IBM FALL 2020 QUANTUM CHALLENGE
FINAL WEEK PROBLEM WRITE-UP

WITOLD JARNICKI

beit.tech

Abstract. I present a solution leading to a score of 6065 (5th place at the time of writing this). Additionally, I tell a brief history of attempted cost-lowering strategies.

1. A brief history of submissions

Table 1 presents a list of ideas introduced, along with their (approximate) score and a link to the section describing the details. All my solutions use a common approach described in Section 2. It is recommended that the reader familiarize themselves with that section as a prerequisite.

Idea name	Score	Section
CNF proof of concept	$\sim 3\,000\,000$	3
Board symmetry (later banned)	$\sim 170\,000$	4
Matching	$\sim 20\,000$	5
Weighted phase oracle	$\sim 7\,500$	6
Micro-optimizations	$\sim 6\,500$	7
Smart QRAM loading	6 065	8

Table 1. Ideas introduced into my solution throughout the contest.

2. The common computation model and notation

I made a very early decision to use a classical Boolean-logic oracle for each board. Before that I made some attempts at building a "true quantum" approach involving considering all shot possibilities, but I was always running out of qubits, so ultimately I gave up on it.

Each of my solutions follows the diagram on Figure 1.

The parts A (address ancilla preparation $|0\rangle|a\rangle \longmapsto |f(a)\rangle|a\rangle$), O (phase oracle), and D (the diffusion operator) must be independent of the input data.

The part Q (loading QRAM) can depend on data and must adhere to the rule $|f(a)\rangle|a\rangle|0\rangle|0\rangle \longmapsto |f(a)\rangle|a\rangle|g(a)\rangle|0\rangle$, where $g(a)$ is the unprocessed data of board a.

I have bent the last rule a little in one of my intermediate solutions, but I requested that the organizers withdraw it, once I learnt it was forbidden. For the final solution I have $s = 16$ and $g(a)$ is the exact bitmap of zeroes and ones, as read row-by-row from board a.

3. CNF proof of concept

The idea behind this solution is that being an unsolvable board can be expressed by brute force as a conjuction of 56 (the number of ways of aligning the shots) clauses, each consisting of an alternative of (4 or 6) qubits corresponding to the fields not covered by a particular shot combination.

Brief description of the circuit:

E-mail address: witek@beit.tech.

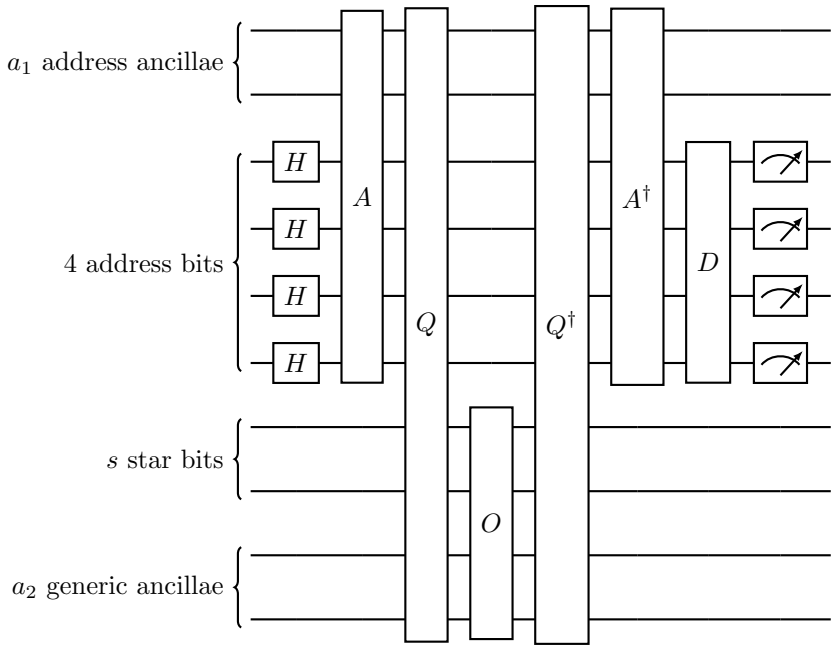

FIGURE 1. A circuit template used by all my solutions.

- $a_1 = 0$, $s = 16$, $a_2 = 8$.
- A does nothing.
- Q is a brute-force sequential data loader.
- O computes the CNF clauses ancilla-less and uses the ancillae to store the intermediate conjunctions. Data aggregation is performed by ancilla-less CCCCX. Finally, we flip the phase on the formula result and uncompute everything.
- D is the standard Grover diffusion operator.

4. Exploiting board symmetry

One can prove that for each 4×4 board with 6 chosen elements there exists a board isomorphism (consisting of permuting columns, permuting rows, and optionally flipping about the diagonal) such that 3 particular fields always end-up chosen (numbers 9, 14, and 15, to be exact). The solution preprocesses the boards so that that is the case. We can then significantly reduce the number of clauses, as some are guaranteed to evaluate to true.

The circuit is pretty much identical to the one from Section 3, except that it uses $s = 13$, has fewer clauses to compute, and uses the regained qubits as ancilla to speed up some of the ancilla-less operations.

5. Perfect matching

At some moment it occurred to me that one can greatly simplify the formula involved. One can observe that a board is unsolvable iff there exists a permutation $\sigma \in S_4$ such that there are stars at all the fields $(j, \sigma(j))$, $j = 1, \ldots, 4$.

This means that we can go from 56 clauses of 4–6 qubits down to 24 clauses of 4 qubits and still stay within the new rules that disallow reshaping the board. Additionally, for a chosen subset C of columns $\{1, 2, 3, 4\}$ we can efficiently compute in one pass the value of B_C defined as whether the board is certified unsolvable by one of the permutations from $\{\sigma \in S_4 : \sigma(1), \sigma(2) \in C \not\ni \sigma(3), \sigma(4)\}$.

Brief description of the circuit:

- $a_1 = 0$, $S = 16$, $a_2 = 8$.
- A does nothing.

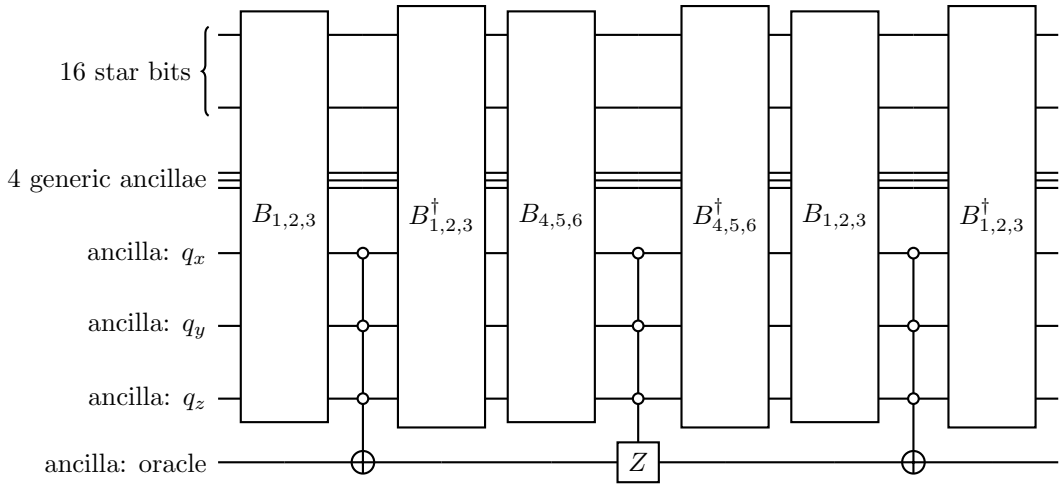

FIGURE 2. The circuit that flips phase on the alternative of $\{B_{C_1}, \ldots, B_{C_6}\}$. The part $B_{a,b,c} = B^\dagger_{a,b,c}$ computes B_{C_a}, B_{C_b}, and B_{C_c}, putting the result in q_x, q_y, and q_z, respectively.

- Q is a standard Gray-code data loader.
- O flips the phase on the alternative of efficiently-computed values of B_C, taken over the six two-element subsets of $\{1, 2, 3, 4\}$. The exact circuit is shown of Figure 2.
- D is the standard Grover diffusion operator.

6. Weighted phase oracle

Observe that a diffusion operator amplifies the amplitude of the marked element not only when its phase is completely flipped, but in fact in any case where its phase is somehow changed. Naturally, the effect is much stronger as the phase approaches a complete flip.

We can take advantage of that and significantly simplify the circuit from Section 5. As at most two out of B_Cs evaluate to true, we can change the phase by $2\pi/3$ for all the boards certified unsolvable by particular B_C. The solvable boards will stay in-phase, while the unsolvable one will be shifted by either $2\pi/3$ or $4\pi/3$ — in both cases sufficiently close to a complete flip for the amplitude to get amplified. Figure 3 shows the structure of a circuit P_C performing a single phase shift.

We now replace the oracle by consecutive calls to P_C for all six values of C.

7. Micro optimizations

There are several techniques that make the circuit shorter without affecting its correctness. This is the list of optimizations I performed, probably not exhaustive, as I am bound to have forgotten some.

- Replace the CCX (6 CX gates per call) gates that are later uncomputed by Margolus-like gates (RCCX, 3 CX gates per call).
- Write your own implementation of the RCCX gate, you get cost lowered by 2 per call in comparison to the one provided by Qiskit.
- Avoid negating bits (applying the X gate). Instead of that remember that a bit was negated and modify the calls that refer to it. In particular, implement the gates that I called RCnCX, RnCCX, and RnCnCX.
- Implement a special version of two neighboring RCCX gates where you only change the first argument. This will take 4 CX gates instead of the original 6 — see Figure 4 for details.
- Reorganize the code so that the phenomenon mentioned above occurs more often (the Margolus gate is not symmetric, i.e. $\mathrm{RCCX}(a, b, c) \neq \mathrm{RCCX}(b, a, c)$).
- Implement a special version of two neighboring RCCX gates where you negate one of the arguments. Instead of 6 CX gates you get 2 for negating the first argument and 4 for negating the second.

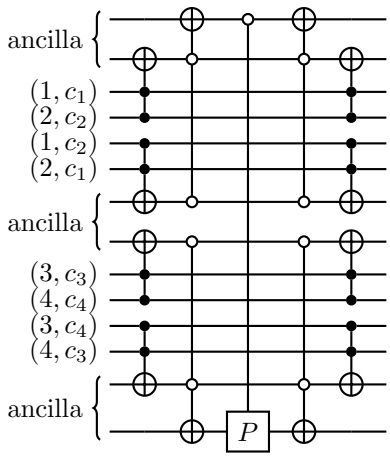

FIGURE 3. The circuit P_C performing the B_C-controlled phase shift P for a column subset $C = \{c_1, c_2\}$. We denote the elements of $\{1, 2, 3, 4\} \setminus C$ by c_3 and c_4.

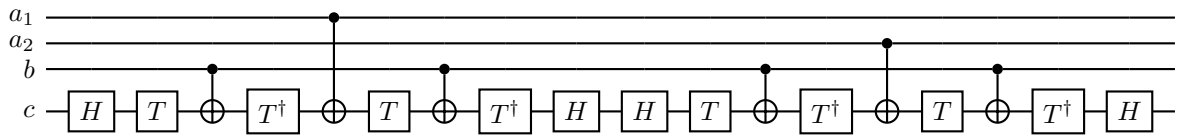

FIGURE 4. When performing an RCCX(a_2, b, c) immediately after an RCCX(a_1, b, c), the middle 8 gates (including 2 CX) can be removed, as they together constitute identity.

- Delay the execution of the last one-qubit gate on the target of recent RCCXs. It is likely that can be subject to gate synthesis very soon.
- Keep results of reused operations in unused ancilla qubits.
- Look at the resulting QASM code and try to find places to rewrite manually.

8. Optimizing the QRAM set-up

This is the last round of optimizations I made. The idea came to me when looking at the way QRAM is prepared for the sample data set. Observe that the last star is present in boards 0, 1, 2, 3, 4, 5, 6, and 8. Therefore, instead of performing the 8 board-conditioned CX calls sometime on the way of the Gray-code chain, we can do the following.

- Perform a CX conditioned on the MSB of address being zero, anytime during the set-up.
- Perform a CX in the Gray-code chain step corresponding to address 7.
- Perform a CX in the Gray-code chain step corresponding to address 8.

This saves us 5 CX calls with no additional overhead.

I generalized this method to use the potential control bits already used during the QRAM set-up process. More precisely, these were the cases used as possibilities additional to just negating a star in the Gray-code chain.

- No condition, just negate for all boards.
- Condition of address bit j being one, $j = 0, \ldots, 3$.
- Condition on two less significant bits of the address being j, $j = 0, \ldots, 3$.
- Condition on two more significant bits of the address being j, $j = 0, \ldots, 3$.

The very last optimization that I made is the one that promotes applying a set of the "block-controlled" operations such that two of the boards are left with no stars to negate manually. This allowed skipping some of the Gray-code chain steps and gain a better score.