Quantum Algorithms and Simulation for Parallel and Distributed Quantum Computing

Rhea Parekh
Independent Scholar
rheaparekh12@gmail.com

Andrea Ricciardi
Independent Scholar
andrea.ricciardi@live.com

Ahmed Darwish, Stephen DiAdamo
Technische Universität München
{a.darwish, stephen.diadamo}@tum.de

***Abstract*—A viable approach for building large-scale quantum computers is to interlink small-scale quantum computers with a quantum network to create a larger distributed quantum computer. When designing quantum algorithms for such a distributed quantum computer, one can make use of the added parallelization and distribution abilities inherent in the system. An added difficulty to then overcome for distributed quantum computing is that a complex control system to orchestrate the various components is required. In this work, we aim to address these issues. We explicitly define what it means for a quantum algorithm to be distributed and then present various quantum algorithms that fit the definition. We discuss potential benefits and propose a high-level scheme for controlling the system. With this, we present our software framework called Interlin-q, a simulation platform that aims to simplify designing and verifying parallel and distributed quantum algorithms. We demonstrate Interlin-q by implementing some of the discussed algorithms using Interlin-q and layout future steps for developing Interlin-q into a control system for distributed quantum computers.**

***Index Terms*—Distributed quantum computing, distributed quantum algorithms, quantum software, networked control systems**

I. Introduction

SCALING quantum computers up to levels where practical quantum algorithms can be executed will require a number of technological breakthroughs. In the present state of technology, scaling quantum computers past the 100 qubit mark has proven challenging [1]. Even when quantum computers can support a large number of qubits in a single system, if current methods error correction methods like surface codes are used, the amount of control signals required to perform error correction will scale with the number of qubits, potentially bottlenecking logical instructions for an algorithm's execution [2]. To overcome these obstacles, a potential solution is to instead create smaller-scale quantum computers and interlink them using a quantum network to perform quantum algorithms over a distributed system. The benefit of using smaller, interlinked quantum processors is the ability to perform larger quantum circuits on more robust and controllable quantum processors albeit with the added—potentially easier—problem of using distribution methods. When one can use networked quantum computers, an additional ability to use parallelism in algorithm design is enabled.

When moving from monolithic to distributed quantum computers, a variety of challenges arise. Indeed, there are many technological challenges to overcome towards building distributed quantum computers. A naturally arising problem to consider in this perspective is performing two-qubit operations between qubits that are physically separated between two quantum computers. To perform two-qubit operations with monolithic quantum technologies, generally the two qubits are physically near each other, and if not, swap-gates are applied to bring them near enough, known as the qubit routing problem [3]. On the other hand, for two-qubit operations between distributed qubits, one needs a new technique for transporting the control information between devices. Possible options are to physically transmit qubits via a potentially noisy and lossy medium [4], using quantum teleportation [5], [6], transferring control information to a flying qubit [7], [8], or using the method introduced in [9] using one entangled pair and a two bits of classical communication as seen in Fig. 1.

Once a method of performing non-local two qubit gates is selected, quantum circuits designed for monolithic systems need to then be remapped to a logically equivalent distributed version. To perform the remapping, one starts with the topology of the networked quantum computers, each with their own quantum processor chip structures. A monolithic circuit is converted such that any multi-qubit operation involving qubits located on different processors is replaced with a logically equivalent set of instructions orchestrating the additional tasks needed for the non-local operation. This remapping problem has been addressed in a variety of ways [10]–[15], but until distributed quantum computing becomes more standardized, the most applicable method for generating and optimizing distributed circuits remains an open problem.

The next problem arising is how to design and develop a control system for a distributed system of quantum computers. Already a step in this direction is the concept of cloud quantum computing which takes user input—usually as a circuit—and a software layer converts the input into control instructions for a single quantum computer [16], [17]. The quantum computer performs the computation and the results are sent back to the user via a communication network. For a distributed system of quantum computers, additional network connections are needed between the quantum computers. Moreover, the connections cannot simply be classical channels, but quantum channels will be needed for either distributing entanglement or moving data-containing qubits. Networked control systems for classical distributed systems have been developed in various scenarios [18], for example in GPU clusters [19], but a key problem that is not as critical for classical systems for computing is that the quantum computers need to be

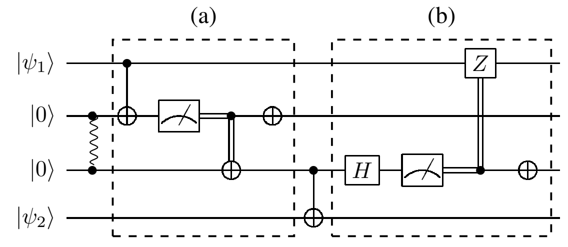

Fig. 1. Circuit diagram for a non-local CNOT gate between $|\psi_1\rangle$ and $|\psi_2\rangle$ where (a) is the cat-entangler sequence and (b) the cat-disentangler sequence. The upper two qubits and the lower two qubits are physically separated between quantum computers.

highly time-synchronized to perform joint measurements, for one. It is therefore a unique problem to design networked control systems for distributed quantum computers. Proposals addressing such control systems are found in [10], [20].

Finally, once the ability to perform distributed quantum algorithms is enabled, one can then start to consider the various quantum algorithms that can benefit from being distributed and parallelized. Such examples have been considered such as of distributed Shor's algorithm [9], Quantum Phase Estimation (QPE) [21], and accelerated Variational Quantum Eigensolver (VQE) [10]. Further, a mathematical framework for expressing and analyzing distributed quantum algorithms has been developed in [22]. Now that the hardware technology is beginning to catch up with the theory, a relatively open field remains is to better understand what advantages—especially while considering the cost of execution—there really are to gain when moving into a distributed setting.

In this work, we investigate two angles for distributed quantum computing. We consider firstly a formalization of parallel and distributed quantum programs and consider a collection of quantum algorithms fitting this formalization. Next, we introduce a novel software simulation tool for simulating distributed quantum algorithms called Interlin-q. Interlin-q is a Python library built on top of QuNetSim [23]—a quantum network simulator—which generates and simulates the control steps needed in an asynchronous setting to simulate distributed quantum algorithms. The overall goal of the platform is to provide a tool for validating algorithms for distributing quantum circuits and testing control systems. In addition, one can use Interlin-q to simulate parallel and distributed algorithms to then benchmark the approaches for their distribution and parallelization efficiency. In this work, we provide an overview of the software library in its current state and some demonstrations. Overall, interlinking quantum computers to perform distributed quantum algorithms will inevitably be an important part of quantum computing in the coming future. This work aims to shed light on the open problems and foreseeable benefits of distributed quantum computing, an increasingly important topic for the future of quantum computing.

II. Monolithic to Distributed Algorithms

To start our investigation of distributed quantum algorithms, we generalize the concept of mapping monolithic quantum algorithms to distributed quantum programs and scheduling them for execution. Executing a distributed quantum algorithm on a distributed quantum computer has a general preparation and execution stages: 1) Allocate logical qubits within the network of quantum computers; 2) Remap circuits for the possibly distributed qubit assignment; 3) Generate a schedule for the control operations; 4) Distribute and execute the schedule; and 5) Merge the outputs. Some quantum algorithms, of which we investigate in the next section, have a particular structure that allows them to gain a large "horizontal" speedups when parallelized, where other quantum algorithms requiring many logical qubits can more readily be executed on nearer-term quantum computers via a distributed quantum computer. To model this staged process of preparation and execution, we start with a QPU structure as collection of integers $Q = [q_1, ..., q_k]$ representing a network of k QPUs where each QPU i has $q_i \in \mathbb{N}$ logical qubits. In this model, it is implied that the quantum network topology is completely connected entanglement units are created during runtime. With this, we define a quantum parallel program.

Definition 1 (Parallel Program): A program P is the instruction-set needed to perform a *monolithic* execution of a quantum circuit including the logical circuit and the number of times to repeat the execution of the circuit. A schedule $S(i)$ is a mapping from an execution-round number i to sets of integers, where $|S(i)|$ is always the number of QPUs in the network. The k-th set of $S(i)$ represents the programs $\mathcal{P}_i \subset \{P_j\}_{j=1}^n$, where there are n programs total to run, executing at time i on QPU k where two distinct sets in $S(i)$ are not necessarily disjoint. A collection of programs $\{P_j\}_{j=1}^n$, a function $M : O^n \mapsto O$ for O the output of a program which acts as a central merging function, and a schedule form a parallel program $\mathcal{P} = \{\{P_1, ..., P_n\}, S(i), M\}$.

Definition 2 (Distributed Program): Given QPUs $Q = [q_1, ..., q_k]$, a distributed program dP is a program P where the circuit execution instructions of P are assigned to qubits from multiple distinct QPUs from Q. In this framework, it implies there exists an i where there are at least two distinct sets both containing P.

To generate $\mathcal{P}$, the collection of programs and schedule, Algorithm 1 is used. Input to Algorithm 1 is 1) The specifications of the distributed quantum computers $Q = [q_1, ..., q_n]$; 2) The circuit input to the program with width w, that is, the number of qubits simultaneously needed to run the circuit; 3) An algorithm $\mathcal{A}$ which takes Q as input and determines an allocation for w logical qubits or determines no allocation exists; 4) A collection of monolithic programs $\{P_i\}_{i=1}^n$. The output of the algorithm is a schedule for executing a distributed program $\{\{dP_i\}_{i=1}^n, S(i), M\}$. In Fig. 2 is a depiction of how such a system could perform.

Example 1: Let $\{P_1, ..., P_{10}\}$ be a collection of programs that run circuits with width $w = 4$ and $Q = [10, 10]$. If $\mathcal{A}$ is an algorithm that greedily allocates qubits, then the output of Algorithm 1 is: $S(0) = \{\{1, 2, 3\}, \{3, 4, 5\}\}$, $S(1) = \{\{6, 7, 8\}, \{8, 9, 10\}\}$ and $\{dP_1, ..., dP_{10}\}$, where dP_3 and dP_8 are distributed between the two QPUs and the other