Towards efficient automatic oracle synthesis and resource estimation using QDK and QIR

I-Tung Chen*,
Department of Electrical and Computer Engineering,
University of Washington, Seattle, WA, USA
Email: itungc@uw.edu
* Corresponding author

Chaman Gupta,
Department of Material Science and Engineering,
University of Washington, Seattle, WA, USA
Email: chaman@uw.edu

2023 IEEE International Conference on Quantum Computing and Engineering (QCE) | 979-8-3503-4323-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/QCE57702.2023.10207

***Abstract*—Automatic oracle synthesis (AOS) provides a promising route to speed up the process of constructing a complex quantum oracle. Here, we expand the existing AOS in the Microsoft Quantum Development Kit (QDK) by adding new functionalities and introducing a new workflow. We also optimize the AOS process by reducing the number of T gates using measurement-based uncomputation. Furthermore, we compare the resource requirements of the oracles generated using AOS to those of the existing QDK library oracles using the Azure Quantum Resource Estimator. The results suggest that the oracles generated using AOS perform better in runtime but use more qubits compared to the corresponding QDK library oracle. With the presented workflow, one can easily implement and estimate the quantum resources required for oracles with complex arithmetic functions. Finally, we present a strategy that can further reduce the number of physical qubits needed in AOS.**

I. Introductions

Quantum oracles are often elusive and needed to be handcrafted in different quantum algorithms [1], [2]. Automatic oracle synthesis (AOS) using Q# [3] is a promising route to implement arbitrary quantum oracles. In addition, Q# can be compiled into LLVM [4] quantum intermediate representation (QIR) [5], which allows AOS to be performed using logic networks such as XOR-AND-Inverter graphs (XAGs) [6]. Here, we expand the existing AOS from the Microsoft Quantum Development Kit (QDK) [7], [8], [9] with new arithmetic operators and 64-bit integer input type. With the help of our expansion on AOS, one can write a quantum program as follows:

```
namespace OracleGenerator.Classical {
  internal function AddMultiply(x: Int, y: Int, z:
    Int): Int {
    return (x + y*z);
  }
}
namespace Operation {
  operation AddMultiply(
    inputs: (Qubit[], Qubit[], Qubit[]),
    output: Qubit[]
  ): Unit {}

  @EntryPoint()
  operation Program(): Unit {
    use (x, y, z) = (Qubit[], Qubit[], Qubit[]);
    use v = Qubit[];
    AddMultiply((x, y, z), v);
  }
}
```

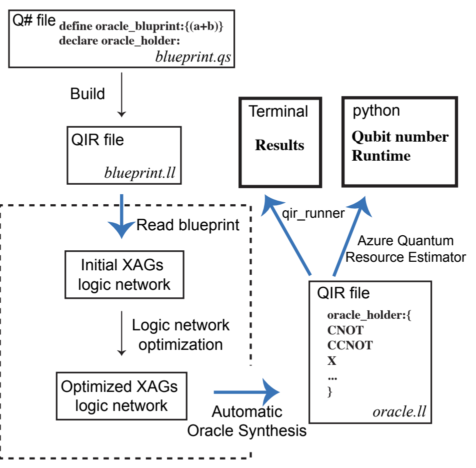

Fig. 1. **Compilation flow for automatic oracle synthesis.**

The program contains an oracle blueprint AddMultiply in the Classical namespace. The operation AddMultiply is empty and will be generated using AOS. We also compare the resources required for the oracles generated using AOS with similar existing Q# library functions. In addition, a measurement-based uncomputation [10] was incorporated to optimize the AOS process and reduce the quantum resources.

II. Workflow and case study

The workflow of AOS [3], as shown in Fig. 1, can be described in the following. First, a desired blueprint is first defined in Q# and an empty holder is also declared to hold the forthcoming AOS codes. The Q# file is compiled into a QIR file (.ll) and read by the AOS program, which generates and optimizes XAGs networks [11]. Finally, a reversible quantum logic circuit is generated using universal quantum gates (NOT, CNOT, and Toffoli gates) in the QIR file. The QIR file can be simulated using QIR-runner and loaded by Azure Quantum Resource Estimator [12] for quantum resource estimation. Our work's main contributions can be identified through the blue arrows, as depicted in Fig. 1.

Now, we apply the same workflow for Grover's algorithm to search for a missing ISBN digit of a book. This case study

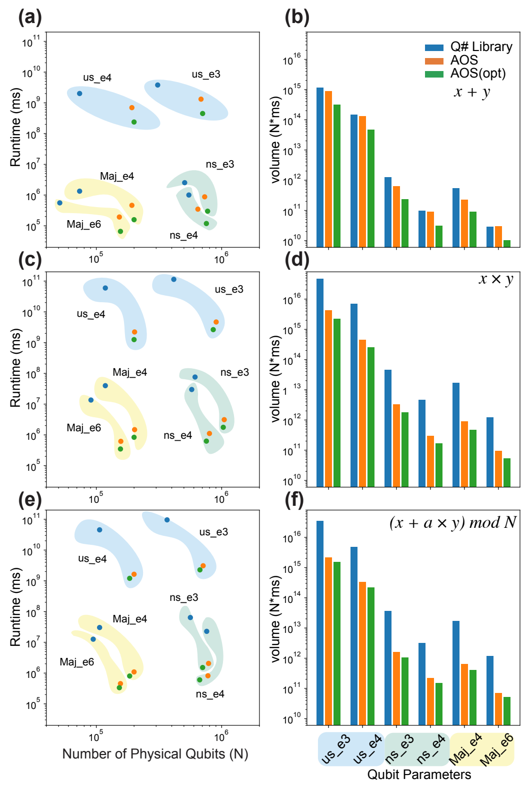

Fig. 2. **Resources required for library functions in Q# compared with their respective AOS functions.**

is referenced from [13]. Here, the oracle uses the $(x + a \cdot y) \bmod N$ operation to determine which digit (0 to 9) satisfies the ISBN rule, where a and N are constant inputs, and x and y are qubit registers. We can simply define the oracle blueprint as

```
namespace OracleGenerator.Classical {
  internal function addmod(x: Int, y: Int): Int {
    let (a, N) = (6, 11);
    return ((x + ((a*y)%N))%N);
  }
}
```

III. AOS Resource Estimation

We compare the resource needed for the oracles generated using AOS with similar Q# library oracles. The library oracles used here are AddI, MultiplyI, and MultiplyAndAddByModularInteger, which corresponds to AOS implementations x+y, x*y, and (x+a *y) mod N, respectively. x and y represent qubit registers inputs while a and N are constants, which do not count as inputs. The only difference between the two is that the library oracles use in-place computation. The resource estimation result is shown in Fig. 2. We estimate the resources on six different qubit parameters [12] which have the operational characteristic that may correspond to future ion-based qubits, superconducting qubits, and Majorana qubits. As shown in Fig. 2, the library oracles require fewer qubits, but a longer runtime than oracles generated using AOS, which can be explained by the in-place computation differences. A reduction in the runtime can be seen between the optimized (green) and original (orange) AOS from the measurement-based uncomputation. Finally, the resources required to run Grover's algorithm in both oracles show similar results as before. Note that we used the ns_e3 qubit parameter here.

Algorithm	Physical Qubits	Total Runtime
AOS oracle	621,740	4ms 752us
Library oracle	572,850	77ms 646us

IV. Conclusions and Outlook

The AOS workflow empowers developers to implement complex arithmetic functions without requiring in-depth knowledge of the library functions that map such arithmetic functions to qubit registers. The path forward is to include the support of floating point and fixed point data types and incorporate the in-place computation to reduce the number of qubits required for the AOS. The source code used in this project can be found here [14].

V. Acknowledgements

We thank Mariia Mykhailova and Dr. Mathias Soeken from Microsoft Quantum for mentoring this project, the fruitful weekly discussions, and the detailed feedback on the drafting of this work. We also thank UW's QuantumX and AQET for funding this work.

References

[1] D. W. Berry, A. M. Childs, and R. Kothari, "Hamiltonian simulation with nearly optimal dependence on all parameters," in *2015 IEEE 56th annual symposium on foundations of computer science*, pp. 792–809, IEEE, 2015.
[2] S. P. Jordan, "Fast quantum algorithm for numerical gradient estimation," *Physical review letters*, vol. 95, no. 5, p. 050501, 2005.
[3] M. Soeken and M. Mykhailova, "Automatic oracle generation in microsoft's quantum development kit using qir and llvm passes," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1363–1366, 2022.
[4] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *International symposium on code generation and optimization, 2004. CGO 2004.*, pp. 75–86, IEEE, 2004.
[5] QIR Alliance, *QIR Specification*, 2021. Also see https://qir-alliance.org.
[6] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. T. Calvino, D. S. Marakkalage, *et al.*, "The epfl logic synthesis libraries," *arXiv preprint arXiv:1805.05121*, 2018.
[7] I. L. Markov and M. Saeedi, "Constant-optimized quantum circuits for modular multiplication and exponentiation," 2015.
[8] I. L. Markov and M. Saeedi, "Faster quantum number factoring via circuit synthesis," *Phys. Rev. A*, vol. 87, p. 012310, Jan 2013.
[9] G. Meuli, M. Soeken, E. Campbell, M. Roetteler, and G. D. Micheli, "The role of multiplicative complexity in compiling low t-count oracle circuits," 2019.
[10] C. Gidney, "Halving the cost of quantum addition," *Quantum*, vol. 2, p. 74, jun 2018.
[11] G. Meuli, M. Soeken, and G. De Micheli, "Xor-and-inverter graphs for quantum compilation," *npj Quantum Information*, vol. 8, no. 1, p. 7, 2022.
[12] M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram, and A. Vaschillo, "Assessing requirements to scale to practical quantum advantage," 2022.
[13] Microsoft, "Use the q# libraries, https://learn.microsoft.com/en-us/training/modules/use-qsharp-libraries/."
[14] "GitHub repository of this project, https://github.com/itungc/automatic-oracle-synthesis-using-qdk-and-qir.git,"