I. SCAFFOLD

Scaffold [1] is a quantum programming language developed by a team from Princeton University and others. Using Scaffold, the computational operations and data structures involved in a quantum algorithm can be programmed and finally compiled into a machine-executable form. The team also developed Scaffold's compiler, ScaffCC, which optionally compiles Scaffold source code into instructions that can eventually be executed on QX, a quantum simulator developed by QuTech Labs.

The Scaffold is an extension of the C language. It adds new data types, such as `qbit` and `cbit`, and defines quantum operations, including Pauli-X gates, Hadamard gates, and other quantum logic gates. A program written with Scaffold comprises the *classical part* and the *quantum part*. The former includes classical data types and control structures; the latter includes quantum data types and operations.

A complete Scaffold program usually consists of multiple modules, and since quantum circuits are always "reversible," these modules must satisfy the following requirements to be executed on a quantum device.

- Either consists only of unitary quantum operations.
- Or be able to be compiled into unitary quantum operation instructions.

To compile some classical modules into unitary quantum instructions, Scaffold includes the CTQG module, which can compile some classical circuits into an instruction set consisting of only NOT, controlled-NOT (CNOT), and Toffoli gates. For example, when calculating the addition `a+b`, if `N`-bit binary numbers can represent both `a` and `b`, this classical instruction can be composed by `6N-3` CNOT gates and `2N-2` Toffoli gates without auxiliary qubits. Therefore, the operations such as classical addition computed by the user in Scaffold are eventually compiled and executed as a set of quantum operation instructions.

In the following, we briefly introduce what syntactic details Scaffold adds to the classical programming language.

A. Quantum Data Types

The most basic quantum data type in Scaffold is the quantum register **`qreg`**, which can be declared by the statement **`qreg`** `qs[n]`, where `n` denotes the number of qubits in the register, even if `n=1`, it is still treated as a quantum register, not as a separate qubit. Alternatively, multiple quantum registers can be declared as a single quantum structure **`qstruct`**:

```
qstruct struct1 {
qreg first[10];
qreg second[10];
};
```

In this case, quantum structure **`struct1`** contains two quantum registers: `first` and `second`. Through the following two statements, one can access the first qubit of `second` in the quantum structure **`struct1`**.

```
struct1 qst;
qst.second[0];
```

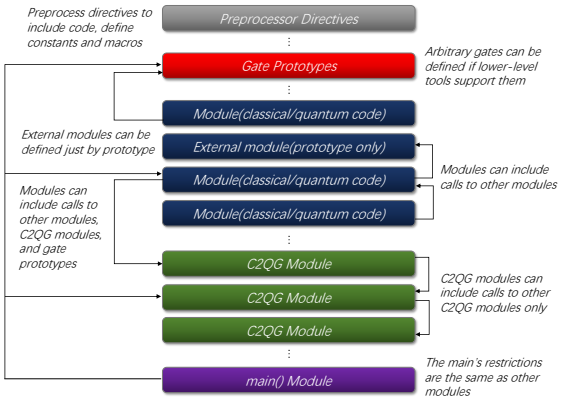

Fig. 1. The whole structure of a Scaffold program [1].

B. Quantum Gates

The quantum gate operation in Scaffold can be divided into two categories according to the implementation: the built-in quantum gates in the standard library and the quantum gates defined by the gate prototype function. The use of the first type of gate function only requires the introduction of the header file `gates.h`.

According to the prototype function, the second type of gate operation requires its own definition.

```
gate gatename(type_1 parameter_1, ..., type_n parameter_n);
```

The above statement defines the gate operation named `gatename`, which consists of `n` arguments, the data type of these `n` arguments can be quantum registers or classical unsigned integers, characters, floating point numbers, and double precision floating point numbers, both of which must be passed to the function by reference (if the classical data type with the `const` keyword can be passed by value). At this point, in the module that defines the gate operation `gatename`, the gate operation can be called according to the following statement:

```
gatename(parameter_1, ..., parameter_n);
```

```
//module prototypes. They are defined elsewhere
module U (qreg input[4], int n);
module V (qreg input[4]);
module W (qreg input[4], float p);

//Quantum control primitive
module control_example(qreg input[4])
if (control_1[0]==1 && control_2[0]==1)
    U(input);
else if (control_10==1 && control_2[0]==0)
    V(input);
else
    W(input);
```

Fig. 2. Example quantum control primitive for controlled execution of 3 different modules, U, V, and W [1]

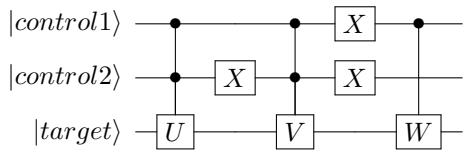

Fig. 3. The quantum circuit corresponding to the program in Figure 2.

C. Loops and Control Structures

Like the C language, Scaffold supports `if`, `switch`, and `loop` statements, but the control styles of these statements can only contain classic information. Scaffold also provides a quantum control statement; the control predicate contains qubits. When computing the control predicate statement, the quantum state of the qubit is determined, and the code is executed according to the result. For example, Figure 2 shows a simple Scaffold program taken from [1], which describes quantum control primitive for controlled execution of 3 different modules. Obviously, this program will determine which code to execute according to the quantum states of control qubits `control_1[0]` and `control_2[0]`. According to the principle of delay measurement, this code will be compiled into the circuit shown in Figure 3. Note that the qubits in the control predicate cannot be used again in the program.

D. Modules

The Scaffold supports modular design for readability, maintainability, and other features. A complete program usually consists of one or more modules, each of which is responsible for one task and can pass data of classical or quantum data types between modules. The syntax for defining a module is as follows.

```
return_type module module_name
(type_1 parameter_1, ..., type_n parameter_n);
```

where `return_type` can be null, integer, character, floating-point, double-precision floating-point, or structure, and the requirements for the argument list are the same as those for the gate prototype function. The defined module can be called by

```
module_name(parameter_1, ..., parameter_n);
```

Figure 1 shows the whole structure of a Scaffold program, taken from [1].

References

[1] Ali J Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Chen-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. Scaffold: Quantum programming language. Technical report, Department of Computer Science, Princeton University, 2012.