Open Quantum Assembly Language

Andrew W. Cross and Lev S. Bishop and John A. Smolin
and Jay M. Gambetta

arXiv:quant-ph/1707.03429
2017

Dylan Miracle
ICS 698-02
Spring 2021
Feb 17, 2021
Dr. Jigang Liu

Open Quantum Assembly Language

Dylan Miracle
Department of Computer Science
Metropolitan State University
St. Paul, Minnesota, USA
dylan.miracle@my.metrostate.edu

February 28, 2021

1 Background

Quantum computing ecosystem includes many different software architectures, compilers and languages. The dominant tool used in developing quantum algorithms is the quantum gate model acting on arrays of qubits. IBM Quantum Experience allows users to simulate and implement quantum circuits on IMB simulators and hardware. Users of the IBM QE are able to develop circuits graphically using a drag and drop circuit builder of as code using the QASM language defined in this paper.

2 Main idea/conclusion

The purpose of QASM is to act as an Intermediate Representation between an imagined quantum circuit and a compiled quantum code, real signals that control a quantum computer. The intermediate representation is similar to x86 assembly code acting between a C program and machine code. In this paper allowed operation and the language syntax is defined.

3 Support facts/algorithms/methods

QASM allows developers to declare two kinds of data, quantum registers, qubits, and classical registers, bits. These are declared as qreg, creg respectively. Each register is a 1D array of either qubits or classical bits. These are referenced as C arrays. For example the following code snippet will create two qubits and three classical bits. Then will measure the value of the first qubit and store the output in the first classical register.

```
qreg a[2];
creg b[3];
measure a[0] -> b[0];
```

You can see that the syntax is similar to C and creates a human readable method for manipulating quantum bits.

The paper parameterizes all single qubit quantum gates as rotations around three axis. In order to implement a gate in QASM developers may either use one of the predefined gates, or implement their own gate as a series of rotations. The definition is given in Equation (2) and is implemented in code as U(theta, phi, lambda) a; which will apply the unitary transform to the qubits in register a.

4 Arguments/disagreements/concerns

QASM is limited in abstraction, you can only define operations mathematically as unitary gates. While this is the way real qubits are manipulated in hardware implementations, for quantum computing to gain more traction users will need abstractions that they can use to build more complicated and useful algorithms.

QASM is also limited to a single code branch. That is there is only and entry, and exit with no flow control, conditionals or branching. This is because the circuit model represented by QASM is a single path, thus a language that defines such a circuit must obey this restriction. However this makes creating programs difficult and does not allow such basic programming principles such as DRY (Don't Repeat Yourself) coding. If a developer wishes to implement a series of gates 10 times, the programmer will have to enter the routine 10 times into the code as opposed to having fundamental coding tools such as loops. This makes QASM unwieldy to code and often difficult to read for more complex algorithms.

5 Interesting findings

The paper presents the header that defines all the standard quantum gates available to developers. The header also reserves all the symbols used for common gates. The set of gates defined in the header are all that is needed to implement many common algorithms. The gates implemented in the header are also available to users of the IBM Quantum Experience as well as the gates that are implemented by the hardware.

In addition to the header definition the paper goes on to show six common quantum algorithms implemented using QASM. The most basic algorithm implemented is the Quantum teleportation algorithm, in which a single qubit is teleported from one register to another. In the context of classical computing this is a simple operation and not an algorithm, however in quantum computing an series of operations are required as quantum information may not be copied, the so called "No cloning theorem". The simplest way to understand the no cloning theorem is to understand that all quantum operations must be reversible and imagine that we can copy a qubit. Now imagine that you have 2 registers, one with a single qubit and another that will hold the copied qubit. In the forward copy you start with a qubit in register 1 and end with the qubit in register 1 and 2. Now imagine we go backwards. How will you know which register was the one that started with the qubit and how can you find out the state of the register you overwrote. Thus we must either throw out the reversibility requirement or the ability to copy. Reversibility is fundamental to the math of unitary transformations and cannot be discarded, so we must assume we cannot copy a qubit.

6 Quotations

"Our goal in this document is to describe an interface language for the Quantum Experience that enables experiments with small depth quantum circuits."

"The Quantum Experience standard header defines the gates that are implemented by the hardware, gates that appear in the Quantum Experience composer, and a hierarchy of additional user-defined gates. Our approach is to define physical gates that the hardware implements in terms of the abstract gates U and CX. The current physical gates supported by the Quantum Experience are a superset of the abstract gates, but this is not true of all physical gate sets and devices. Choosing to use abstract gates merely to define physical gates gives some fexibility to add or change physical gates at a later time without changing Open QASM. We believe this approach is preferable to invisibly compiling abstract gates to physical gates or to changing the underlying set of abstract gates whenever the hardware changes."