[i+1], q[i]
 S3[i] : 0 <= i <= 1;      S3[i] -> q[i+2], q[i]
 S6[i] : 0 <= i <= 2;      S6[i] -> q[i+3], q[i]
S10[i] : 0 <= i <= 3;     S10[i] -> q[i+4], q[i]
```

Figure 4. 1D Iteration Domains and Access Relations

```
          Extended Structures
# All H gates aggregated into one domain
S0[j, i] : 0 <= j <= 4 and i = 0;
S0[j, i] -> q[j];

# All R gates aggregates into one domain
S1[j, i] : 0 <= j <= 3 and 0 <= i <= j;
S1[j, i] -> q[j+1], q[0*j+i];
```

Figure 5. 2D Iteration Domains and Access Relations

From here on, QRANE exploits a consistent pattern where the frontier is occupied by a single H gate and a number of controlled phase gates. QRANE creates one singleton domain per H gate, and shuttles the CR gates occupying the frontier into their own domain. In time-step 4, we have R_3 and R_4 now in the frontier, giving QRANE the opportunity to test for linearity amongst both qubit access arguments of the two gates. For two-qubit gates, QRANE considers the control and target qubits to serve as coordinates in a 2D coordinate space, immediately revealing linearity between these points. The read/write access relations $[i] \rightarrow q[2]$, $q[i]$ are constructed based on the positive stride of 1 for the target qubit accesses.

The above pattern is repeated until the dependence graph is empty, yielding the iteration domains and access relations shown in Figure 4.

4.2 Raising Dimensionality through Merging

We can increase the dimensionality and improve the efficiency of our affine abstractions by repeatedly "merging" lower-dimensional structures into single structures that characterize N-D loop nests. A merger of two domains requires that we introduce a new linear dimension into the structure that serves as the "outer-most" loop variable, while the existing dimensions of the constituent domains are preserved as the "inner-most" loop variables. This step allows QRANE to represent sequences of d-dimensional domains as a single linearly consistent system of $d+1$ dimensions.

Continuing with our QFT example, QRANE recognizes that the 4 R gate sequences can be reproduced by a system of 2 linear variables, j and i, where j controls the number of R gate sequences, and i controls the number of R gates occurring in each sequence. We can also reproduce the qubit access sequences by varying the control accesses by a linear function of j and the target accesses by a linear function of i. That is to say, the control qubit in each sequence takes the value of $j+1$ with $0 \leq j \leq 3$, and the target qubit is equivalent to the value of i.

At this point, QRANE amends the singleton domains for each H gate by introducing an outer loop variable j (distinct from that of the R gate system) ranging from zero to four. The extra linear variable in this system is a byproduct of the dependence graph-based reconstruction, but the compression effect is identical to that of a single domain on one variable. At this point no further merges are possible, so QRANE completes the reconstruction phase. The resulting N-Dimensional affine abstractions are depicted in Figure 5.

4.3 Scheduling

The schedule maps the points of each iteration domain to some multi-dimensional time coordinate composed of

both scalar and linear dimensions. By default, QRANE constructs the *implicit* schedule for the program, which simply maps the points of each iteration domain to the original 1D time-space of the program, as shown in 6. However, QRANE may invoke the scheduling capabilities of ISL to produce legal transformations that re-order the quantum operations to according to some scheduling heuristic. For example, QRANE can leverage the *minfuse* and *maxfuse* heuristics of the PLuTo scheduling algorithm in attempt to minimally or maximally fuse the loops encoded in our affine abstractions. The re-ordering produced with the *maxfuse* option is also shown Figure 6. This schedule interleaves the points of each domain to minimize all dependence distances in the time-space. The effect of these schedule transformations on circuit compilation metrics is investigated in Section 6.

```
              Implicit Schedule
S0[0, 0] -> [0]; S0[1, 0] -> [5];
S0[2, 0] -> [9]; S0[3, 0] -> [12];
S0[4, 0] -> [14]; S1[3,3] -> [13];
S1[j, 0] -> [1+j] : 0 <= i0 <= 3;
S1[j, 1] -> [5+j] : 0 < i0 <= 3;
S1[j, 2] -> [8+j] : 2 <= i0 <= 3;

            PLuTo Maxfuse Schedule
S0[j, 0] -> [j, j, 0] : 0 <= j <= 4;
S1[j, i] -> [j, i, 1] : j <= 3 and 0 <= i <= j;
```

Figure 6. Schedule Examples for QFT-5

5 Recovering

We now present our methodology for delinearizing OpenQASM code into polyhedral abstractions. The lifting is divided into 4 stages: dependence analysis, 1D and N-D domain recovery, and scheduling. Firstly, since we deal with straight-line code, we assign each quantum operation (Q-OP) a *global-id* corresponding to its sequence number in the program. We refer to this bijection of $\{global\ id \rightarrow \text{Q-OP}\}$ as the *time space* of the original program. We also maintain a separate bijection $\{global\ id \rightarrow domain\ instance\}$, hereafter called the *domain space*, so that each Q-OP in the program is associated with a specific point in some iteration domain. The range of the relation is initially empty and is continuously updated throughout the reconstruction process to reflect set membership of reconstructed iteration domains. Our reconstruction requires all gates on 3 or more qubits to have been previously decomposed into equivalent one and two-qubit gates. In addition, we defer to future work the handling of dynamic control dependencies.

5.1 Auxiliary Dependence Analysis

We first perform a quick dependence analysis pass on the OpenQASM stream. As in [29], the DAG creation process from a qubit access trace is straightforward. We annotate each qubit with the global-id of the gate that most recently wrote to it, and draw dependencies between all subsequent gates reading from and writing to that qubit. The annotation is then updated to reflect the latest write. The result is a set of point-to-point dependencies in the original time space of the input sequence. This simplified dependence representation is necessary for the subsequent components of QRANE.

5.2 One-Dimensional Domain Recovery

Many prior works in trace analysis process memory access streams of each unique instruction, attempting to find minimal affine loop nests capable of reconstructing the access patterns. In the context of OpenQASM programs, we have very little auxiliary information to differentiate sets of quantum operations as belonging to the same or separate generating instructions. Thus, we can set rules for operation grouping that provide precise and correctness-preserving limitations on the reconstruction.

Firstly, we limit each loop statement to issue accesses of a single gate type. Secondly, as described in Sec. 4.1, QRANE asserts linear relationships among quantum gates based on their qubit access patterns. In the 1D recovery stage then, QRANE attempts to draw these relations between the set of gates that have most recently accessed each element of the qubit register. This set of gates is equivalent to the "frontier" of the dependence graph, or those vertices that have zero in-degree. QRANE iteratively fetches the frontier of the dependence graph, finds an affine function of one variable that covers a large number of points in the current time-step, and removes the covered gates off the dependence graph while updating the *domain space* to reflect the new gate grouping. The removal operation cause an in-degree reduction for all neighbors of the covered set, of which some will enter the frontier. Alg. 1 performs these steps.

Algorithm 1: Construct One-Dimensional Domains

Input: G: Data Dependence Graph
Output: D: Mapping of unique ID to iteration domain

```
1  Let domain_count = 0
2  Let d = 1
3  while G not empty do
4      F = copy_frontier (G)
5      Q1, Q2 = separate_by_num_args (F)
6      DOM = selection_policy (G, d, Q1, Q2)
7      G.remove (DOM.points)
8      D[domain_count] = DOM
9      domain_count += 1
10 return D
```

Once the dependence frontier is retrieved, we split the nodes into sets of one ($Q1$) and two ($Q2$) qubit gate operations per our single gate type criterion. While these sets are further subdivided by gate type, only the most frequently

occurring gate type is chosen to comprise *Q1* and *Q2*. QRANE must now decide from which of these sets should a domain be constructed? While decisions are correct with only a view of the frontier, QRANE may optionally perform a lookahead in the dependence graph up to breadth d (default two), computing the ratio r of one to two qubit gates among the immediate d neighbors of each vertex in the frontier. The rationale of this heuristic is that if more two-qubit gates are expected in the next time-step, then QRANE ought to draw a domain over *Q1* to make space in the frontier for more two-qubit gates (and vice-versa).

5.3 Finding Affine Qubit Access Functions

QRANE proceeds to sort the list of qops (either *Q1* or *Q2*) from the frontier by their access arguments. We construct a directed graph over the list of sorted qops, where each qop is connected to all qops coming after it in the list. A directed edge (q_1, q_2) between two qops is annotated with the slope between all arguments of q_2 and q_1, as well as their lexicographic ordering (positive or negative) in the time-space of the program. In the case of two qubit gates, we first find the largest co-linear subset from the *Q2* set by considering the control and target qubits as coordinates in a 2D space in order to reveal point groupings that could potentially exhibit linearity. The idea is that the longest path of constant *annotation* in this graph is the set of qubits reachable by a consistent linear stride while maintaining consistent lexicographic order. Qubit access sequences must be both produced by some affine function, *and* maintain a consistent lexicographic ordering, lest a valid affine schedule for the sequences be impossible.

We perform a depth-first search on each vertex in the topological ordering of G, shown in lines six and seven of Algorithm 2, establishing a baseline annotation on the first edge of each path and only extending a path when the annotations are equal to the baseline. The stride of the longest, constant annotation path found in that search is selected as the stride of the affine function covering the chosen domain from the frontier.

5.4 Access Relations

A key product of the one-dimensional reconstruction phase is an "access matrix" that simply encodes all of the register accesses made by a domain. Its form is shown in the matrix below:

$$\begin{bmatrix} i_{1,1} & \dots & i_{1,m} & 1 & a_{1,1} & 1 & a_{1,2} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ i_{n,1} & \dots & i_{n,m} & 1 & a_{n,1} & 1 & a_{n,2} \end{bmatrix} \tag{1}$$

This matrix encodes a system of n quantum operations whose register arguments *could* be generated by affine function of m variables. Thus, each point in the domain occupies one row of the matrix, and the values in the first m columns of each row represent the integer values of each loop iteration variables corresponding to that point in the domain. The two fixed column vectors of constant value one represent the constant term of the affine function potentially generating the access sequence. The columns $a_{n_i,1}$ and $a_{n_i,2}$ take the value of the control and target qubits for each operation, respectively. The matrix in Equation 1 thus models a two-qubit gate system, and a domain of one-qubit gates would have just one constant column and one arguments column. The solution to this system of equations then gives affine access relations for each point in the domain, allowing us to generate *Read* and *Write*-relations mapping each point in a domain to a multi-dimensional affine access to the qubit register. This "access matrix" form is utilized in Section 5.5 when verifying the linear consistency of a potential higher-dimensional domain merger.

Algorithm 2: One-Dimensional Domain Selection

```
Input: Q: Set of points from G.frontier
Output: Selected Domain
1 Let Q' = ∅ if Q.is_two_qubit_gate_set then
2 |   Q' = largest_colinear_subset (Q)
3 else
4 |   Q' = Q
5 Let P = ∅
6 foreach vertex v ∈ topological_ordering (G) do
7 |   Perform Depth-First Search with source v, recording
  |     annotation a' of each outgoing edge, and recursing
  |     on paths with annotations equal to a'
8 |   Store longest such path in P
9 return construct_domain (max ((P)))
```

5.5 N-Dimensional Domain Reconstruction

Once all gate accesses are modeled by one-dimensional affine abstractions, QRANE iteratively merges lower-dimensional domains into higher-dimensional ones by augmenting structures with new iteration variables, as per Alg.3. QRANE will examine a set of d-dimensional domains in effort to detect repeated or similar register access patterns by equivalent gate types. If the access patterns made by multiple domains can be generated by a consistent *(d+1)*-dimensional affine loop nest, QRANE will "merge" the domains into a single *(d+1)*-dimensional domain. This step is necessary to improve the expressiveness and efficiency of our abstractions, and to enhance the quality of information available when computing schedule transformations.

We first describe the conditions that must be met for two d-dimensional domains, A and B, to be merged into a single higher-dimensional domain. These are higher-dimensional generalizations of the rules described in Sections 5.2 and 5.3:

***A* and *B* have equivalent gate types.** This is a requirement to maintain the gate type consistency stipulation from Section 5.2

***A* and *B* are lexicographically consistent.** Just as the points of any one-dimensional must be lexicographically consistent relative to one another, the lexicographic ordering of the domains must match the original lexicographic ordering of the lower-dimensional domains. That is, if A and B are lexicographically positive themselves, then all points in B must be lexicographically positive relative to all points in A.

***A* and *B* form a consistent system of *(d+1)* dimensions.** To test for consistency of the "new" system, QRANE *concatenates* the access matrices of A and B and *prepends* a column representing the *d+1* loop variable. This new matrix is then reduced to test for consistency. For example:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 3 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 3 \end{bmatrix}$$

A and B are the access matrices of domains that represent two single-qubit operations accessing qubits $(0, 2)$, and $(1, 3)$, respectively. The two-dimensional system represented by matrix C is consistent, and thus A and B are candidates for merging.

Algorithm 3: Construct n-Dimensional Domains

Input: D: 1D iteration domains, l: Search Limit
Output: D': ND iteration domains

```
Let D' =
while D not empty do
    Let W =
    C = {D × D where D_i D_j is linearly consistent and
      j − i >= l}
    while C not empty do
        d = max_cardinality (C)
        W.insert (d)
        C.erase (x_i ∈ d)
    D' = D - W
    D = W
return D'
```

QRANE searches the set of lower-dimensional domains, admitting any groupings of domains that meet the three previously described conditions as *candidate* higher-dimensional domains. Since some OpenQASM circuits may be large and highly irregular, a large number of lower-dimensional domains may be produced; QRANE searches for candidate operations up to a user-defined *search limit*, offering the user a trade-off between execution time and the quality of the reconstruction.

QRANE selects domains from this list to proceed to the next iteration of the reconstruction. While the candidate list is not empty, QRANE picks the largest domain by cardinality from the list and removes all other domains that reference the now covered set of lower-dimensional domains. At the end of this process, QRANE has a set D of d-dimensional domains that were not merged with any other domains, and a set D' of selected *(d+1)*-dimensional domains. D is offloaded to a set of *unchanged domains*, and D' becomes the focus of the search for the next iteration of the N-Dimensional reconstruction process. The process completes when QRANE makes no further mergers at a given level.

5.6 Scheduling

As identity schedule, QRANE utilizes the implicit 1D linear schedule representing the operation number of a given iteration domain point. We thus defer the task of computing a more compact schedule to ISL. This doesn't pose any scalability issue, specially when leveraging our parallel reconstruction feature. All affine transformations available in ISL can then be used, in particular the Pluto minfuse (nofuse) and maxfuse loop fusion heuristics [6, 61], which we evaluate in Sec.6.

5.7 Parallel Lifting

QRANE leverages the *implicit* one-dimensional linear schedule (i.e. operation number) to enable the parallel reconstruction of the affine IR. This is achieved by decomposing the input QASM into N disjoint chunks, each of which can be lifted in parallel. Optionally, the user may insert OpenQASM "barrier" statements into circuit, forcing QRANE to independently delinearize and schedule the sub-circuits delineated by the barriers. While this has the obvious advantage of accelerating the reconstruction process, it also has the (unforeseen) benefit of allowing us to use larger values for our search parameters, thereby improving QRANE's reconstruction quality. In addition, we introduce in the implicit schedule a leading scalar dimension representing the chunk number, converting it to a schedule with two time-components. Such an additional schedule dimension yields semantically equivalent results when passed to ISL's dependence analysis, but the structural representation of these dependencies might be different and has the potential to impact later scheduling decisions, e.g., by splitting the index space into multiple pieces a scheduler can assign independent linear schedules for each piece. A comparison of the *parallel vs. sequential reconstruction* is included in the Appendix, while results presented in Sec. 6 focus exclusively on the parallel lifting.

5.8 Verification

The user may request that QRANE run its verification passes to ensure correctness of the reconstruction and scheduling. QRANE invokes ISL's code generation utilities to produce C code capable of generating OpenQASM from its affine abstractions. The resulting OpenQASM stream is sent through QRANE's dependence analysis in order to verify the results.

QRANE performs four checks. First, it checks that all Q-OPS are accounted for in the affine abstractions by invoking

the Barvinok Integer Set Counting library [64] to count the points of the iteration domain produced for the second. Second, QRANE checks for lexicographic positivity in the minimum delta of the reordered and original dependencies. A lexicographically negative delta implies that one or more original dependencies have been violated in the reordered OpenQASM. Next, QRANE tallies up the number of accesses to each qubit by each gate type in the original program, and checks that the same tallies of the reordered program are exact, ensuring that no operations were left out of the reconstruction and all access relations are correct. Finally, QRANE utilizes the NetworkX [25] implementation of the VF2 [11] algorithm to check for semantic isomorphism between the reordered and original dependence graphs. That is, each vertex in the dependence graph is labeled with the full quantum operation it represents, and the isomorphism check only admits a vertex mapping if their labels are identical.

6 Evaluation

In this section, we apply QRANE to three large OpenQASM benchmark sets to evaluate the effectiveness of the reconstruction and the optimization potential provided by the recovered affine abstractions. Section 6.2 examines the reconstruction quality in terms of recovered domain dimensionality and density. Section 6.3 examines QRANE's scalability. Finally, Section 6.4 explores the impact of two affine schedule transformations on circuit compilation metrics. QRANE is available on GitHub [17].

6.1 Experimental Testbed

We evaluate QRANE on three benchmark sets, **QUEKO** by Tan and Cong [57], IBMQX [33], a large resource of quantum circuits gathered by Zulehner et al. from RevLib [49, 65] and prior works, and **QASMBench** by Li et al. [41] at Pacific Northwest National Lab.

QUEKO is a set of quantum circuits designed by construction with known optimal depth. We use two circuit sub-classes of **QUEKO**, the **BIGD** set and the **BSS** set. The former consists of small circuits with varying gate densities (proportions of 1-qubit and 2-qubit gates), while the latter was designed for *scaling studies*, with ten sub-classes of circuits with optimal depth values and ten variations within each class. **IBMQX** [68] consists of 158 circuits premapped to legacy IBMQX architectures using between 5 and 16 qubits. These circuits vary dramatically in gate counts, ranging from five to half a million gates. Since these circuits were targeted for smaller architectures and all follow a similar mapped structure, the circuits tend to be very narrow. **QASMBench** provides several low-level implementations of quantum algorithms and subroutines at the core of chemistry, optimization, machine learning, and more. Across the small, medium, and large categories, the circuits exhibit a variety of widths (qubits), depths, gate densities, and other useful metrics. We chose to focus on larger circuits for this evaluation, as our results for the small and medium classes were very similar to those of the IBMQX benchmarks; large circuits reveal interesting qualities of QRANE's behavior with respect to the relationship between circuit width, depth, and gate density. All QASMBench circuits were decomposed several times in QISKIT such that all gates were one and two-qubit decompositions and all gate blocks were inlined.

We utilize Tket version 0.13 to collect compilation metrics. We use Barvinok library version 0.41.4 and ISL 0.23 for operations on integer sets and maps, as well as for the latter's scheduling capabilities. All experiments were run on an AMD Ryzen Threadripper 3970x 32-Core CPU operating at 4.5GHz. and equipped with 128 GB RAM.

6.2 Reconstruction Coverage

To assess the quality of QRANE's reconstruction, we focus on the QUEKO-BSS, IBMQX, and QASMBench-Large benchmarks because their larger circuits pose a greater compression challenge. Figures 7, 8, and 9 chart the "Reconstruction Profile" obtained with a search limit of 1024 and lookahead-breadth of two. The Reconstruction Profile contains, for all domain dimensions produced for a benchmark, all of the domain cardinalities recovered for those dimensions and the number of domains of those cardinalities. QRANE's primary goal is to compress as many Q-OPS into high density iteration domains as possible, relying on dimensionality extension to pack more points into a single domain while respecting the linearity of the system. While higher dimensions generally indicate greater compression, the objective of this test is to chiefly examine the density of domains produced by QRANE across the dimensionality spectrum. While QRANE has to shuttle many "stray points" into domains of 1, 2, and 3 points, in QUEKO-BSS, domains of cardinality four and above contain 2.59x as many points as contained in lower cardinality domains. QRANE performs even better on the IBMQX benchmark, assigning 2.99× times as many points into domains of cardinality four or higher than of lower-cardinality domains.

QRANE produces very dense lower-dimensional domains for the QASMBench-Large circuits. Most circuits involve between 16 to 27 qubits, along with the "ising_model" outliers on 500 and 1000 qubits. QRANE greatly benefits from a wider qubit register, since the density of one-dimensional abstractions is directly limited by the size of the register. If QRANE can shuttle more points into dense 1D domains, then it will have less work to do in merging irregular, heterogeneous domains produced over smaller registers, such as in IBMQX. Furthermore, QRANE prefers low depth, high qubit occupancy (gate density) circuits. Such circuits exhibit patterns where one and two-qubit gates are applied in parallel across the entire register, leading to fewer long gate paths in the dependence graph. The ising_model circuits are the perfect use cases for QRANE; these circuits have very wide

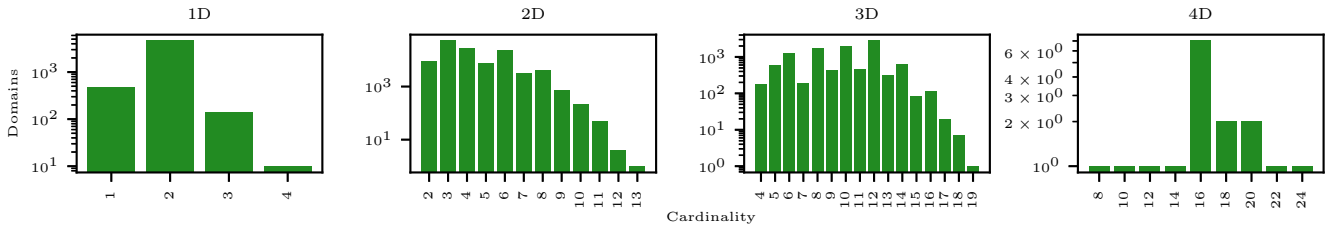


Figure 7. Reconstruction Quality for QUEKO-BSS-20QBT

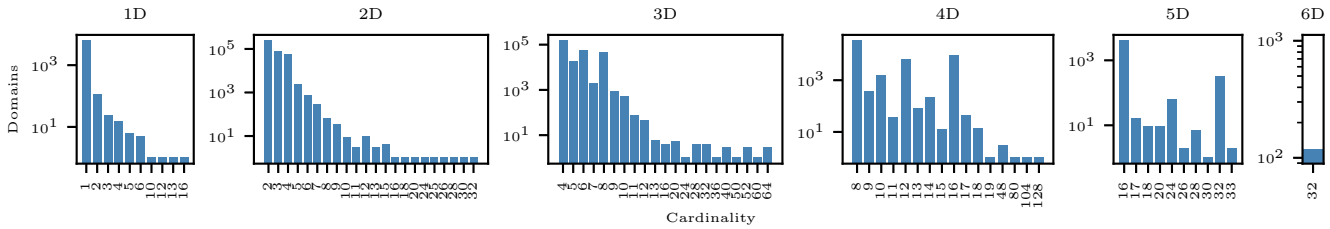


Figure 8. Reconstruction Quality for IBMQX

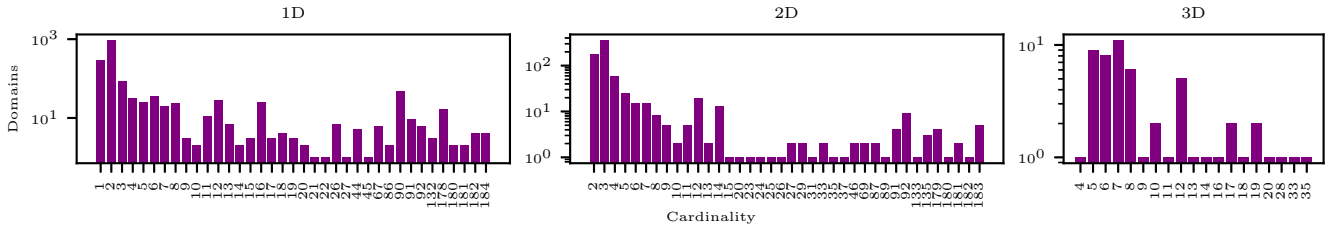


Figure 9. Reconstruction Quality for QASMBench-Large

registers, and are amongst the lowest depth and highest gate density circuits in the benchmark set.

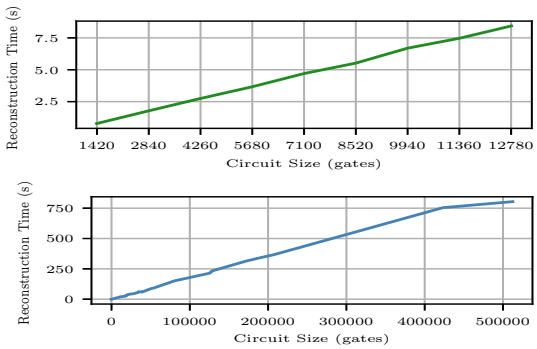


Figure 10. Sequential Subcircuit Reconstruction Scaling for QUEKO-BSS (top) and IBMQX (bottom)

6.3 Reconstruction Scaling

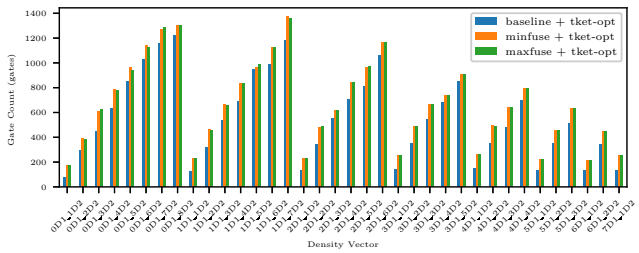
We briefly examine the scalability of QRANE's reconstruction. Figure 10 shows average reconstruction time over increasing circuit sizes. For QUEKO-BSS, we chunk all files into sub-circuits of 1000 operations, and impose no N-Dimensional search limit within each sub-circuit. For IBMQX, we also chunk all files into 1000 gate sub-circuits, but also impose a search limit of 512 domains. We do not here that, while the circuits are divided into sub-circuits, the execution time is indicative of *independent, sequential processing* of each sub-circuit. This was done to highlight the noticeable scalability benefits of decomposing the circuit into sub-circuits and performing independent reconstructions, all without sacrificing reconstruction quality. The addition of multi-threading directives, then, gives even faster results. These plots show that QRANE scales uniformly with increasing circuit size, making it an ideal fit for a larger compilation toolchain Furthermore, custom gate subroutines (marked by the "gate" qualifier in OpenQASM 2.0) can be delinearized just once, and the recovered abstractions simply reused for each occurrence of that subroutine in the circuit, further reducing reconstruction time. A further look into QRANE's scalability is provided in the Appendix.

6.4 Exploring Affine Transformations

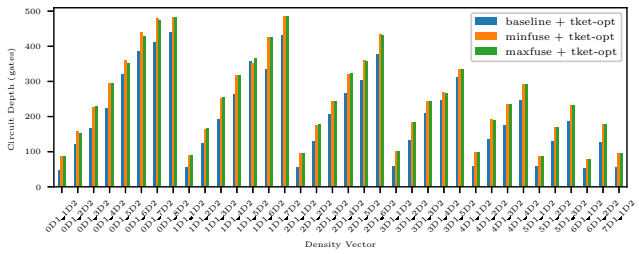
In this section, we evaluate the impact of two polyhedral schedule transformations methods — PLuTo Minfuse and PLuTo Maxfuse — on the Gate Count and Circuit Depth metrics after optimization and routing in Tket. Metrics are collected for the IBM Montreal (27 qubits) and IBM Brooklyn (65 qubits) architectures [31]. Here we have chosen the

QUEKO-BIGD set to examine the effect of these transformations on circuits with a variety of one and two-qubit gate densities (ratio of number of single-qubit gates to two-qubit gates). We remind the reader that these heuristics are from classical computing, and model classic data locality. Hence, results from Table 3 should be interpreted as an exploratory study of the two ends of the loop distribution/fusion spectrum. We do not intend to show immediate advantage, but to instead highlight the potential for future schedule transformations enabled by QRANE's reconstruction. The design of new scheduling heuristics with objectives leveraging quantum domain knowledge, similar in spirit to Shi et al's work [52], is left as future work enabled by QRANE.

Restructured QASM programs are obtained from building and executing the loop structure defined by a given affine transformation. Original and re-ordered circuits are then provided to TKET for compilation, and absolute gate count and circuit depth are collected. TKET has been configured to run NoiseAwarePlacement, FullPeephole Optimization, Routing, and Rebase to IBM gates [55].



(a) Absolute Gate Count



(b) Absolute Circuit Depth

Figure 11. Impact of Transformations on QUEKO-BIGD with Tket for IBM Brooklyn

We show in Table 3 the *baseline* metric (gate count and circuit depth) obtained on each architecture. We split the circuits into instances exhibiting improvement and deterioration. These are represented as *negative* (correspondingly, positive) variations over the baseline. One can observe then, overall, deterioration can range from 22% to 28% on both metrics, while improvement can range from 12% to 19%.

Zooming into the individual results, we observe that the worst deterioration takes place when the density of 1-qubit gates (d1) is greater than that of 2-qubit gates (d2). This is expected since 2-qubit gates tend to dominate the qubit mapping process due to them being the root cause for the insertion of SWAP operations. We observe that the Maxfuse heuristic tends to outperform Minfuse at low one-qubit gate densities, while this gap closes to minimal or no performance difference as one-qubit gate density increases.

Table 3. Mean Improvement/Deterioration of Transformations on Gate Count and Circuit Depth - QUEKO BIGD.

Result	TKET	Improvement		Deterioration	
Transformation	Gate Count (gates)				
	baseline	minfuse	maxfuse	minfuse	maxfuse
Montreal	516.32	-98.13 (40)	-88.75 (44)	117.78 (296)	118.81 (292)
Brooklyn	553.44	-74.92 (36)	-67.59 (39)	137.36 (300)	138.20 (297)
Transformation	Circuit Depth (gates)				
	baseline	minfuse	maxfuse	minfuse	maxfuse
Montreal	193.25	-30.12 (51)	-28.34 (50)	49.94 (285)	49.01 (286)
Brooklyn	204.93	-36.02 (45)	-33.52 (46)	58.67 (291)	58.10 (290)

We emphasize again that the Minfuse and Maxfuse heuristics are designed for classical program optimizations, and as such, are fairly agnostic to the quantum context in which we are working. These heuristics do not take into account the device architecture or hardware fidelity metrics, nor are they tuned to the common behaviors and structures of modern quantum programs. We suspect that in cases of improvement, the transformations were able to better reveal to the optimizer candidate patterns for replacement, while in cases of deterioration, these patterns were made harder to detect [13]. Nevertheless, we believe that future affine transformations utilizing QRANE as a starting point will be able to better expose commutativity relationships and connectivity patterns among dense two-qubit gate routines. The close gap in Tket's average metric performance between the original circuits and the re-ordered circuits, in spite of the generalized nature of the applied heuristics, foretells the potential for strong quantum-contextualized polyhedral transformations enabled by QRANE.

7 Conclusion

We presented the design and implementation of QRANE, the first polyhedral delinearizer that lifts quantum assembly into an affine IR. QRANE delinearizes qubit index expressions to build iteration domains, access relations and the identity schedule while preserving the semantics in a cost-effective fashion. Our results show the high-quality of the lifting process, the coverage, and the potential impact on large quantum circuits. The next logical step is to customize the reconstruction process considering topological knowledge, qubit physical parameters and tailoring affine transformations to exploit domain knowledge [1, 7, 40].

8 Data Availability Statement

QRANE's software artifact, data sets, installation and usage instructions are available as a Docker image [16].

References

[1] Khaled Abdelaal and Martin Kong. 2021. Tile Size Selection of Affine Programs for GPGPUs Using Polyhedral Cross-Compilation. In *Proceedings of the ACM International Conference on Supercomputing* (Virtual Event, USA) *(ICS '21)*. Association for Computing Machinery, New York, NY, USA, 13–26. https://doi.org/10.1145/3447818.3460369

[2] Travis Augustine, Janarthanan Sarma, Louis-Noël Pouchet, and Gabriel Rodríguez. 2019. Generating Piecewise-Regular Code from Irregular Structures. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 625–639. https://doi.org/10.1145/3314221.3314615

[3] Cedric Bastoul. 2004. Code generation in the polyhedral model is easier than you think. In *Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004.* 7–16. https://doi.org/10.1109/PACT.2004.1342537

[4] Debjyoti Bhattacharjee, Abdullah Ash Saki, Mahabubul Alam, Anupam Chattopadhyay, and Swaroop Ghosh. 2019. MUQUT: Multi-Constraint Quantum Circuit Mapping on Noisy Intermediate-Scale Quantum Computers. arXiv:1911.08559 [quant-ph]

[5] Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin Vechev. 2020. Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 286–300. https://doi.org/10.1145/3385412.3386007

[6] Uday Bondhugula, Albert Hartono, J. Ramanujam, and P. Sadayappan. 2008. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Tucson, AZ, USA) *(PLDI '08)*. ACM, New York, NY, USA, 101–113. https://doi.org/10.1145/1375581.1375595

[7] Lorenzo Chelini, Tobias Gysi, Tobias Grosser, Martin Kong, and Henk Corporaal. 2020. Automatic Generation of Multi-Objective Polyhedral Compiler Transformations. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques* (Virtual Event, GA, USA) *(PACT '20)*. Association for Computing Machinery, New York, NY, USA, 83–96. https://doi.org/10.1145/3410463.3414635

[8] Chun Chen, Jacqueline Chame, and Mary Hall. 2005. Combining Models and Guided Empirical Search to Optimize for Multiple Levels of the Memory Hierarchy. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO '05)*. IEEE Computer Society, Washington, DC, USA, 111–122. https://doi.org/10.1109/CGO.2005.10

[9] Andrew M Childs, Eddie Schoute, and Cem M Unsal. 2019. Circuit transformations for quantum architectures. *arXiv preprint arXiv:1902.09102* (2019).

[10] Patrick J. Coles, Stephan J. Eidenbenz, Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr M. Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo N. Djidjev, David Gunter, Satish Karra, Nathan Lemons, Shizeng Lin, Andrey Y. Lokhov, Alexander Malyzhenkov, David Dennis Lee Mascarenas, Susan M. Mniszewski, Balu Nadiga, Dan O'Malley, Diane Oyen, Lakshman Prasad, Randy Roberts, Philip Romero, Nandakishore Santhi, Nikolai Sinitsyn, Pieter Swart, Marc Vuffray, Jim Wendelberger, Boram Yoon, Richard J. Zamora, and Wei Zhu. 2018. Quantum Algorithm Implementations for Beginners. *CoRR* abs/1804.03719 (2018). arXiv:1804.03719 http://arxiv.org/abs/1804.03719

[11] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence* 26, 10 (2004), 1367–1372.

[12] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. 2019. On the Qubit Routing Problem. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135)*, Wim van Dam and Laura Mancinska (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 5:1–5:32. https://doi.org/10.4230/LIPIcs.TQC.2019.5

[13] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. 2020. Phase Gadget Synthesis for Shallow Circuits. *Electronic Proceedings in Theoretical Computer Science* 318 (May 2020), 213–228. https://doi.org/10.4204/eptcs.318.13

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

[15] Paul Feautrier. 1991. Dataflow analysis of array and scalar references. *International Journal of Parallel Programming* 20, 1 (01 Feb 1991), 23–53. https://doi.org/10.1007/BF01407931

[16] Blake Gerard. 2022. *BlakeGerard/qrane-artifact:*. https://doi.org/10.5281/zenodo.6333004

[17] Blake Gerard, Tobias Grosser, and Martin Kong. 2022. QRANE: Lifting QASM Programs to an Affine IR. https://github.com/BlakeGerard/qrane Online; accessed on March 2022.

[18] Blake Gerard and Martin Kong. 2021. Exploring Affine Abstractions for Qubit Mapping. In *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*. 43–54. https://doi.org/10.1109/QCS54837.2021.00009

[19] Sylvain Girbal, Nicolas Vasilache, Cédric Bastoul, Albert Cohen, David Parello, Marc Sigler, and Olivier Temam. 2006. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies. *International Journal of Parallel Programming* 34, 3 (2006), 261–317.

[20] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. 2013. Quipper: A Scalable Quantum Programming Language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. ACM, New York, NY, USA, 333–342. https://doi.org/10.1145/2491956.2462177

[21] Martin Griebl, Paul Feautrier, and Christian Lengauer. 2000. Index Set Splitting. *Int. J. Parallel Program.* 28, 6 (2000), 607–631. https://doi.org/10.1023/A:1007516818651

[22] Tobias Grosser, Armin Groesslinger, and Christian Lengauer. 2012. Polly—performing polyhedral optimizations on a low-level intermediate representation. *Parallel Processing Letters* 22, 04 (2012), 1250010.

[23] Tobias Grosser, Jagannathan Ramanujam, Louis-Noel Pouchet, Ponnuswamy Sadayappan, and Sebastian Pop. 2015. Optimistic delinearization of parametrically sized arrays. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. 351–360.

[24] Jingzhe Guo and Mingsheng Ying. 2020. Software Pipelining for Quantum Loop Programs. *arXiv preprint arXiv:2012.12700* (2020).

[25] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[26] P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

[27] Niranjan Hasabnis and R. Sekar. 2016. Lifting Assembly to Intermediate Representation: A Novel Approach Leveraging Compilers. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 311–324. https://doi.org/10.1145/2872362.2872380

[28] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R Brown, Diana Franklin, Frederic T Chong, and Margaret Martonosi. 2015. Compiler management of communication and parallelism for quantum computation. *ACM SIGARCH Computer Architecture News* 43, 1 (2015), 445–456.

[29] Justin Holewinski, Ragavendar Ramamurthi, Mahesh Ravishankar, Naznin Fauzia, Louis-Noël Pouchet, Atanas Rountev, and P. Sadayappan. 2012. Dynamic Trace-based Analysis of Vectorization Potential of Applications. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* (Beijing, China) *(PLDI '12)*. ACM, New York, NY, USA, 371–382. https://doi.org/10.1145/2254064.2254108

[30] IBM. 2018. Qiskit. https://qiskit.org

[31] IBM. 2021. IBM Quantum Services. https://quantum-computing.ibm.com/services

[32] IBM-Research, Alexandre Blais, Robin Blume Kohout, Francesco Buscemi, Jonas Bylander, Isaac Chuang, Vincent Dwyer, Mark Everitt, Michael Geller, Thomas Haener, Andrew Houck, Thomas Ihn, Sabre Kais, Miguel Ángel Martín-Delgado, Rod Van Meter, Harumichi Nishimura, Russell Rundle, Enrique Solano, Damian Steiger, Todd Tilma, Christophe Vuillot, Martin Weides, James Wootton, and Shigeru Yamashita. 2018. *Open Source Quantum Information Science Kit*. https://qiskit.org

[33] Johannes Kepler University Linz Institute for Integrated Circuits. 2019. *IIC JKU - IBMQX QASM Circuits*. https://github.com/iic-jku/ibm_qx_mapping/tree/master/examples Online; accessed on August 2021.

[34] F. Irigoin and R. Triolet. 1988. Supernode Partitioning. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '88)*. ACM, New York, NY, USA, 319–329. https://doi.org/10.1145/73560.73588

[35] Raban Iten, Romain Moyard, Tony Metger, David Sutter, and Stefan Woerner. 2020. Exact and practical pattern matching for quantum circuit optimization. arXiv:1909.05270 [quant-ph]

[36] Ali JavadiAbhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. 2012. *Scaffold: Quantum programming language*. Technical Report. PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE.

[37] Shoaib Kamil, Alvin Cheung, Shachar Itzhaky, and Armando Solar-Lezama. 2016. Verified Lifting of Stencil Computations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Santa Barbara, CA, USA) *(PLDI '16)*. ACM, New York, NY, USA, 711–726. https://doi.org/10.1145/2908080.2908117

[38] Alain Ketterlin and Philippe Clauss. 2008. Prediction and Trace Compression of Data Access Addresses through Nested Loop Recognition. In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization* (Boston, MA, USA) *(CGO '08)*. Association for Computing Machinery, New York, NY, USA, 94–103. https://doi.org/10.1145/1356058.1356071

[39] Martin Kong. 2021. On the Impact of Affine Loop Transformations in Qubit Allocation. *ACM Transactions on Quantum Computing* (2021). https://doi.org/10.1145/3465409

[40] Martin Kong and Louis-Noël Pouchet. 2019. Model-Driven Transformations for Multi- and Many-Core CPUs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 469–484. https://doi.org/10.1145/3314221.3314653

[41] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2020. Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation. *arXiv preprint arXiv:2005.13018* (2020).

[42] Gushu Li, Yufei Ding, and Yuan Xie. 2019. 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* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 1001–1014. https://doi.org/10.1145/3297858.3304023

[43] Dmitri Maslov, Sean M. Falconer, and Michele Mosca. 2008. Quantum Circuit Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 4 (2008), 752–763. https://doi.org/10.1109/TCAD.2008.917562

[44] Charith Mendis, Jeffrey Bosboom, Kevin Wu, Shoaib Kamil, Jonathan Ragan-Kelley, Sylvain Paris, Qin Zhao, and Saman Amarasinghe. 2015. Helium: Lifting High-performance Stencil Kernels from Stripped x86 Binaries to Halide DSL Code. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. ACM, New York, NY, USA, 391–402. https://doi.org/10.1145/2737924.2737974

[45] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. 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* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 1015–1029. https://doi.org/10.1145/3297858.3304075

[46] Prakash Murali, David C. Mckay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. 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* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 1001–1016. https://doi.org/10.1145/3373376.3378477

[47] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[48] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

[49] Revlib. 2021. *Revlib: An Online Resource for Reversible Functions and Circuits*. http://www.revlib.org/functions.php Online; accessed on August 2021.

[50] Gabriel Rodríguez, José M. Andión, Mahmut T. Kandemir, and Juan Touriño. 2016. Trace-based Affine Reconstruction of Codes. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization* (Barcelona, Spain) *(CGO '16)*. ACM, New York, NY, USA, 139–149. https://doi.org/10.1145/2854038.2854056

[51] Manuel Selva, Fabian Gruber, Diogo Sampaio, Christophe Guillon, Louis-Noël Pouchet, and Fabrice Rastello. 2019. Building a Polyhedral Representation from an Instrumented Execution: Making Dynamic Analyses of Nonaffine Programs Scalable. *ACM Trans. Archit. Code Optim.* 16, 4, Article 45 (Dec. 2019), 26 pages. https://doi.org/10.1145/3363785

[52] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Optimized compilation of aggregated instructions for realistic quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1031–1044.

[53] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2019. Qubit Allocation As a Combination of Subgraph Isomorphism and Token Swapping. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 120 (Oct. 2019), 29 pages. https://doi.org/10.1145/3360546

[54] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira. 2018. Qubit Allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization* (Vienna, Austria) *(CGO 2018)*. ACM, New York, NY, USA, 113–125. https://doi.org/10.1145/3168822

[55] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. tket: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (Nov 2020), 014003. https://doi.org/10.1088/2058-9565/ab8e92

[56] Bochen Tan and Jason Cong. 2020. Optimal Layout Synthesis for Quantum Computing. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 137, 9 pages. https://doi.org/10.1145/3400302.3415620

[57] Bochen Tan and Jason Cong. 2020. Optimality Study of Existing Quantum Computing Layout Synthesis Tools. *IEEE Trans. Comput.* (2020), 1–12. https://doi.org/10.1109/TC.2020.3009140

[58] Swamit S Tannu and Moinuddin Qureshi. 2019. Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 253–265.

[59] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *arXiv preprint arXiv:1802.04730* (2018).

[60] Anand Venkat, Mary Hall, and Michelle Strout. 2015. Loop and Data Transformations for Sparse Matrix Code. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Portland, OR, USA) *(PLDI '15)*. ACM, New York, NY, USA, 521–532. https://doi.org/10.1145/2737924.2738003

[61] Sven Verdoolaege. 2010. isl: An Integer Set Library for the Polyhedral Model.. In *ICMS*, Vol. 6327. Springer, 299–302.

[62] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, José Ignacio Gómez, Christian Tenllado, and Francky Catthoor. 2013. Polyhedral Parallel Code Generation for CUDA. *ACM Trans. Archit. Code Optim.* 9, 4, Article 54 (Jan. 2013), 23 pages. https://doi.org/10.1145/2400682.2400713

[63] Sven Verdoolaege and Tobias Grosser. 2012. Polyhedral extraction tool. In *Second International Workshop on Polyhedral Compilation Techniques (IMPACT'12), Paris, France*, Vol. 141.

[64] Sven Verdoolaege, Rachid Seghir, Kristof Beyls, Vincent Loechner, and Maurice Bruynooghe. 2007. Counting integer points in parametric polytopes using Barvinok's rational functions. *Algorithmica* 48, 1 (2007), 37–66.

[65] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. 2008. RevLib: An Online Resource for Reversible Functions and Reversible Circuits. In *Int'l Symp. on Multi-Valued Logic*. 220–225. RevLib is available at http://www.revlib.org.

[66] Xin-Chuan Wu, Marc Grau Davis, Frederic T Chong, and Costin Iancu. 2020. QGo: Scalable Quantum Circuit Optimization Using Automated Synthesis. *arXiv preprint arXiv:2012.09835* (2020).

[67] Xiangzhen Zhou, Sanjiang Li, and Yuan Feng. 2020. Quantum circuit transformation based on simulated annealing and heuristic search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 12 (2020), 4683–4694.

[68] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. Efficient mapping of quantum circuits to the IBM QX architectures. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1135–1138. https://doi.org/10.23919/DATE.2018.8342181