A Software Tool for Mapping and Executing Distributed Quantum Computations on a Network Simulator

D. Ferrari[*†@] S. Nasturzio[*] M. Amoretti[*†@]

[*]Department of Engineering and Architecture - University of Parma, Italy, [†]Quantum Information Science @ University of Parma, Italy,
[@]davide.ferrari1@unipr.it,michele.amoretti@unipr.it

Abstract

The growing demand for large-scale quantum computers is motivating research on distributed quantum computing (DQC) architectures [1]. To support the research community in the design and evaluation of distributed quantum protocols, many simulators have been devised to aid with the construction of the topology and the deployment of computations on it [2, 3, 4, 5]. However, the process of setting up a simulation requires strong expertise in the simulator itself, thus being inconvenient for those who are only interested in quantum protocol evaluation or in the design of supporting tools such as quantum compilers. In this work, we present a software tool denoted as DQC Executor, that accepts as input the description of the network and the code of the algorithm, and then executes the simulation by automatically constructing the network topology and mapping the computation onto it, in a framework-agnostic way and transparently to the user.

DQC Executor

The tool is in its early stages and currently supports automatic deployment of distributed quantum algorithms to the NetSquid [5] simulator, which allows a detailed physical modeling of individual network components. The description of the network is provided by the user in a specific YAML format. The distributed algorithm, instead, is defined with the OpenQASM [6] language.

NetSquid is one of the most advanced platform for simulating quantum networking and modular computing systems subject to physical non-idealities. It ranges from the physical layer and its control plane up to the application level. This is achieved by integrating several key technologies: a discrete-event simulation engine, a specialized quantum computing library, a modular framework for modeling quantum hardware devices, and an asynchronous programming framework for describing quantum protocols.

Open source code available at: `https://github.com/qis-unipr/dqc-executor`

Network Description with YAML

The first step in constructing a distributed quantum simulation is defining the structure and topology of the network in terms of nodes, connections, QPU's qubit map, and internode ebit coupling map. The DQC Executor tool can do this with a YAML formatted file. The choice of such format is due to its straightaway integration with NetSquid-NetConf snippet, a customizable open-source extension to NetSquid's framework for automatic network setup.

```yaml
network: main_network

ebit_coupling_map:
    entries:
        - alice[2] <-> bob[2]
        - alice[2] <-> charlie[1]

components:
    alice:
        type: quantum_node
        properties:
            n_of_qubits: 3
            ebits:
                - 2
    bob:
        type: quantum_node
        properties:
            n_of_qubits: 3
            ebits:
                - 2
            topology:
                - 0,2
                - 2,0
                - 1,2
                - 2,1
    charlie:
        type: quantum_node
        properties:
            n_of_qubits: 2
            ebits:
                - 1

quantum_connection_bob_alice:
    type: quantum_connection
    properties:
        length: 20
    connect_to:
        node1: alice
        node2: bob

quantum_connection_alice_charlie:
    type: quantum_connection
    properties:
        length: 20
    connect_to:
        node1: alice
        node2: charlie
```

A network such as the one illustrated in Figure 1 can be described with the YAML file shown above. Note that one must specify connections between nodes and which qubits in each node will be used as ebits. It is also possible to accurately describe the internal topology of each node, which is otherwise assumed to be all-to-all.

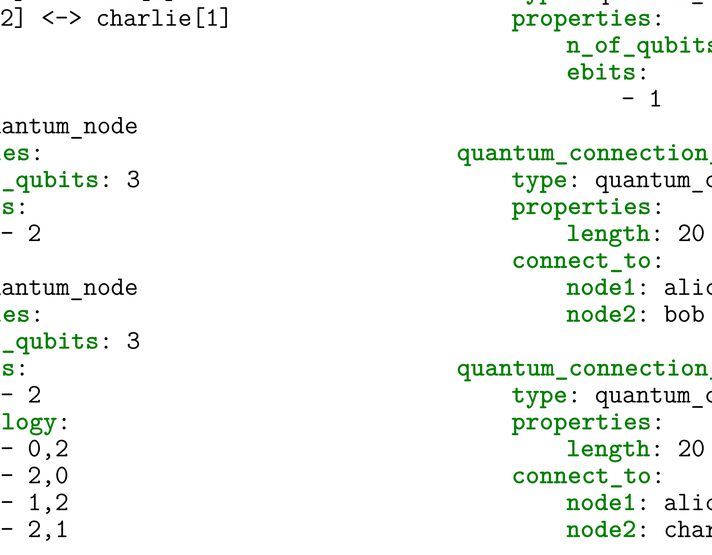

Figure 1: Example of network topology.

Distributed Algorithms with OpenQASM

Currently, DQC Executor only supports QASM files were all qubits of a QPU, including ebits, fit inside a single register. The mapping between between registers and nodes, as well as qubits int a register and ebits, is derived from matching the nodes' names and ebits numbering in the YAML file with registers' names and qubits indexes in the QASM. Regarding the new operations needed for a distributed algorithm, they are defined as opaque gates inside the QASM itself, and then parsed by Qiskit QASM's Parser. After the circuit has been parsed, one can visualize it and note that the opaque gates are represented by boxes, as shown in Figure 2, their implementation left to be specified later on in the simulation.

```
OPENQASM 2.0;
include "qelib1.inc";
opaque entangle e1, e2;
opaque remoteCX c,e1,t,e2;

qreg alice[3];
qreg bob[3];
qreg charlie[2];

h alice[0];
cx alice[0], alice[1];
entangle alice[2], bob[2];
remoteCX alice[0], alice[2], bob[0], bob[2];
entangle alice[2], bob[2];
remoteCX alice[0], alice[2], bob[1], bob[2];
entangle alice[2], charlie[1];
remoteCX alice[0], alice[2], charlie[0], charlie[1];
```

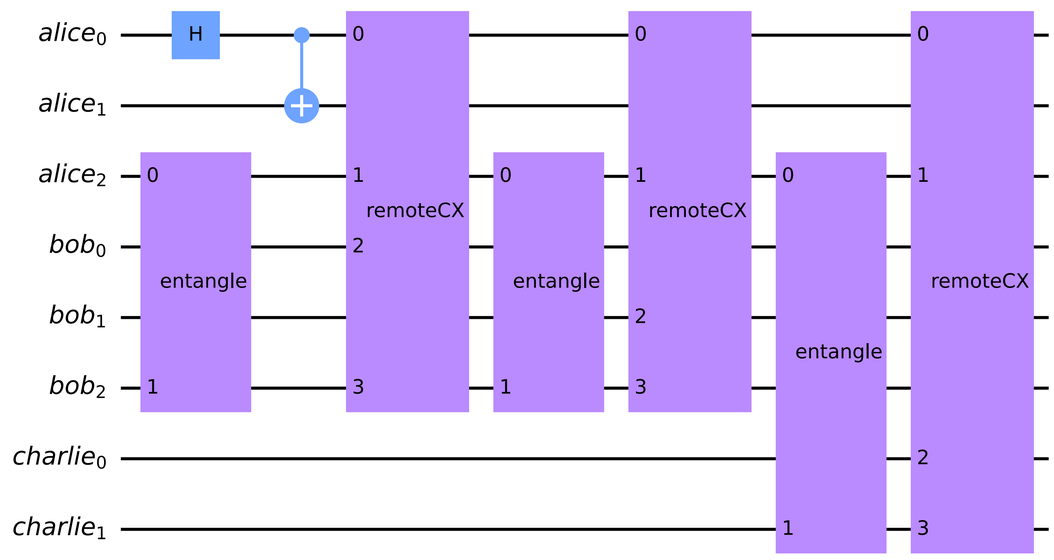

Figure 2: Example of circuit creating a shared GHZ state.

Tool Structure

As depicted in Figure 3, DQC Executor is characterized by three main running phases: input parsing, pre-processing, and simulation. The first one concerns the parsing of the YAML configuration file and the OpenQASM description, to extract the required information. The pre-processing phase sets up all the components required for the simulation, verifies the coherence of the algorithm and the topology, establishes connections between nodes, and finally constructs the simulation protocols that will be executed by the simulated nodes. The final phase is indeed the execution of the computation by each node, which receives the whole quantum circuit and starts executing only the gates that pertain to it.

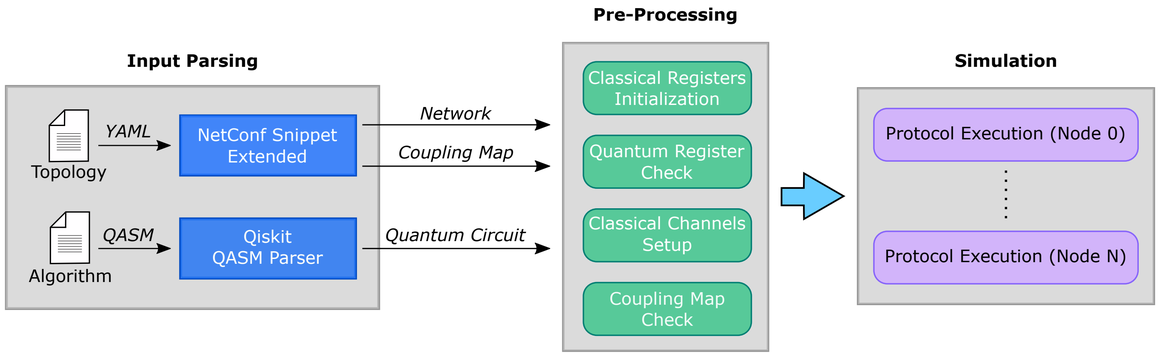

Figure 3: Structure of the DQC Executor.

References

[1] D. Ferrari, A. S. Cacciapuoti, M. Amoretti, and M. Caleffi. Compiler Design for Distributed Quantum Computing. *IEEE Transactions on Quantum Engineering*, (4100720):1–20, 2021.

[2] A. Dahlberg and S. Wehner. SimulaQron - a simulator for developing quantum internet software. *Quantum Science and Technology*, (1):015001, 2018.

[3] S. Diadamo, J. Notzel, B. Zanger, and M. M. Bese. QuNetSim: A Software Frameworkfor Quantum Networks. *IEEE Transactions on Quantum Engineering*, pages 1–1, 2021.

[4] T. Matsuo. Simulation of a Dynamic, RuleSet-based Quantum Network. *arXiv: 1908. 10758*, 2021.

[5] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. Oliveira, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres Knoop, D. Elkouss, and S. Wehner. NetSquid, a NETwork Simulator for QUantum Information using Discrete events. *Communications Physics*, 4(1):164, 2021.

[6] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. *arXiv: 1707. 03429*, 2017.