Efficient Distribution of Quantum Circuits

Ranjani G Sundaram
Department of Computer Science, Stony Brook University, New York, NY, USA

Himanshu Gupta
Department of Computer Science, Stony Brook University, New York, NY, USA

C. R. Ramakrishnan
Department of Computer Science, Stony Brook University, New York, NY, USA

Abstract

Quantum computing hardware is improving in robustness, but individual computers still have small number of qubits (for storing quantum information). Computations needing a large number of qubits can only be performed by distributing them over a network of smaller quantum computers. In this paper, we consider the problem of distributing a quantum computation, represented as a quantum circuit, over a homogeneous network of quantum computers, minimizing the number of communication operations needed to complete every step of the computation. We propose a two-step solution: dividing the given circuit's qubits among the computers in the network, and scheduling communication operations, called migrations, to share quantum information among the computers to ensure that every operation can be performed locally. While the first step is an intractable problem, we present a polynomial-time solution for the second step in a special setting, and a $O(\log n)$-approximate solution in the general setting. We provide empirical results which show that our two-step solution outperforms existing heuristic for this problem by a significant margin (up to 90%, in some cases).

2012 ACM Subject Classification Computing methodologies → Distributed computing methodologies; Computing methodologies → Distributed algorithms

Keywords and phrases Distributed Quantum Computing, Hypergraph Min-Cut

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.41

Funding This work was partially supported by NSF CNS Award # 1815306.

1 Introduction

Motivation and Driving Problem. Over the recent past, there has been a steady increase in the capacity of quantum computing hardware. A crucial obstacle to achieving the full potential of quantum computing is the limited number of *qubits*, which are basic stores of quantum information, in any single quantum computer (QC). Distributing a large quantum computation over a network of QCs is a way to overcome this obstacle [7, 5]. Distributing a computation among a network of QCs will be important to realize the promise and power of quantum computation. Quantum *circuits* form a useful abstraction between higher-level quantum *programs* written in languages such as Qiskit [14] and Quipper [15], and lower-level computing hardware. In this paper, we consider the problem of *optimally distributing a given quantum circuit for evaluation over a network of QCs.* This distribution involves mapping the qubits in the circuit to individual QCs such that the communication needed to perform operations whose operands span different QCs is minimized. The distribution problem is novel to quantum computing in two important ways:

1. Quantum circuits have an explicit linear structure that makes it possible to determine the cost of a distribution before the circuit is evaluated. In contrast, classical programs have conditionals and loops where data dependencies cannot be predicted before execution.
2. *Entanglement*, which is a phenomenon unique to quantum computing, enables modes of communication that permit efficient sharing of information between operations.

The above differences enable novel solutions to the problem of distributing quantum computation (see §2 for an elaboration of these key concepts).

Efficient techniques for executing a single operation with operands that span different QCs in a network have been studied before [11, 18, 19]. More recently, [8] posed the problem of finding an optimal distribution of a quantum circuit under a naive communication model in terms of balanced graph partitioning, and solved it using well-known heuristics for that problem [12]. A formulation with a refined communication model that exploits quantum entanglement was considered in [3], where the optimization problem was shown to be intractable and solved using hypergraph partitioning heuristics [1]. A more detailed comparison with related work is in §2.3.

Our Approach and Contributions. We pose the problem of optimal distribution of quantum circuit evaluation in two steps (see overview in §3). In the first step, we partition the qubits of the circuit using ordinary balanced graph partitioning, but with the edge weights determined by accurately estimating the cost of placing the corresponding qubits in different partitions. In the second step, we solve the problem of optimal placement of operations for a given partition. The main contributions of the paper are:

- For an appropriately formulated `DQC` problem of optimizing distributed evaluation of quantum circuits, we develop an efficient heuristic that outperforms the prior work by a significant margin (up to 90%) across a wide range of parameters (see §6).
- Our algorithm for the `DQC` problem is a two-step heuristic. For the second step of covering gates using minimum number of communication primitives, we design an optimal algorithm for a specialized setting (see §4), and an $O(\log n)$-approximation algorithm for the general setting (see §5).

We defer the proofs of all lemmas and theorems to Appendix A.

2 Background and Related Works

2.1 Quantum Computation and Communication

This paper's technical development can be followed with a high-level understanding of three of quantum computing/communication concepts: the structure of quantum circuits, properties of certain quantum gates, and characteristics of quantum communication that are directly used in our development. In the following, we give an overview of these three concepts. For a more thorough review of this area, the reader is referred to standard sources such as [13].

Quantum Circuits. Quantum computation is typically abstracted as a *circuit*, where horizontal "wires" represent *qubits* which carry quantum data, and operations on the qubits performed by vertical "gates" connecting the operand wires. Quantum computers (QCs) evaluate a circuit by applying the gates in the left-to-right order, so this circuit can also be understood as a sequence of machine-level instructions (gates) over fixed number of data cells (qubits).

Analogous to classical Boolean circuits, there are several universal gate sets for quantum computation: any quantum computation can be expressed by a circuit consisting only of gates from a universal gate set. In particular, a special binary gate called CZ along with the set of all possible unary gates forms a universal gate set; we make use of this universal gate set in this paper. Fig. 1 shows the pictorial representation of an example circuit, consisting only of unary gates (boxes) and CZ gates (vertical connectors). Quantum circuits also comprise

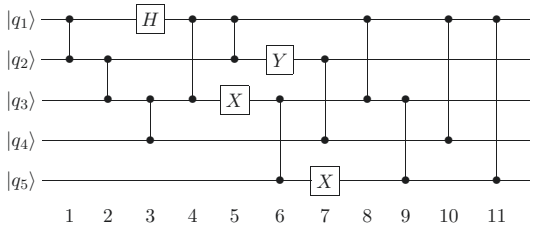


Figure 1 Running Example Quantum Circuit.

of measurement operations which yield classical bits as results, and conditional operations that are guarded by classical bit values. For the purposes of this paper, these operations can be treated as unary operations.

Quantum Communication. If a given quantum circuit is to be evaluated in a distributed fashion over a network of QCs, we have to first distribute the qubits over the QCs. But such a distribution may induce gates in the circuit to span different QCs. To execute such *non-local* gates, we need to bring all operands' values into a single QC via quantum communication. However, physical transmission of arbitrary qubits can incur *irreparable* communication errors, as the *No Cloning Theorem* [17, 9] proscribes making *independent* copies of arbitrary qubits. An alternative approach to communicate qubits is via *teleportation* [4], which requires an a priori distribution of maximally-entangled pair (MEP) of qubits (e.g., Bell Pair) over the two nodes. With an MEP distributed over nodes A and B, teleportation of a qubit state from A to B can be accomplished using classical communication and local gate operations, while consuming/destroying the MEP.

Creating "Linked Copies" of a Qubit via Cat-Entanglement. Another means of communicating qubit states is by creating *linked copies* of a qubit across QCs, via *cat-entanglement* operations [11, 19] which, like teleportation, require a Bell Pair to be shared *a priori*. These linked copies are particularly useful in efficient distributed evaluation of circuits involving only CZ and unary gates, as follows. *CZ* gates have two special properties: (i) we can freely replace an operand q with a linked copy of q; and (ii) *CZ* operations *commute* with cat-entanglement operation. As a consequence, the linked copies act like shared memories that continue to remain "in sync" across QCs as long as only *CZ* operations are executed on them. Unlike *CZ* gates, unary gates destroy the cat-entanglements in general.[1] Before applying a unary operation on q, we have to disentangle any linked copies via a dual operation called *cat-disentanglement* which doesn't require a Bell Pair.

2.2 Our Model of Distributed Evaluation of Quantum Circuits

Our model for distributed evaluation of quantum circuits is as follows.

1. Without loss of generality, we assume, that the given (centralized) circuit is composed only of unary and CZ gates. Circuits using other gate sets can be rewritten to this form, with only a polynomial expansion in size.
2. We assume that memory for Bell pairs used in cat-entanglement (called ***ebits***) are distinct from that used for computation qubits, as in prior works [3].

[1] An important exception are phase-shift gates which also preserve cat-entanglement.

3. Our network model consists of homogeneous set of QCs: the memory capacity for computation/data qubits is nearly the same in all QCs. Thus, when distributing an input circuit, we partition the number of qubits almost-uniformly across the given QCs.
4. We consider only a "static" assignment of qubits to QCs, i.e., each qubit has a home where all the unary operations are performed.
5. Due to the advantages listed in §2.1, we use cat-entanglement as the only mechanism to communicate qubits.
6. Gates in the given quantum circuit are executed in the order they appear in the circuit: no further optimizations such as gate reordering are done during distributed evaluation.

Cost Model and Objective. Let us assume that the given (centralized) quantum circuit is already optimized in the sense that successive gates on different operands are executed in parallel. For such an optimized circuit, if we disallow any gate reordering optimization (#6 above), it is easy to see that all distributed evaluation schemes would incur the same computational cost/time. Thus, our optimization objective in distribution of a given quantum circuit is to minimize the communication cost, i.e., the number of cat-entanglement operations needed to evaluate the gates. As mentioned above, cat-entanglements require a priori shared MEPs, which is an expensive resource – generating them can incur substantial latency due to the stochastic nature of underlying processes [10, 16]. Note that cat-disentanglement operations only require local operations and classical communication, and furthermore, only done following prior cat-entanglements. Hence, their cost can be ignored (or folded into the cost of cat-entanglements).

2.3 Related Work

The problem of implementing non-local gates in distributed quantum circuits was considered in [11]. That paper showed necessary and sufficient communications needed to implement several gates non-locally. In particular, they used the idea of cat-entanglement to share linked copies across QCs. Such sharing was further explored in [19], which also introduced the terms "cat-entanglement" and "cat-disentanglement". The paper generalized sharing to the multi-partite case, by essentially composing cat-entanglement operations. Both [11] and [19] focus on optimal implementation of given non-local gates.

In contrast, [8] and [3] focus on optimizing the assignment of qubits to QCs in order to minimize communication costs. A teleportation-based model is used in [8]: non-local operands of a gate are teleported to a common QC for the gate operation, and then teleported back to their original locations. The paper poses the optimization problem in terms of balanced k-partition of a weighted graph with qubits as vertices. The paper uses the graph partitioning algorithm of [12] to derive good partitions.

The closest work to ours is that of [3], where cat-entanglement operations are used to share linked copies of qubits. The optimal assignment problem is posed in terms of balanced *hypergraph* partitioning, with hyperedges capturing the set of gates that can be executed with linked copies. The assignment problem is shown to be intractable via reduction from the hypergraph balanced k-min-cut problem. In contrast, we have developed a two-step heuristic, which uses a simpler min-cut problem in the first step and a coverage problem in the second step that can be solved optimally or near-optimally. We compare our work empirically with theirs in §6.

3 DQC Problem Formulation and High-Level Algorithm

In this section, we formally define the problem of efficient distribution of quantum circuits, and give a high-level description of our proposed algorithm.

Quantum Circuits. We use $q_1, q_2, \ldots, q_n$ to denote *qubits*. Quantum circuits are composed of gates. Based on §2, we consider the universal gate set with (binary) CZ and unary gates. Since we only use one type of binary gate and the *type* of unary gate does not play a role in our context, we need to only represent the operands of each gate and not the gate itself. We use the below notion of an *abstract* quantum circuit that elides the gate information.

▶ **Definition 3.1** ((Abstract) Quantum Circuit). *Given a set of qubits $Q = \{q_1, q_2, \ldots\}$, an* abstract quantum circuit *C over Q is a sequence of gates $\langle g_1, g_2, \ldots\rangle$ where each g_k is one of:*

- *(q_i, q_j): the pair of operands for a CZ gate that occurs as the k-th gate.*
- *q_i: the operand of an unary gate that occurs as the k-th gate.*

Throughout the article, we thus represent binary gates in a circuit as triplets (q_i, q_j, t), and unary gates as pairs (q_i, t), where t is the index/timestamp of the gate in the circuit.

3.1 Distributed Quantum Circuit (DQC) Problem

Our goal is to determine an efficient distribution of a given quantum circuit, over a given network of QCs. Efficient distribution essentially entails two tasks: distributing the qubits over the distributed QCs, and then executing the gates efficiently over the distributed QCs. We represent the distribution of qubits over the distributed QCs as a *partition* of the qubits. We also define a concept of *migrations* which represent the cat-entanglement operations; each migration may facilitate execution of one or more gates, which is captured via a notion of coverage of gates by migrations.

Informally, the DQC problem is to (i) partition the given circuit across distributed QCs, and (ii) for the given partition, determine a small set of migrations that are sufficient to execute/cover all the non-local gates in the circuit. The overall goal of the DQC problem is to minimize the number of cat-entanglements (or migrations, as we represent them as). Below, we formalize the notions of partitions, migrations, and execution of gates by migrations (via a notion of coverage), before defining the DQC problem formally.

Partitioning. Distributing a quantum circuit first entails assigning qubits to quantum computers in the network. In this paper, we implicitly assume homogeneous quantum computers, and thus, we consider only near-balanced partitions of the qubits across QCs.

▶ **Definition 3.2** (Partition). *Given integer $k > 0$ and real $\nu > 1$, a balanced (k, ν)-partition of a finite set S divides S into k components i.e., disjoint subsets of S, such that each component is of size at most $\nu \cdot \frac{|S|}{k}$.*

A partition can be represented by a total function π that maps S to a set of labels $P = \{p_1, p_2, \ldots, p_k\}$.

For our DQC problem, we consider (k, ν)-balanced partitions of the set of qubits Q using the set of QCs, $P = \{p_1, p_2, \ldots, p_k\}$, as the partition labels. Note that such a partition assigns qubits to QCs. Throughout this paper, we refer to quantum computers p_i as partitions, and $\pi(q)$ as the *home-partition* of q. In addition, we refer to a quantum circuit with an already given partition as a *partitioned circuit*.

Migrations. A migration essentially represents the cat-entanglement operation to create a *linked* copy of a qubit in a partition other than its home-partition. We allow migrations of a qubit q_i to occur only initially (i.e., $t = 0$) or right after a unary operation on q_i. This is without loss of generality, because, since cat-entanglement commutes with CZ, we can move any migration to the most recent unary operation or $t = 0$. Also, disentanglement operations are implicit – done, if needed, immediately before any unary operations, so are not represented.

▶ **Definition 3.3** (Migration). *Given a set of qubits Q, abstracted circuit C and partition P, a* migration *is a triple (q_i, p_k, j) where $q_i \in Q$, $p_k \in P$ such that $p_k \neq \pi(q_i)$ (i.e. not the home-partition of q_i), and either $j = 0$ or $(q_i, j) \in C$, i.e., j is the index of an unary gate on q_i in C.*

Coverage of Gate by Migration(s). Consider a binary gate (q_i, q_j, t), where $\pi(q_i) \neq \pi(q_j)$ i.e., the operands are in different partitions. We refer to such gates as *non-local* gates. To execute such a non-local binary gate, we can: (i) Migrate q_i to the home partition of q_j, (ii) Migrate q_j to the home partition of q_i, or (iii) Migrate both q_i and q_j to a third partition where the gate operation can then be performed. In each of the above cases, since unary gates do not commute with migrations, we need to ensure that there are no unary gate operations on the migrated qubit between the migration and the binary gate operation being covered. This notion is formalized below, in terms of coverage of a gate via migrations.

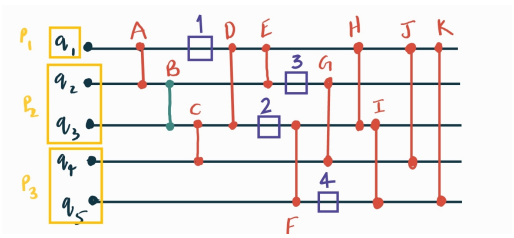
▶ **Definition 3.4** (Coverage). *For a given partition π, a binary gate (q_i, q_j, t) can be covered by a single or a pair of migrations in one of the following ways.*

1. *By $(q_i, \pi(q_j), t')$, if $t' < t$ and there are no unary gates on q_i between t' and t; or*
2. *By $(q_j, \pi(q_i), t')$, if $t' < t$ and there are no unary gates on q_j between t' and t; or*
3. *By the pair $\{(q_i, p_k, t'), (q_j, p_k, t'')\}$, if $t', t'' < t$ and there are no unary gates on q_i between t' and t and on q_j between t'' and t.*

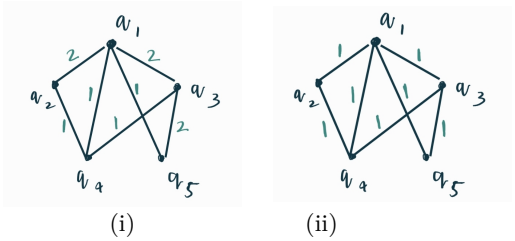
Note that the third condition above allows a gate (q_i, q_j, t) to be executed at a partition other than the home-partitions of either of the operand qubits (i.e., by migrating *both* the operand qubits to a third partition.

DQC Problem Formulation. Given a quantum circuit, the number k of distributed QCs, and a partitioning factor $\nu \geq 1$, the `DQC` problem is to (k, ν)-partition the qubits into the distributed QCs and determine the set of migrations that cover the non-local gates of the partitioned circuit, such that the total number of migrations required is minimized. The `DQC` problem is known to be NP-hard by a reduction from the hypergraph min-cut problem [3].

DQC Example. Consider the running example of our quantum circuit from Fig. 1. Fig. 2a shows the distribution of qubits into three partitions: $p_1 = \{q_1\}$, $p_2 = \{q_2, q_3\}$ and $p_3 = \{q_4, q_5\}$. The non-local binary gates are represented by red vertical lines, and the local ones by green lines. Let $t_i (i \geq 1)$ be the timestamp associated with the i-th unary gate. The set of all possible migrations for this partitioned circuit is: $\{(q_1, p_2, 0), (q_1, p_3, 0), (q_2, p_1, 0), (q_2, p_3, 0), (q_3, p_1, 0), (q_3, p_3, 0), (q_4, p_1, 0), (q_4, p_2, 0), (q_5, p_1, 0), (q_5, p_2, 0), (q_1, p_2, t_1), (q_1, p_3, t_1), (q_3, p_1, t_2), (q_3, p_3, t_2), (q_2, p_1, t_3), (q_2, p_3, t_3), (q_5, p_1, t_4), (q_5, p_2, t_4)\}$. A example of a set of migrations that can cover all the non-local gates is: $\{(q_1, p_2, 0), (q_4, p_2, 0), (q_5, p_2, 0), (q_1, p_2, t_1), (q_5, p_2, t_4)\}$. This set of migrations execute all the non-local gates in partition p_2.



(a) Partitioning of our running example quantum circuit; the three partitions are shown by yellow boxes.



(b) Edge weights on the graph used for partitioning the running example. (i) Simple weights based on the total number of binary gates between the qubits, and (ii) Refined edge weights based on the optimal `MS-HC` algorithm in §4.

Figure 2 Partitioning and Edge Weights, on the running example.

3.2 Two-Step DQC Algorithm

In a recent work, Martinez and Heunen [3] reduce the `DQC` problem to a balanced k-min-cut of an appropriate hypergraph, and thus, use a hypergraph-partitioning heuristic to solve the problem. Here, we propose a use natural two-step procedure – in effect, reducing the `DQC` problem to a sequence of two simpler problems; in fact, we show that the second step can be solved near-optimally and even optimally in a special setting. In particular, our proposed algorithm consists of the following two steps: (i) First, we compute a "good" (k, ν)-partition of the qubits for distribution over the k QCs; (ii) Then, we determine a minimal set of migrations required to cover all the non-local gates of the partitioned circuit.

Step 1: Partitioning Qubits into QCs/Partitions. To compute a partition that will intuitively yield a small set of migrations in the second step, we compute the partitioning of the qubits by solving a balanced k-min-cut problem over an edge-weighted graph over qubits as vertices. In particular, given a circuit C, we define a weighted graph $G = (V, E)$ where V is the set of qubits in C and $E = \{(q_i, q_j) | (q_i, q_j, t) \in C\}$. The weight on each edge intuitively represents the cost of keeping the qubits in different partitions; at its simplest, the weight of an edge (q_i, q_j) can be the number of binary gates of the type $(q_i, q_j, \_)$ in C for some t. A more refined weight function is defined at the end of this section.

In essence, in the first step, we partition the qubits by computing a balanced min-k-cut of the of the above weighted graph. The balanced min-k-cut problem is NP-hard even for $k = 2$, with no known approximation algorithms. Therefore, we consider an approximate version of the problem, viz., (k, ν)-balanced graph partitioning for $\nu > 1$ as defined in Def. 3.2. There are several works that address the (k, ν)-balanced min-cut problem. Notably, [2] gives a $O(\log^2 n)$-approximation algorithm for $\nu = 1 + \epsilon$ (ϵ is arbitrarily small) when k is a constant. In our evaluation, we use a third-party graph partitioning algorithm called KaHyPar [1].

Step 2: Selecting Migrations. Given a partitioned circuit, we then find the smallest set of migrations that cover all the non-local gates. This minimization problem can be solved optimally in polynomial time when restrict each gate to be executed only in the home-partition of one of its operands (§ 4). For the general case, we provide an $O(\log N)$-approximation algorithm (where N is the number of binary/CZ gates in the circuit C).

Refined Weight Function in Step 1. The total number of CZ gates between two qubits q_i and q_j, originally proposed as the weights of edges in Step 1 above, is actually just an upper bound on the number of migrations needed to cover those gates. A more accurate cost will be the actual/optimal number of migrations needed to cover $(q_i, q_j, \_)$ gates if q_i and q_j were in separate partitions. We use the above intuition to construct an *induced* circuit $C'_{(i,j)}$ consisting only of gates on q_i and q_j. We distribute $C'_{(i,j)}$ such that the two qubits q_i and q_j are in different partitions and compute the *minimum* number of migrations needed to cover all the binary gates in $C'_{(i,j)}$. The minimum number of migrations in $C'_{(i,j)}$ can be computed using the optimal `MS-HC` algorithm described in §4, and we use this minimum value as the weight on the edge (q_i, q_j) of the weighted graph used in Step 1. See Fig. 2b.

4 Optimal Selection of Migrations under Home-Coverage

In this section, given a quantum circuit and a partitioning of its qubits to computers/partitions, we design an efficient algorithm to compute the optimal set of migrations needed to execute the partitioned circuit. The algorithm developed in this section is used as a subroutine in the Two-Step `DQC` Algorithm presented in §3.

Here, we will assume that a gate can only be executed at the home-partition of one of its operand-qubits; this restriction helps minimize execution memory needed at individual partitions, and in addition, simplifies the migration-selection problem sufficiently that we can design an optimal algorithm for the problem of selection of migrations to cover all gates. We will relax this assumption in the next section where we describe an algorithm for the migration-selection problem in the more general setting.

We start with formally incorporating the above assumption in our problem formulation by restricting the original definition (Def. 3.4) of coverage of gates by migrations.

Home Coverage of Gates by Migrations. Given a quantum circuit, recall the definition (Def. 3.4) of coverage of gates by migrations, for a given partition π. Therein, we defined that a gate can be covered by migration(s) in many ways. To impose the condition that each gate be executed only at a home-partition of its operand qubits, we restrict the definition of coverage by allowing a migration to cover a gate only if it migrates one of the operand qubits to the home-partition of the other qubits. We define the notion of home-coverage below.

▶ **Definition 4.1** (Home-Coverage). *For a given partition π, a binary gate gate (q_i, q_j, t) can be home-covered by a migration in one of the following ways.*

1. *By $(q_i, \pi(q_j), t')$, if $t' < t$ and there are no unary gates on q_i between t' and t; or*
2. *By $(q_j, \pi(q_i), t')$, if $t' < t$ and there are no unary gates on q_j between t' and t.*

Memory Conservation by Home-Coverage. Restricting execution of gates at home partitions of its operands is one way to implicitly minimize/constrain the size of execution memory needed. For example, consider a circuit with n qubits $\{q_1, q_2, \ldots, q_m\}$, no unary gates, and $n-1$ binary CZ-gates of the type (q_i, q_{i+1}, t_i) for some t_i's. Suppose that each qubit is distributed to a different partition. In this case, a potential set of migrations to cover the gates could be to migrate all qubits to a *single* partition and execute all the gates there. However, this would require $n-1$ units of additional memory to hold all the migrated qubits. In contrast, the alternate solution restricted by home-coverage would migrate qubit q_i to $\pi(q_{i+1})$ for each i, requiring only one additional memory at each partition. In addition, as mentioned above, restricting the coverage of gates to home-coverage allows design of an optimal algorithm for selection of migrations, and thus, an efficient overall solution for the `DQC` problem as observed in our evaluations.

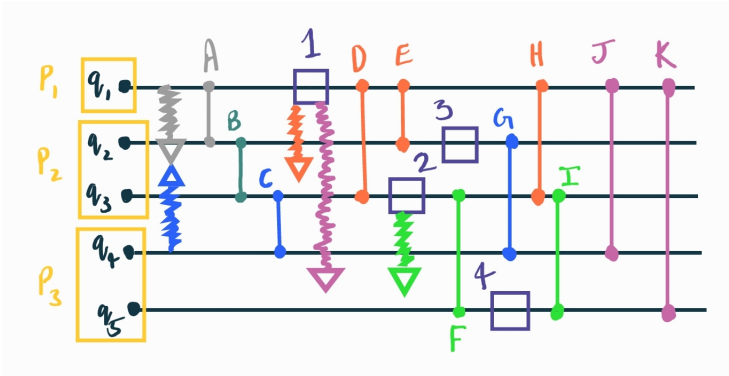


Figure 3 Illustrating the MS-HC Problem. The set of gates (straight vertical lines) covered by a migration (squiggly lines) m are shown in the same color as m. E.g., gates D, E and H are covered by the migration (q_1, p_2, t_1).

MS-HC Problem. Migration Selection under Home-Coverage. Given a quantum circuit C and a partitioning of C's qubits to given partitions/QCs, the MS-HC problem is to determine the smallest set of migrations that can execute the gates of the partitioned circuit in the home partitions of one of its operands. More formally, let Q be the set of qubits in a given circuit C, P be the set of given partitions, π be the partitioning function, and M be the set of all potential migrations (as per Def. 3.3). The MS-HC problem is to select a smallest set of migrations M such that each gate in C is home-covered by some migration in M.

We start with illustrating the MS-HC problem via an example. Then, we design an optimal algorithm for the MS-HC problem. Recall our running example of a quantum circuit from Example 2a. As shown in Fig. 3, one solution to the MS-HC problem for the given partition of the qubits is $\{(q_1, p_2, 0), (q_4, p_2, 0), (q_1, p_2, t_1), (q_1, p_3, t_1), (q_3, p_3, t_2)\}$, where as before t_i is the timestamp of the i-th unary gate. This solution can be easily verified as optimal.

4.1 Optimal MDS-HC Algorithm

We start with a high-level description and intuition of the proposed optimal algorithm, followed by the proofs of two key claims needed to design the algorithm.

High-level. The MS-HC problem can be easily formulated as the well-known `set-cover` problem. However, MS-HC is in fact a very special case of the `set-cover`, allowing us to design a polynomial-time optimal algorithm. In particular, first, we show that each gate (element) is home-covered by exactly two migrations (sets), and thus the MS-HC problem can be formulated as a `vertex-cover` problem in an appropriate graph G_{HC}, where migrations are represented as nodes and gates as edges. Second, we show that the graph G_{HC} is actually a bipartite graph, by showing that it has no odd length cycles. Thus, the MS-HC problem reduces to the vertex-cover problem in the bipartite graph G_{HC}, and thus, solvable optimally in polynomial time.

Each Gate is Home-Covered by Two Migrations. Recall from Def. 3.3 that a migration is a triplet (q, P, t), where the qubit q is being migrated to partition P at time t, and we only allow migrations with a timestamp t of either 0 or corresponding to some unary gate on qubit q. The below lemma shows that each gate is home-covered by exactly two such migrations.

▶ **Lemma 4.2.** *For a given partitioned quantum circuit, a binary gate in the circuit is home-covered by exactly two migrations.*

It is easy to see that the above lemma holds for the partitioned circuit of our running example. See Figure 5 in Appendix A.

Bipartite Graph over Migrations. Based on the intuition from Fig. 5, we can construct a graph $G_{HC}(V = M, E = T)$ where T is the set of all non-local gates (i.e., gates with different home-partitions of the operands) in the given partitioned circuit and M is the set of all migrations as per Def. 3.3 (i.e., with a timestamp of 0 or a unary gate on the migrated qubit). More formally, an edge/migration (q_i, q_j, t) connects the unique vertices $(q_i, \pi(q_j), t_i)$ and $(q_j, \pi(q_i), t_j)$ where t_i and t_j are as defined in Lemma 4.2. Based on this graph representation of home-coverage of gates by migrations, we can solve the `MS-HC` problem by computing the optimal vertex-cover of edges in G_{HC}, since a vertex cover of G_{HC} corresponds exactly to a set of migrations that home-cover all the given gates. Below, we prove that, the G_{HC} graph is actually a bipartite graph – which allows us to compute the optimal vertex-cover of G_{HC} in polynomial time.

▶ **Lemma 4.3.** *For a given partitioned circuit, we claim that the G_{HC} graph, as defined above, has no odd-length cycles, and is thus a bipartite graph.*

Optimal Algorithm for MS-HC. Based on the above two lemmas, we can now solve the `MS-HC` problem by computing an optimal vertex cover of the bipartite graph G_{HC} for a given partitioned circuit. The pseudo-code of the overall algorithm is given in Appendix A.4.

▶ **Theorem 4.4.** *Given a partitioned quantum circuit, Algorithm 2 returns an optimal set of migrations that home-cover all the non-local gates of the given partitioned circuit.*

5 Near-Optimal Selection of Migrations under General Coverage

In this section, we relax the assumption made in the previous section, i.e., allow execution of non-local gates of a partitioned circuit in partitions different from any of the home-partitions of its operands. We do this by using the original notion of coverage defined in Def. 3.4, where a non-local gate may also be executed by migrating both of its operand qubits to a third partition. In such a general setting, for a given partitioned circuit, we consider the problem of selecting a minimum number of migrations to cover all non-local gates, and present a polynomial-time approximation algorithm.

MS-GC Problem: Selection of Migrations to Cover Gates. Given a partitioned quantum circuit, the `MS-GC` problem is to determine a set a migrations of minimum size that covers all the non-local binary gates of the given partitioned circuit.

The above `MS-GC` problem generalizes the `set-cover` problem in some sense, as it allows an element to be covered by a pair of sets (i.e., an element is covered only if both the sets are selected). However, the `MS-GC` problem also has a special structure, which makes proving its intractability non-trivial; however, we conjecture the `MS-GC` problem to be NP-hard. In either case, since the objective function is not submodular, the simple greedy algorithm that iteratively selects the migration with most "benefit" does not offer a performance guarantee. Below, we will design a polynomial-time approximation algorithm for the `MS-GC` problem. We start with a definition.

▶ **Definition 5.1** (Same-Partition Set (of Migrations)). *A set of migrations M is called a* same-partition set *if for every pair of migrations (q_1, p_1, t_1) and (q_2, p_2, t_2) in M, we have $p_1 = p_2$.*

5.1 Algorithm G*: Approximation Algorithm for the MS-GC Problem

Our proposed algorithm G^* is a greedy algorithm that picks, at each iteration, a same-partition set of migrations with highest benefit-density (as defined below), until all the non-local gates of the given partitioned circuit are covered by the picked set of migrations. Note that, at each stage, the G^* algorithm may pick more than one migration. We will later develop a procedure (Algorithm 1) to select a same-partition set with highest benefit-density, which form a single iteration of the G^* algorithm. Below, we first show that G^* will deliver an $O(\log N)$-approximate solution to the `MS-GC` problem, where N is the total number of non-local gates in the given partitioned circuit. We now define the notion of benefit and benefit-density of a set of migrations.

▶ **Definition 5.2** (Benefit; Benefit-Density). *Consider a stage/iteration in the G^* algorithm, where a set of migrations have already been selected. For a set X of migrations, we define its* benefit $B(X)$ *as the number of (non-local) gates covered by X that have not been covered yet by the set of migrations already selected in previous iterations. We define the* benefit-density *of X as $B(X)/|X|$).*

▶ **Theorem 5.3.** *For the `MS-GC` problem, the G^* algorithm delivers a solution with at most $|O| \ln N$ number of migrations, where O is the optimal solution and N is the total number of non-local gates in the given partitioned circuit.*

5.2 Dividing a Set into Same-Partition Subsets

We now show that a set of migrations can be divided into a disjoint collection of same-partition subsets such that the benefit of the original set is at most the summation of the benefit of the subsets. We start with a formal definition and on observation.

▶ **Definition 5.4** (Benefit Graph). *The benefit-graph, denoted by $B_M(V, E)$, for a set of migrations M at a certain stage of G^* is defined as follows. The set of vertices $V(B_M)$ is the set of migrations in M, and each node/migration m in $V(B_M)$ is assigned a node-weight of $B(m)$. Consider a pair of migrations m_1, m_2 in M. The pair of vertices (m_1, m_2) are connected by an edge in B_M if and only if $B(\{m_1, m_2\})$ is non-zero, in which case we also assign a weight of $B(\{m_1, m_2\})$ to the edge (m_1, m_2).*

▶ **Lemma 5.5.** *At any stage of the G^* algorithm, a set O of migrations can be divided into disjoint subsets $O_1, O_2, \ldots, O_l$ such that $B(O) \leq \sum_{i=1}^{l} B(O_i)$.*

5.3 G* Iteration: Selecting an Optimal Same-Partition Set

We now design an algorithm to select the same-partition set of migrations with highest benefit-density, at a given stage/iteration of the G^* algorithm. Let us consider a stage of the G^* algorithm where a set of migrations have already been selected. Our goal is to pick a set $\mathcal{M}$ of same-partition migrations from the remaining migrations such that $B(\mathcal{M})/|\mathcal{M}|$ is maximized, where $B(\mathcal{M})$ is the benefit of $\mathcal{M}$ at the given stage of the G^* algorithm. Our overall algorithm of selecting an optimal $\mathcal{M}$ consists of the following steps.

1. For each given partition p_i, let M_i be the set of remaining (i.e., not yet selected by G^*) migrations that migrate a qubit to p_i. Note that each M_i is a maximal set of same-partition migrations.
2. For each M_i, we find a set $\mathcal{M}_i \subseteq M_i$ with highest benefit-density, as described later.
3. From the above $\mathcal{M}_i$'s, We pick the $\mathcal{M}_i$ with highest $B(\mathcal{M}_i)/|\mathcal{M}_i|$ as the optimal $\mathcal{M}$.

It is easy to see that the above returns an optimal same-partition set of migrations, as any same-partition set of migrations must be a subset of M_i for some i.

Selecting a Subset of M_i with Highest Benefit-Density. We now discuss how to select an optimal subset within a given M_i . We start with a lemma, which will help reduce the problem to that of selecting an optimal induced subgraph in the benefit graph of M_i.

▶ **Lemma 5.6.** *Let M be a same-partition set of migrations. Then, $B(M) = \sum_{m\in M} B(\{m\}) + \sum_{m_1,m_2\in M} B(\{m_1, m_2\})$.*

The above lemma implies that finding the best subset within a given M_i is equivalent to finding an induced subgraph H in the benefit-graph of M_i that has the highest density of weight, i.e., maximum value of $(\sum_{e\in E(H)} w(e) + \sum_{v\in V(H)} w(v))/|V(H)|$. This is a weighted generalization of the well-studied *densest subgraph problem* which can be solved optimally in polynomial time. We discuss this below.

Weighted Densest Subgraph Problem. Given an unweighted graph G, the *densest subgraph problem* is to select an induced subgraph H in G such that $|E(H)|/|V(H)|$ is maximized. This problem can be solved optimally in polynomial time [6]. We are interested in the weighted generalization of this problem referred to as the *weighted densest subgraph* problem, wherein vertices and edges have (positive) weights associated, and the goal is to select an induced subgraph H with maximum $(\sum_{e\in E(H)} w(e) + \sum_{v\in V(H)} w(v))/|V(H)|$. The simple LP-based optimal algorithm as well as the more time-efficient 2-approximation algorithm for the unweighted version given in [6] can be both easily generalized to our weighted version of the problem. We briefly give both the algorithms, and defer the performance guarantee proofs (as they are simple generalizations of the proofs for the unweighted case in [6]).

LP-based Optimal Algorithm. The weighted densest subgraph problem can be easily represented as an ILP; the corresponding LP is as follows. Here, w_i is the weight of vertex i, and w_{ij} is the weight of edge (i, j) in the given graph.

$$\text{Maximize} \quad \sum_i y_i w_i + \sum_{ij} x_{ij} w_{ij}$$

$$\text{Subject to:} \quad (i)\ \ 0 \le y_i, x_{ij} \le 1; \quad (ii)\ \ x_{ij} \le y_i; \quad (iii)\ \ \sum_i y_i \le 1; \quad (iv)\ \ x_{ij} \le y_j.$$

The optimal solution is obtained by: (i) Solving the above fractional LP; let the solution be $\{\overline{y_i}, \overline{x_{ij}}\}$. (ii) Picking a threshold τ appropriately and setting $\overline{y_i} = \lfloor \overline{y_i}/\tau \rfloor$. In (ii), we pick a threshold the LP objective; we do so by exhaustively trying all $\overline{y_i}$ values as the threshold. It can be shown by a simple generalization of the proof for the unweighted version [6] that the above process returns an optimal solution for the weighted densest problem.

2-Approximate Greedy Algorithm. A simple greedy algorithm to solve the weighted densest subgraph problem is to iteratively remove a vertex with the lowest sum of node weight and weight of incident edges, keep track of the resulting n subgraphs over n interations, and picking the best among them. This greedy algorithm can be easily shown to be 2-approximate, by a simple generalization of the proof for the unweighted version [6].

Overall G* Iteration. Algorithm 1 gives the pseudo-code of the overall algorithm for a single G^* iteration, which selects the same-partition set with highest benefit-density.

Algorithm 1 Single Iteration of G^* Algorithm.

Input: The set of remaining migrations M, at a certain stage of G^* Algorithm.
Output: A set of same-partition migrations $\mathcal{M}$ in M, with the highest benefit-density.

1: Let $M_1, M_2, \ldots, M_k$ be the disjoint and maximal same-partition subsets of M corresponding to each of the partitions $p_1, p_2, \ldots, p_k$.
/* For each M_i, find a subset of M_i with highest benefit-density */.
2: **for all** $i \leq k$ **do**
3: Construct the benefit-graph $B(M_i)$ of M_i.
4: Find the *weighted densest subgraph* H_i in $B(M_i)$.
5: $\mathcal{M}_i \leftarrow$ Vertices in H_i
6: **end for**
7: $\mathcal{M} \leftarrow$ The $\mathcal{M}_i$ with highest benefit-density.
8: **return** $\mathcal{M}$.

6 Evaluation

We now evaluate our algorithms empirically over random quantum circuits. The goal of our empirical study is to evaluate the performance of proposed techniques in terms of number of migrations incurred, i.e., the number of cat-entanglements, and compare them with the prior approach from [3].

Algorithms Compared. We compare our techniques with the algorithm proposed in [3], which is based on computing a min-cut in an appropriate hypergraph; we refer to this algorithm as `Martinez-19`. Our algorithms are all based on the Two-Step Algorithm of §3, with just different subroutines (from §4-5) for the second step. For the first-step of partitioning the qubits using a simple weighted graph, we use a third-party solver KaHyPar [1] and the refined weighted scheme discussed in §4. For the second-step, we use three different schemes and refer to the overall `DQC` algorithms as follows: (i) `Home-Cover` algorithm uses Algorithm 2 in the second step, (ii) G^*`_LP`, G^*`_Approx`, G^*`_Simple` algorithms use Algorithm G^* in the second step, with the LP-based, 2-approximation greedy, and simple greedy algorithms respectively for solving the weighted densest subgraph problem. The simple greedy algorithm to compute the weighted densest subgraph simply removes one vertex at a time to improve the weighted-density of the remaining graph, and stops when the remaining graph's weighted density can't be improved by removing any vertex. All our algorithms use the refined weights described in §3.2; usage of refined weights yielded a performance advantage of around 6% compared to using the simple weights.

Random Circuit Inputs and Parameter Values. We run our simulations over randomly generate quantum circuits. To generate quantum circuit instances, we vary the following parameters: number of qubits, total number of gates per qubit, fraction of gates that are binary (CZ) gates, and number of given partitions. We use an imbalance-factor ν of 1.1 for all the experiments. In the below plots, we vary one of the parameters and fix the remaining three to their default values. The default value for number of qubits is 50, total number of

gates per qubit is 50, and number of partitions is 10. For the fraction of binary gates, we use two default values: 50% and 80%, as this parameter has a strong impact on performance of algorithms and in practice, the fraction of binary gates can be high.[2] Each data point in the below experiments is obtained from an average of 5 different random instances.

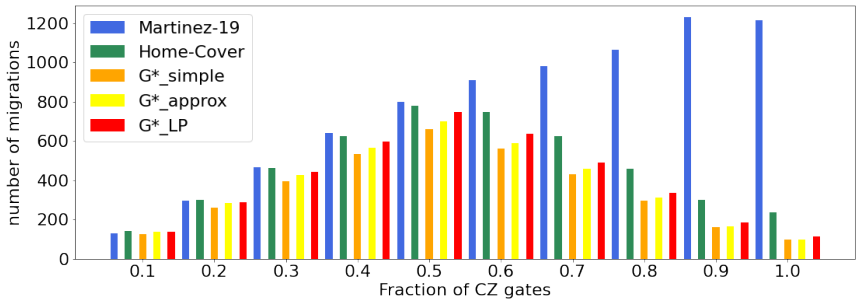
Evaluation Results for Varying Parameters. We present our results in Figures 4b-4f. In general, we observe that all the three G^*-based algorithms performing similarly and significantly outperforming the `Martinez-19`, across all our experiments. In particular, the G^*-based algorithms perform up to 90% better (i.e., incurring merely 20% of the migrations/cat-entanglements used by `Martinez-19`; see Fig. 4a). The `Home-Cover` algorithm, which implicitly conserves execution memory usage at any single partition, also performs up to 80% better than `Martinez-19`. Among the G^*-based algorithms, surprisingly the G^*`_Simple` performs slightly better than the other two; note that, this does not contradict the optimality of LP-based algorithm for the densest weighted subgraph, since densest weighted subgraph solution is only used within an iteration of the G^* algorithm.

Figure 4a plots results for varying fraction of binary (CZ) gates in the circuit. We observe that the performance of all our algorithms, unlike `Martinez-19`, improves significantly with increasing fraction of binary gates; this can be attributed to the fact that higher fraction increases the number of gates covered by a single migration, and our algorithms are able to take better advantage of it. Figures 4b and 4c show the performance of various algorithms for increasing number of qubits, with 50% and 80% CZ gates respectively, while using default values for other parameters. The performance of our algorithms in comparison to `Martinez-19` improves with increasing number of qubits. Figures 4d and 4e plot results for varying number of partitions. Here, we observe that performance of `Home-Cover` wrt G^*-based algorithms worsens with increase in partitions; this may be due to the fact that `Home-Cover`'s restriction of home-partition execution becomes more pronounced with increase in number of partitions. Finally, Figures 4f and 4g plot the performance of algorithms for varying number of gates per qubit. We discuss execution memory usage in Appendix A.8.

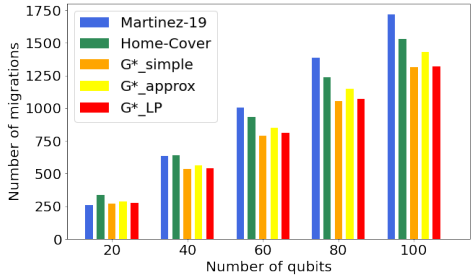
7 Conclusions

In this paper we considered the problem to distribute a quantum circuit over a network of QCs, such that the communication cost minimized. We presented a two-step heuristic that outperforms prior techniques. In future work, we plan to look at various generalizations of the problem, e.g., for nodes with non-uniform capacities, links with non-uniform communication costs, constraining the execution memory at each partition, allowing multiple modes of communications (e.g., teleportation as well as cat-entanglement), allowing for unary gates to be executed at any partition (i.e., allowing for dynamic home-partition of a qubit). In addition, we plan to investigate better heuristics for the first step of our algorithm.

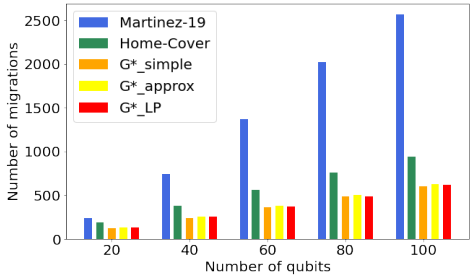
[2] E.g., the Quantum Fourier Transform (QFT) circuit has has $O(n^2)$ controlled-phase gates and $O(n)$ other gates; since the controlled-phase gates can be treated like CZ gates, the fraction of binary gates in a QFT circuit can be arbitrarily high. Also, when we convert binary gates in an arbitrary circuit to CZ gates, the fraction of binary gates is expected to be 50% as a binary gate yields a CZ gates flanked by unary gates (and long runs of unary gates can be considered as a single unary gate, in our context).



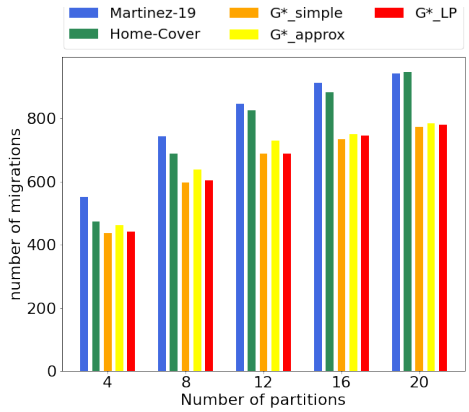
(a) Number of migrations used for various random circuits with varying fraction of binary gates.



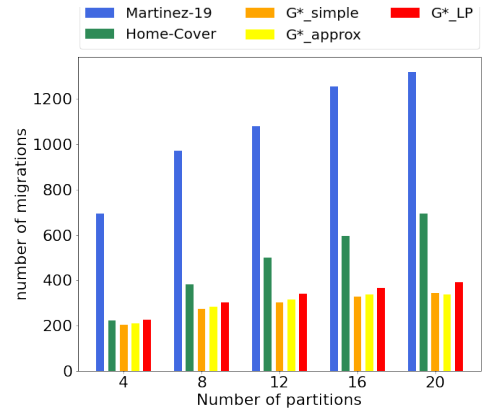
(b) Varying number of qubits, for Frac-CZ=0.5.



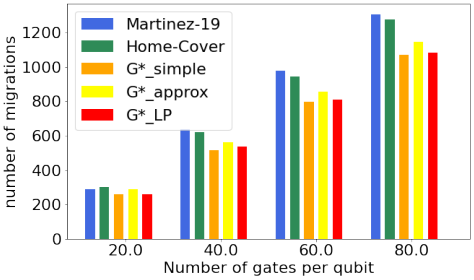
(c) Varying number of qubits, for Frac-CZ=0.8.



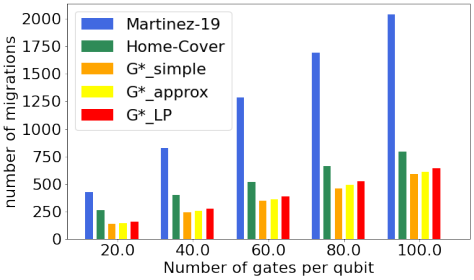
(d) Varying number of partitions, for Frac-CZ=0.5.



(e) Varying number of partitions, for Frac-CZ=0.8.



(f) Varying gate ratio, for Frac-CZ=0.5.



(g) Varying gate ratio, for Frac-CZ=0.8.

Figure 4 Performance of various algorithms for varying fraction of CZ gates ((a)), varying number of qubits ((b)–(c)), varying number of partitions ((d)–(e)), and varying number of gates per qubit ((f)–(g)). Above, Frac-CZ (fraction of CZ gates) is 0.5 in (b), (d), and (f), and is 0.8 in (c), (e), and (g).

References

1 Yaroslav Akhremtsev, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Engineering a direct k-way hypergraph partitioning algorithm. In *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2017.

2 Konstantin Andreev and Harald Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6), 2006.

3 P. Andres-Martinez and C. Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Phys. Rev. A*, 100(3), 2019.

4 C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein–Podolsky–Rosen channels. *Phys. Rev. Lett.*, 70(13), 1993.

5 M. Caleffi, A. S. Cacciapuoti, and G. Bianchi. Quantum internet: from communication to distributed computing!, 2018.

6 Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 2000.

7 J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello. Distributed quantum computation over noisy channels, phys. rev. a. *Phys. Rev. A*, 59(4249), 1999.

8 O. Daei, K. Navi, and M. Zomorodi-Moghadam. Optimized quantum circuit partitioning. *Int. J. Theor. Phys.*, 59(12):3804–3820, 2020.

9 D. Dieks. Communication by EPR devices. *Physics Letters A*, 92(6), 1982. doi:10.1016/0375-9601(82)90084-6.

10 L.-M. Duan, M. D. Lukin, J. I. Cirac, and P. Zoller. Long-distance quantum communication with atomic ensembles and linear optics. *Nature*, 414(6862), November 2001. doi:10.1038/35106500.

11 J. Eisert, K. Jacobs, P. Papadopoulos, and M.B. Plenio. Optimal local implementation of non-local quantum gates. *Phys. Rev. A*, 62(052317-1), 2000.

12 B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2), 1970.

13 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

14 Qiskit. https://qiskit.org/.

15 Quipper. https://www.mathstat.dal.ca/~selinger/quipper/doc.

16 Nicolas Sangouard, Christoph Simon, Hugues De Riedmatten, and Nicolas Gisin. Quantum repeaters based on atomic ensembles and linear optics. *Reviews of Modern Physics*, 2011.

17 W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886), 1982. doi:10.1038/299802a0.

18 A. Yimsiriwattana and S. J. Lomonaco Jr. Distributed quantum computing: A distributed Shor algorithm. *Quantum Information and Computation II*, 5436, 2004.

19 A. Yimsiriwattana and S. J. Lomonaco Jr. Generalized GHZ states and distributed quantum computing. *AMS Cont. Math.*, 381(131), 2005.

A Appendix A

A.1 Illustrating Lemma 4.2 over Running Example

See Figure 5.

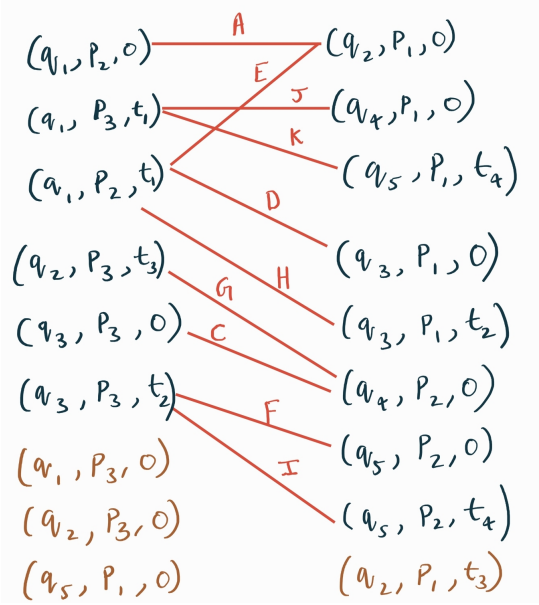


Figure 5 The G_{HC} graph connecting migrations with the home-covered binary gates, for the partitioned circuit of the running example. Here, each binary gate is represented by an edge that connects the two migrations (represented as triplets) that home-cover it.

A.2 Proof of Lemma 4.2

Proof. Consider a arbitrary binary gate $g = (q_i, q_j, t)$. Let t_i (t_j) be timestamp of the last unary gate on q_i (q_j) before t or 0 if there is no unary gate on q_i (q_j) before t. Then, its easy to see by Def. 4.1 that the only migrations that home-cover g are $(q_i, \pi(q_j), t_i)$ or $(q_j, \pi(q_i), t_j)$. ◀

A.3 Proof of Lemma 4.3

Proof. Consider a cycle of length n in G_{HC}, and let the i^{th}vertex in the cycle be the migration (q_i, p_i, t_i) for $0 \leq i \leq n-1$. We claim that $\pi(q_i) = \pi(q_{(i+2) \mod n})$ for $0 \leq i \leq n-1$. Consider the sequence of vertices $(q_i, p_i, t_i), (q_{(i+1) \mod n}, p_{(i+1) \mod n}, t_{(i+1) \mod n})$, and $(q_{(i+2) \mod n}, p_{(i+2) \mod n}, t_{(i+2) \mod n})$. Since $(q_{(i+1) \mod n}, p_{(i+1) \mod n}, t_{(i+1) \mod n})$ is connected to both the other vertices, we have that $p_{(i+1) \mod n} = \pi(q_i)$ and $p_{(i+1) \mod n} = \pi(q_{(i+2) \mod n})$ which implies that $\pi(q_i) = \pi(q_{(i+2) \mod n})$. The above implies that for an odd-length cycle, we'll get $p_i = \pi(q_i) = p$ for all i and some particular partition p, which is impossible for a migration/vertex (see Def. 3.3). ◀

A.4 Optimal MS-HC Algorithm Psuedo-Code

Algorithm 2

Input: A partitioned quantum circuit. Let T be the set of non-local gates.
Output: An optimal set of migrations that home-covers all the non-local gates T.

1: $M \leftarrow \emptyset$
2: **for all** $g \in T$ **do**
3: $\quad M \leftarrow M \bigcup \{\text{migrations that home-cover } g\}$.
4: **end for**
5: Construct the bipartite graph G_{HC} over T and M.
6: $M^* = \text{MinVC}(G_{HC})$.
7: **return** M^*.

A.5 Proof of Theorem 5.3

Proof. Let $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ be the sets of migrations picked by the Greedy Algorithm over k iterations. Let a_i is the number of gates covered by $\mathcal{M}_i$, that weren't previous covered by $\mathcal{M}_1$ to $\mathcal{M}_i$ sets. Thus, the overall greedy solution covers $(a_1 + a_2 \ldots + a_k)$ gates, which is equal to N, the total number of gates in the circuit (as the greedy algorithm only terminates when all the gates have been covered). Let be optimal solution be O; here, O is a set of migrations that cover all the N gates in the circuit.

Consider a stage when the Greedy Algorithm has already selected $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{i-1}$ sets of migrations. We can observe the following.

- The total number of gates already covered by the greedy sets $\mathcal{M}_1$ to M_{i-1} is $\sum_{j=1}^{i-1} a_j$. Thus, the number of gates covered by the optimal solution that have not been yet covered by the greedy sets $\mathcal{M}_1$ to $\mathcal{M}_{i-1}$ is at least $N - \sum_{j=1}^{i-1} a_j$. This follows from "monotonicity" of the coverage function; in particular, from the fact that the $\mathcal{M}_1$ to M_{i-1} sets and the optimal solution together still cover N gates.
- By Lemma 5.5, the optimal set O can be divided into disjoint same-partition subsets of migrations $O_1, O_2, \ldots, O_m$ such that $B(O) \leq \sum_{i=1}^{i=m} B(O_m)$ where $B(X)$ is the benefit of set X at this stage.
- By pigeon hole principle and above, there exists a same-partition set O_l of migrations such that $B(O_l)$ at this stage is at least

$$|O_l|(N - \sum_{j=1}^{i-1} a_j)/|O|.$$

- Since the next set $\mathcal{M}_i$ picked by the Greedy Algorithm is a same-partition set of migrations with the highest benefit-density, i.e., one that covers the most number of uncovered (by $\mathcal{M}_1$ to $\mathcal{M}_{i-1}$) gates per unit migration, the number of new gates covered by $\mathcal{M}_i$ is at least $|\mathcal{M}_i|(N - \sum_{j=1}^{i-1} a_j)/|O|$. Thus, we have

$$a_i \geq |\mathcal{M}_i|(N - \sum_{j=1}^{i-1} a_j)/|O|.$$

Now, using the above equation, it is easy to show by induction that $(N - \sum_{j=1}^{i} a_j) \leq N(1 - 1/|O|)^{i'}$, where $i' = \sum_{j=1}^{i} |\mathcal{M}_j|$, the total number of migrations in the Greedy solution till the i^{th}stage. Thus, when $i' = |O| \ln N$, we get $(N - \sum_{j=1}^{i} a_j)$ (the number of uncovered

elements) as less than 1, which is when the Greedy Algorithm stops (after one more step). Thus, the number of migrations selected by the Greedy solution is we $|O| \ln N$, showing that the Greedy Algorithm is $(\ln N)$-factor approximation algorithm, where N is the total number of gates in the circuit. ◀

A.6 Proof of Lemma 5.5

Proof. Let O be the optimal solution of the given instance of the `MS-GC` problem. Let B_O be the benefit-graph of O. We make the following two claims. **First**, we claim that each connected component in the benefit graph B_O is a same-partition set. This claim follows from the observation that if two nodes are *connected* by a path in the benefit graph B_O, then the corresponding migrations migrate the appropriate qubits to the *same* partition. Recall that $B(\{m_1, m_2\})$ is non-zero if and only if m_1 and m_2 are migrating the corresponding qubits to the same partition. **Second,** let $B_{O1}, B_{O2}, \ldots, B_{Ol}$ be the l connected components of B_O, with O_1 to O_l denoting the subsets of migrations corresponding to the connected components. We claim that $B(O) \leq \sum_{i=1}^{l} B(O_i)$. This follows[3] from the following facts: (i) Each O_i is a same-partition set of migrations (from the first claim above), and (ii) If a gate g is covered by migrations in O, then it is covered by a single or a pair of migrations in some O_i.[4] Thus, the lemma follows. ◀

A.7 Proof of Lemma 5.6

Proof. This follows from the fact that if a non-local gate g is covered by some migration(s) in M, then *only one* of the following is true: (i) there is a *unique* migration $m \in M$ that covers g, or (ii) there is a *unique* pair of migrations $\{m_1, m_2\}$ that together cover g. ◀

A.8 Evaluating Execution Memory vs. Communication Trade-off

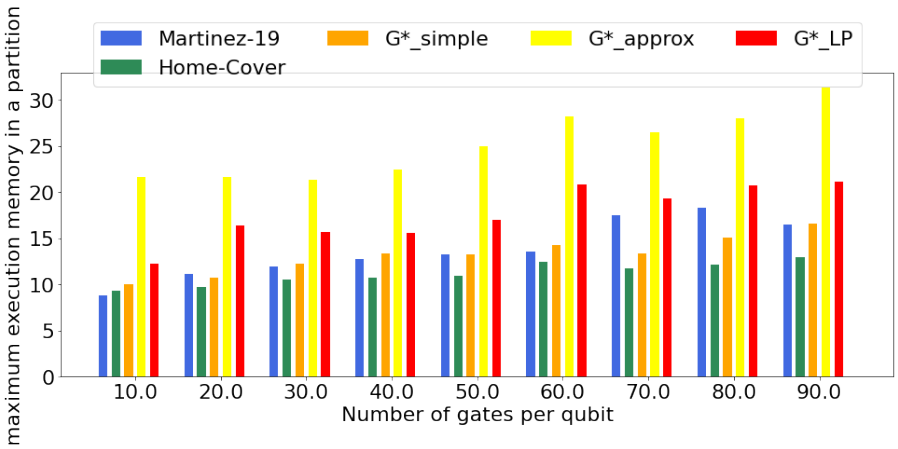


Figure 6 Maximum ebit memory required in a single partition for varying number of gates per qubit.

[3] We note that, in general, $B(O)$ may not be equal to $\sum_{i=1}^{l} B(O_i)$, since a gate may be independently covered by multiple (individual or pairs of) migrations across different O_i's.

[4] Note that a gate g is never covered by a pair of migrations together that lie in different O_i's.

Recall that our `Home-Cover` algorithm restricts execution of binary gates to the home-partition of one of the operands, to intuitively minimize the execution memory (i.e., maximum number of linked copies migrated from other partitions, at any instant) usage in any partition. To verify this intuition, we plot in Fig. 6, the maximum execution memory usage across partitions, and observe a modest difference in the usage of execution memories between `Home-Cover` and G^*`_Simple` algorithms but a higher usage in other G^*-based algorithms. Also, observe that the memory usage of `Martinez-19` is slightly higher than that of G^*`_Simple`.