Technical note on

Compiling quantamorphisms for the IBM Q Experience

A. Neri, R.S. Barbosa and J.N. Oliveira

March 2021

This technical note addresses the poor scalability of explicitly using unitaries in the compilation process proposed in the paper [1]. Shortcutting such a step is acknowledged as a major topic for future research.

As a prospect of how such a challenge is to be addressed, this note shows how to solve it in the simpler case of quantum for-loops, that is, quantamorphisms with recursive pattern of shape $1 + X$ instead of $1 + A \times X$ required by list quantamorphisms.[1] Pattern $1 + X$ is simpler because it does not lead to a polymorphic container, as $1 + A \times X$ does on parameter A. Functional for-loops are implemented by the functional combinator,

$$
\begin{aligned}
&for\ f\ i\ 0 = i \\
&for\ f\ i\ (n+1) = f\ (for\ f\ i\ n)
\end{aligned}
$$

which iterates f as many times as required by the third parameter, which is a natural number. That is: $for\ f\ i\ n = f^n\ i$.

Treating this combinator in the same way as list folds in the current paper one reaches the monadic code:

$$
\begin{aligned}
&(\!|\cdot|\!) :: (Monad\ m) \Rightarrow (b \to m\ b) \to (Int, b) \to m\ (Int, b) \\
&(\!|f|\!)\ (0, b) = return\ (0, b) \\
&(\!|f|\!)\ (n+1, b) = \mathbf{do}\ \{ \\
&\quad b' \leftarrow f\ b; \\
&\quad (m, b'') \leftarrow (\!|f|\!)\ (n, b'); \\
&\quad return\ (m+1, b'') \\
&\quad \}
\end{aligned}
$$

The corresponding quantum counterpart is the recursive matrix definition

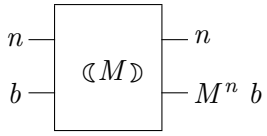
$$
\begin{array}{ccc}
\mathbb{N}_0 \times B & \xleftarrow{\Phi id} & B + \mathbb{N}_0 \times B \\
{\scriptstyle (\!|M|\!)}\downarrow & & \downarrow{\scriptstyle id \oplus (\!|M|\!)} \\
\mathbb{N}_0 \times B & \xleftarrow[\Phi M]{} & B + \mathbb{N}_0 \times B
\end{array}
$$

where $B \xleftarrow{M} B$ is the loop-body and $\Phi X = [\underline{0} \triangledown id \mid succ \otimes X]$. This unfolds to

$$
(\!|M|\!) \quad = \quad N \oplus (succ \otimes M) \cdot (\!|M|\!) \cdot (succ^{\circ} \otimes id)
$$

where the base case is $N = (\underline{0} \triangledown id) \cdot (\underline{0} \triangledown id)^{\circ}$. This is the "quantamorphism"

[1] For-loop quantamorphisms are addressed in [2].



implementing the quantum-*for* gate which iterates M over the b-input, controlled by the n-input (a natural number).

In practice, one is bound to implement an approximation of $⦇M⦈$ for $n<2^k$, where k is the number of control qubits available. In such finite approximations

$$M^n = \underbrace{M \cdot M \cdot ... \cdot M}_{n \leqslant 2^k - 1 \ times},$$

the M in the i-th position should be triggered for all inputs $n \geqslant i$. For instance, here follows the unitary of one such approximation — for-loops with up to 7 iteration steps ($k = 3$):

	B	B	B	B	B	B	B	B
B	id	0	0	0	0	0	0	0
B	0	M	0	0	0	0	0	0
B	0	0	M^2	0	0	0	0	0
B	0	0	0	M^3	0	0	0	0
B	0	0	0	0	M^4	0	0	0
B	0	0	0	0	0	M^5	0	0
B	0	0	0	0	0	0	M^6	0
B	0	0	0	0	0	0	0	M^7

So we need to implement the predicate $(\geqslant i)$ over numbers $n = q_{k-1} ... q_0$ expressed by k qubits, for any k and $0 \leqslant i \leqslant 2^k - 1$. We know that $n \geqslant 0$ always holds and that

$$n \geqslant i \Leftrightarrow (n = i) \vee (n \geqslant i + 1) \tag{1}$$

holds too. By Boolean logic (negation denoted by $\overline{a}$, $a \wedge b$ by juxtaposition and $a \vee b$ by $a + b$) simplification

$$b + \overline{b}a = b + a \tag{2}$$

holds. For $k = 3$ — three control qubits q_2, q_1, q_0 — we get, via (1) simplified by (2):

$$\begin{aligned}
(\geqslant 7) &= q_2\ q_1\ q_0 \\
(\geqslant 6) &= q_2\ q_1\ \overline{q_0} + q_2\ q_1\ q_0 = q_2\ q_1 \\
(\geqslant 5) &= q_2\ \overline{q_1}\ q_0 + q_2\ q_1 = q_2\ (q_1 + \overline{q_1}\ q_0) = q_2\ (q_1 + q_0) \\
(\geqslant 4) &= q_2\ \overline{q_1}\ \overline{q_0} + q_2\ (q_1 + \overline{q_1}\ q_0) = q_2\ q_1 + q_2\ \overline{q_1} = q_2 \\
(\geqslant 3) &= q_2 + \overline{q_2}\ q_1\ q_0 = q_2 + q_1\ q_0 \\
(\geqslant 2) &= \overline{q_2}\ q_1\ \overline{q_0} + \overline{q_2}\ q_1\ q_0 + q_2 = q_2 + \overline{q_2}\ q_1 = q_2 + q_1 \\
(\geqslant 1) &= q_2 + q_1 + \overline{q_2}\ \overline{q_1}\ q_0 = q_2 + q_1 + \overline{q_1}\ q_0 = q_2 + q_1 + q_0 \\
(\geqslant 0) &= 1
\end{aligned}$$

This suggests the following generalization, written in Haskell

$$(\geqslant_k\ i) = (f\ k)\ !!\ i$$

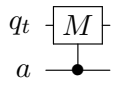
where $(\geqslant_k\ i)$ denotes the $n \geqslant i$ predicate on k-qubit control n at iteration i, and where f is inductive on k:

$$\begin{array}{l} f\ 0 = [1, q_0] \\ f\ (n+1) = [q_{n+1} + e \mid e \leftarrow f\ n] +\!\!+ [q_{n+1} e \mid e \leftarrow f\ n] \end{array} \tag{3}$$

For instance, as earlier on:

$$\begin{array}{l} (\geqslant_3\ 0) = 1 \\ (\geqslant_3\ 1) = q_2 + q_1 + q_0 \\ (\geqslant_3\ 2) = q_2 + q_1 \\ (\geqslant_3\ 3) = q_2 + q_1 q_0 \\ (\geqslant_3\ 4) = q_2 \\ (\geqslant_3\ 5) = q_2(q_1 + q_0) \\ (\geqslant_3\ 6) = q_2 q_1 \\ (\geqslant_3\ 7) = q_2 q_1 q_0 \end{array}$$

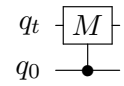
Figure 1 shows the evolution of the position controls as qubits are added, stopping at $(\geqslant_4)$ for space restrictions. Assume one already knows the circuit a which controls the for-loop body M at some given position with k control qubits. Denote this by expression $a \Rightarrow M$, that is, circuit:



Adding another control qubit q_{k+1} involves building $q_{k+1}\, a \Rightarrow M$ and $q_{k+1} + a \Rightarrow M$, as seen in figure 1.

As the overall goal is to create a quantum circuit parametric on the number of control qubits, this can be achieved via a recursive function that reads the sequence of control expressions produced by auxiliary function f (3) and converts these in control (sub)circuits. Let us see some examples:

- For $k \leqslant 0$, the circuit has only a target qubit where nothing happens.
- For $k = 1$, the circuit has one control and one target. This case has two gates and does not need ancillary qubits (4):



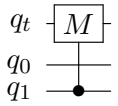
(4)

- If $k = 2$, the function "reuses" the operations of case $n = 1$. In particular, $n\ \leqslant_2\ 0$ is the same (the identity) and $n\ \geqslant_2\ 2$ is also equal if we consider

Figure 1: *Evolution of the position control circuits as more qubits are added. For space economy, q_0 ... q_3 are abbreviated to a ... d.*

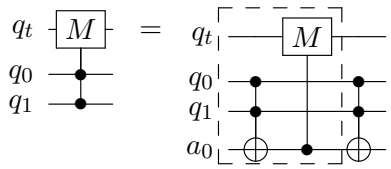
CTL-n=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 qubit	1	a	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2 qubits	1	b+a	b	ba	-	-	-	-	-	-	-	-	-	-	-	-
3 qubits	1	c+(b+a)	c+b	c+ba	c	c(b+a)	cb	cba	-	-	-	-	-	-	-	-
4 qubits	1	d+c+b+a	d+c+b	d+c+ba	d+c	d+c(b+a)	d+cb	d+cba	d	db(c+b+a)	d(c+b)	d(c+ba)	dc	dc(b+a)	dcb	dcba

the implementation of a CX gate, where the control is always the most significant qubit:



(5)

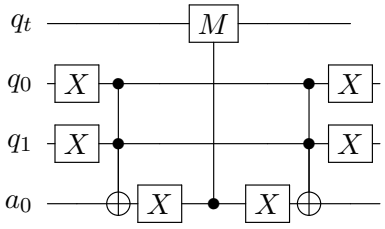
The other cases involve the AND and OR operations, with decompositions. For instance, $n \geqslant_2 3$ involves an AND operation,



(6)

see Figure 1. The gates inside the dotted box are responsible for implementing the AND gate and that outside ensures reversibility.

The circuit below corresponds to $n \geqslant_2 1$ and involves the 'OR' operation and ancillary qubit a_0:



(7)

- For $k \geqslant 3$ the function will reuse the previous sub-circuits. For instance, $(\geqslant_3\ 0)$ is the same and $(\geqslant_3\ 2)$, $(\geqslant_3\ 4)$ and $(\geqslant_3\ 6)$ only need a shift to use the most significant gates as controls — see figure 2 (a,b,c). To make the other equations simpler one can prioritise solving the right side of each equation. Let us see the example $(\geqslant_3 5)$, that is, $q_2q_1 + q_0$, where the priority is in the $+$ operation (OR). This gate is the first gate of the decomposition. Through recursion, the 'AND' gate follows. With no more operation to implement, it is possible to trigger the target and reverse the auxiliary gates (See figure 3).

The compositionality of this approach can be seen in the circuits generated by the Python script available from section *Improving the tool chain* of the project's website. In this site the example chosen instantiates M in our circuits to the S-gate and takes $k = 3$.

The evolution of this compilation scheme to that required by quantamorphisms over lists (studied in the current submission) is a natural one. Observe the recursive pattern already dealt with, $1 + X$ (for-loops), and the one still to be addressed, $1 + A \times X$ (lists). The former is a special case of the later, for $A = 1$ (up to isomorphism). The additional information that needs to be considered concerns the list of inputs of type A to the gate that is being iterated. We are addressing the implementation of this extension at the time of writing.

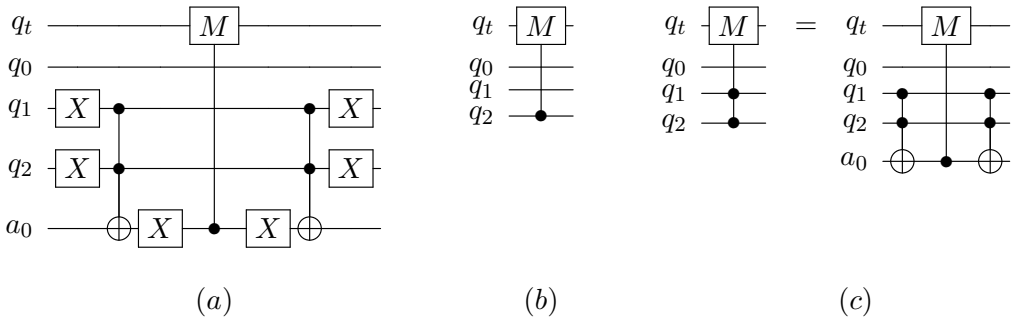


Figure 2: *Control circuits for $k = 3$: (a) OR gate in $(\geqslant_3 2)$; (b) ctr-M in $(\geqslant_3 4)$; (c) AND gate in $(\geqslant_3 6)$.*

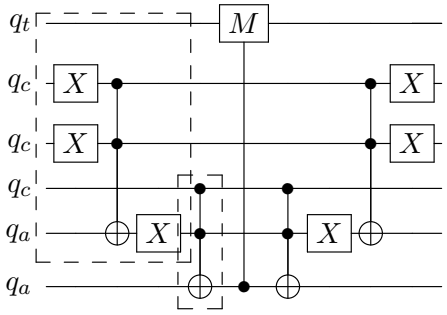


Figure 3: *Circuit for $(\geqslant_3 5)$. The first box captures the OR operation. The second box, which captures the AND, is followed by the gate control-M to trigger the target and finally by gates reversing the auxiliary ones.*

References

[1] A. Neri, R. Barbosa, and J. Oliveira, "Compiling quantamorphisms for the IBM Q-Experience," 2020, submitted to IEEE Trans. Soft. Eng.

[2] A. Neri, "Towards quantum program calculation," Master's thesis, University of Minho, Informatics Department, October 2018.