Research on Quantum Programming Framework Based on NISQ Function

Xiaonan Liu², Le Xu¹*, Chunchao Hu¹ and Lei Jin¹

¹ School of Information Engineering, Zhengzhou University, Zhengzhou, Henan, 450000, China
² State Key Laboratory of Mathematical Engineering and Advanced Computing, Information Engineering University, Zhengzhou, Henan, 450000, China
*Corresponding author’s e-mail: xl@gs.zzu.edu.cn

Abstract. In the past two decades, the field of quantum computing has developed rapidly, is gradually able to solve some difficult classical problems. With the development of the facilities for Noisy Intermediate-Scale Quantum (NISQ), the NISQ technology addresses the existing quantum programming languages or frameworks, suggesting that they cannot fully meet the requirements of Quantum computers. In recent years, some quantum programming languages or frameworks have emerged which claim to support the classical computing algorithms of heterogeneous quantum. However, the code is too cumbersome and redundant, and it is difficult to map the code to heterogeneous quantum classical computation while meeting the requirements of timing. Therefore, a modular programming framework of heterogeneous quantum classical computing algorithm with NISQ function is proposed. The framework optimizes the quantum code and retains the quantum classical interactions that satisfy the temporal constraints.

1. Introduction

NISQ equipment has developed rapidly in recent years, such as IBM's 50-qubit qubit computer and Google's 72-qubit computer, have developed rapidly in recent years. Compared with classical computers, quantum computers carry more difficult tasks. Nowadays, quantum computer is in the NISQ period, and all quantum computers that can be produced at this stage are in the category defined by NISQ. If quantum computing wants to be able to operate for a long time, such objective conditions cannot be completed. The extremely high Error rate and too short stability time make Quantum calculation Error Correction capability imminent. However, all the current studies can only realize the corresponding quantum error correction (QEC) when the number of quantum bits exceeds 1000. There may be many noisy qubits in quantum systems, and it is difficult for people to perform some meaningful quantum computation through error correction[1]. In recent years, although quantum software has made great progress, the existing quantum programming languages and frameworks cannot meet the requirements of NISQ technology well.

Through further research on quantum computing, people are more inclined to solve some classical difficult problems, such as quantum approximate optimization algorithm (QAOA) as a classical quantum hybrid algorithm, which provides power for the first real-world quantum computing application with scientific and commercial value[2].
At present, there are some quantum programming languages or frameworks for classical heterogeneous quantum computing, such as ProjectQ, Q#, and Qiskit. However, there are still some defects: there is no framework to define the classical computation of heterogeneous quantum, which will produce some extra redundant code, which will fuse logic and process control together, which is not conducive to the quantum computation.

This paper proposes a Quantum Domain Specific Language (QDSL), which has well-defined semantics and is a timer based timing control scheme that allows flexible binding between language-level operations using specific semantics on the target platform.

2. Background

2.1. Heterogeneous quantum classical computing architecture
Heterogeneous computing technology emerged in the mid-1980s, and has become one of the research hotspots in the field of parallel/distributed computing due to its economical and effective acquisition of high-performance computing power, good scalability, high utilization rate of computing resources and great development potential. The ideal heterogeneous computing has the following elements:(1) the computing resources it uses have various types of computing power, such as simd, mimd, vector, scalar, special, etc.; (2) It needs to identify the parallelism requirement type of each sub-task in the computation task; (3) It needs to coordinate the operation of computing resources with different computing types; (4) It should not only develop the parallelism in the application problem, but also develop the heterogeneity in the application problem, that is, it should pursue the matching between the computation type of the computing resource and the task (or sub-task) type it executes[3]; (5) The ultimate goal it pursues is to make the execution of computing tasks have the shortest time. OpenCL is a widely accepted parallel computing industry standard[4]. In an OpenCL based architecture, hosts and accelerators can access shared memory to exchange data.

![Host CPU](Host CPU)

![Quantum Coprocessor](Quantum Coprocessor)

![CPU](CPU)

![Control Unit](Control Unit)

![Qubit](Qubit)

Figure 1. Heterogeneous quantum classical computing architecture

By using the shared memory approach, a classical computer can communicate with a quantum coprocessor, but there is a delay, which may be even greater when considering the shared memory-based communication with the memory control unit and the operating system costs. In the NISQ era, what must be done is to correct quantum errors; otherwise, quantum bits may decay, and quantum data will be lost when such slow communication goes on.

Quantum control processors can execute quantum instructions that specify quantum operations with timing and help classical instructions update classical registers and boot program streams. And, as a result of quantum instructions and auxiliary classic performed by the mixed quantum control processor, these classic instruction can exchange data with periodic delays quantum instructions, for rapid interaction between classical computing and quantum computing, possible, not only supports real-time feedback, also support based on the process and selection of circulation, structured program description[5].
2.2. Heterogeneous quantum classical computing algorithm
Because the variational algorithm can significantly reduce the requirement of quantum bit coherence time, it is very promising in the short term and has a wide range of applications in quantum simulation and optimization. Variational algorithms seamlessly integrate classical subroutines such as search and quantum subroutines. In addition, quantum subroutines can use classical constructs such as loops and measuring-based branches. VQE algorithm can explain some properties of classical heterogeneous quantum computing algorithms and their requirements for quantum programming languages and frameworks.

Variable component subbook solicitation (VQE) algorithm. For a larger square matrix $A_n$ of order $n$, if you want to find its eigenvalue $\lambda_1, \lambda_2, ... \lambda_n$, you can use the VQE algorithm to achieve. For the Hamiltonian describing a system, the VQE algorithm can also find its eigenvalues, and then obtain the ground state energy of the system. The basic steps of the VQE algorithm are shown in Figure 2.

![Figure 2. The basic steps of the VQE algorithm](image)

As can be seen from the figure, the VQE algorithm combines quantum computation with classical computation. The VQE algorithm needs to complete three steps successively: the first step is to prepare the experimental state, the second step is to measure the average energy $E_n$ of the experimental state, and the third step is to judge whether $E_n - E_{n-1}$ is less than the threshold we set. If it is, it will return $E_n$ as the energy of our ground state. At this time, the experimental state we prepared is the ground state we are looking for. Otherwise, a new set of parameters $t_n$ is generated by the optimizer to reproduce the experimental state. This is a cyclical iterative process.

3. QDSL overview

3.1. Design principles
The quantum domain specific language (QDSL) designed in this paper is to support quantum experiments and develop heterogeneous quantum classical computing framework. QDSL adopts the following design principles:

(1) Modular system. The framework must be a modular system in which modules can interact with each other but do not interfere with each other’s functions.

(2) An architecture for classical computing of heterogeneous quantum. It ensures that quantum computation and classical computation can be quickly and accurately mapped to classical computers and quantum control processors without the need for additional program structure and special unique variables.

(3) It has core functions and can be expanded. QDSL will provide core functionality with as few quantum concept sets as possible, and support strong scalability in the future.
3.2. Design overview
The QDSL framework includes the following components:

(1) Source file: The quantum program consists of classical and quantum parts. The classical part describes the program in classical languages such as Python, and the QDSL language describes the quantum program.

(2) Compiler: The compiler aspect mainly includes the classical compiler and the quantum compiler. At the same time, a pulse generator is needed to generate pulse signals for quantum optimal control[6].

(3) Programming environment: This environment provides the API interface of classical program and quantum program. At the same time each system component can coordinate with each other to run.

The host program is responsible for describing the classical computing tasks and for making calls to the quantum kernel. The quantum kernel handles the computation in the quantum coprocessor.

4. QDSL language
QDSL is a custom domain specific language. QDSL is designed to define the language's syntax and data structure with as few concepts as possible. QDSL is a special syntax that combines classical operations with quantum operations. For users, it can simplify the difficulty of tedious procedures and improve the programming efficiency, so that users only need to master the core grammar structure. The simplified core syntax simplifies the definition of the language, optimizes the compiler design form, and improves the scalability and stability of the quantum programming language.

Code 1. QDSL implements the quantum Fourier transform.

```python
import config.json,
import operations.

operation dummy(): unit
{
    using(q: qubit) {
        X180(q);
        measure(q);
    }
}

operation cu1(lambda: double, a: qubit, b: qubit): unit {
    UI(a, lambda/toDouble());
    CZ(a, b);
    UI(b, lambda/toDouble());
    CZ(a, b);
    UI(b, lambda/toDouble());
}

operation qft(number: int, q: qubit[]): unit {
    int i = 0;
    while (i < number) {
        int j = i;
        while (j > 0) {
            cu1(3.14/toDouble(2^n), q[i], q[i-j]);
            j=j-1;
        }
        H(q[i]);
        i = i+1;
    }
}

operation main(): bool[] {
    bool[s] output;
    using(q: qubit[s]) {
        for (int i = 0; i < s; i++){
            init(q[i]);
        }
        qft(s, q);
        output = meas_all(q);
        return output;
    }
```
4.1. Module system
The QDSL language is designed to be modular, putting the same kinds of operations in the same package so that they can be compiled separately, avoiding code conflicts and simplifying the way code is written. We use the package at the top of the source file to declare a package, to indicate that all classical or quantum operations defined in this file belong to this package. Operations in a package can be imported from other files using import statements. Some quantum operations are defined in the operation package. Code 1 is to implement quantum Fourier transform algorithm using QDSL.

4.2. Type system
The QDSL language has a powerful static typing system. It is based on the heterogeneous quantum classical computing model. The emphasis is to describe the tasks performed on the processing quantum coprocessor. There are two types of QDSL data types, classical data type and quantum data type. One of the features of QDSL language is its simplicity, so we designed QDSL to support four classical types and one quantum type, as well as two compound types.

4.2.1. Classic type
QDSL supports four classic types, including int, bool, double, and unit. Unit is used to describe the return type of an operation that does not return a value. QDSL also supports other classical types, including addition (+), subtraction (-), multiplication (*), division (/), and (&&), or (||), and comparison (<, >, <=, >=, ==).

4.2.2. Basic quantum types
QDSL defines qubits as the only type of quantum data in the language, which reduces the number of declarations of unwanted quantum data types. When a qubit is called, one or more qubits can be allocated using the using call instruction statement. A qubit allocated by a using statement is a variable reference that uses a qubit type (such as a qubit assigned to variables A and B in Code 1). A qubit-type variable defined in the same module can only be used in the current module. Qubits are de-allocated when exiting the module.

4.2.3. Compound type
A compound type uses arrays and tuples to define the collection of data. An array is an ordered sequence of elements of the same type. A tuple is an ordered heterogeneous collection of elements of any QDSL type.

4.3. Configuration files and operation
4.3.1. Configuration files
The QDSL configuration file format is similar to JSON, using the Package statement to declare the name of the package. The configuration file consists of two main parts. The first part is the platform definition, which describes the characteristics of the target architecture, such as the number of qubits available. The second part is the operation definition, which provides information about the original operation on the architecture. Code 2 is the content of part of the configuration file.
4.3.2. Operation
In QDSL, functions are defined as operations, which are simply functional functions that perform a particular quantum computation.Opaque and operation are used to define these operations, including the platform-dependent configuration files that you can use as the base operations, and user-defined classical or quantum operations. As a user, you can use the operation keyword to start the operation, followed by the function name, argument list, and argument return type, and finally the body of the function. Using function definition statements, classical and quantum computing can be combined.

5. Key technologies of the QDSL framework
In the description of heterogeneous quantum classical computing algorithms, classical computing tasks described by classical languages are included, while QDSL language can only describe the tasks performed by quantum coprocessors. To further optimize the quantum kernel, the two parts must work together, and the runtime system needs to combine the target hardware platform, quantum compiler, host program and quantum kernel.

5.1. Quantum program cycle
The quantum program cycle consists of four phases:
(1) Classical compilation: Firstly, the classical programming language is used as the main program, QDSL is used as the quantum kernel to describe the quantum application program, and then the main program is output as the classical binary file.
(2) Quantum compilation: The quantum compiler compiles the quantum kernel into a quantum binary file composed of classical and quantum hybrid instructions.
(3) Quantum execution: Quantum control processors perform the task of executing quantum binaries. The classical registers are then updated and the corresponding quantum operations are applied to the qubits. In this way, the quantum state evolves under the control of the program, thus
completing the kernel computing task. The results are then sent to the classical host by writing to the shared memory between the quantum coprocessor and the host.

(4) Classical execution: The classical binary continues to execute, then reads the quantum computation results, and performs possible post-processing.

5.2. Operation system
Throughout the framework, each part of the program has its own tasks, including the compiler, the system kernel, the main program, and the quantum control processor, making it impossible for them to work together. In order to make each component work together, this article uses the QDSL runtime system as the supporting environment of the QDSL framework. The QDSL runtime system is designed as a library that runs on a classic host, handling the interaction between the host program, the QDSL kernel, the QDSL compiler, and the quantum coprocessor. It mainly includes five parts: the first part is the system configurator, which is mainly responsible for configuring the execution environment of the quantum program; The second part is the host language interface, which enables the host program to call the quantum kernel to solve problems and read results from the quantum kernel by using the quantum coprocessor. The third part is the interface that calls the various quantum backends to execute quantum code and make them return the calculated results. The interface is implemented as a variety of quantum back-end drivers. The fourth part is the parameter converter and kernel result decoder, which is responsible for the communication between host and kernel. The fifth part is the phase manager, which is responsible for triggering activities at different stages of the program lifecycle model.

5.3. Quantum compilation
QDSL language can be used to describe heterogeneous quantum classical computing applications with classical and quantum interactions. The intermediate representation (IR) can be used to construct the quantum compiler instead of constructing the quantum circuit for easy operation. For example, the origin of the original quantum is OriginIR or the MLIR of Google. The ability to use a quantum coprocessor to execute a mixture of classical and quantum instructions.

5.4. Optimization of the quantum kernel
For the optimization of quantum kernel, we can use the static analysis method to optimize the quantum program. Static methods include constant propagation, loop unrolling, eliminating dead code, and so on. The compiler executes values that can be exported for all static nodes, so there is no need to generate instructions corresponding to the static node. Classical operation instructions can be converted into classical instructions at runtime and passed to the quantum control processor for execution.

6. Conclusion
This article introduces the slow interaction between host computing and quantum coprocessor computing, as well as the fast interaction of classic instructions on the CPU. We designed a quantum domain special language QDSL, QDSL comes with a new compilation framework to match the model. This language allows people to operate heterogeneous quantum classical computing applications more conveniently and quickly.

Acknowledgments
This work was supported by the National Natural Science Foundation of China (No. 61972413, No.61701539).

References
[1] Preskill J. Quantum computing and the entanglement frontier[J]. Physics, 2012.
[2] Farhi E, Goldstone J, Gutmann S. A Quantum Approximate Optimization Algorithm[J]. Eprint Arxiv, 2014.

[3] Stone J E, Gohara D, Shi G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems[J]. Computing in ence and Engineering, 2010, 12(3):66-72.

[4] Sagastizabal R, Bonet-Monroig X, Singh M, et al. Error Mitigation by Symmetry Verification on a Variational Quantum Eigensolver[J]. 2019.

[5] Negrevergne C, Mahesh T S, Ryan C A, et al. Benchmarking quantum control methods on a 12-qubit system[J]. Physical Review Letters, 2006, 96(17):170501.

[6] Werschink J, Gross E K U. Quantum Optimal Control Theory[J]. Journal of Physics B Atomic Molecular & Optical Physics, 2007, 40(18):2007.