Automated distribution of quantum circuits via hypergraph partitioning

Pablo Andrés-Martínez[*] and Chris Heunen[†]

University of Edinburgh

(Dated: September 10, 2019)

Quantum algorithms are usually described as monolithic circuits, becoming large at modest input size. Near-term quantum architectures can only manage a small number of qubits. We develop an automated method to distribute quantum circuits over multiple agents, minimising quantum communication between them. We reduce the problem to hypergraph partitioning and then solve it with state-of-the-art optimisers. This makes our approach useful in practice, unlike previous methods. Our implementation is evaluated on five quantum circuits of practical relevance.

PACS numbers: 03.67.Ac; 03.67.Lx

I. INTRODUCTION

Quantum computation [1–3] employs the laws of quantum mechanics to design systems capable of outperforming classical computers in certain problems [4–6]. Over the past couple of decades, this idea has rapidly developed from theoretical results into actual quantum technology [7–9].

Although there are other approaches [10], the dominant way to present a quantum algorithm is as a quantum circuit [11]: a description of how quantum devices, chosen from a fixed finite set, are applied to different parts of the input system; see Figure 1 for an example. Each of the 'wires' that quantum devices act upon typically consist of a two level quantum system called a quantum bit or qubit. The qubit count grows with the input size, and for relevant problems such as the *unique shortest vector* problem (with applications in cryptography [12]) the circuit grows large: lattice dimension 3 already requires 842 qubits and 95,624 gates [13].

Near-term quantum computing architectures are not capable of executing such large circuits. We are currently entering the era of Noisy Intermediate-Scale Quantum (NISQ) technology [14], being able to fabricate small quantum computing units (QPU for short) ranging from 10 to almost 100 qubits. Much effort is being dedicated to further increase the number of qubits that QPUs can manage, but as the number of qubits grows, the challenge of addressing each qubit individually and shielding them from unwanted interactions and decoherence rapidly becomes unmanageable [15]. To scale up beyond this point, researchers are proposing distributed quan-
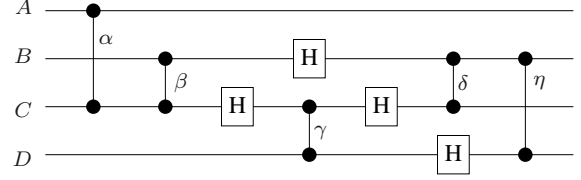

FIG. 1. Example quantum circuit, applying Hadamard gates H and CZ gates $\alpha, \beta, \gamma, \delta$ and η to four input qubits A, B, C, D.

tum architectures [16, 17] that integrate multiple smaller QPUs that cooperate to simulate a larger circuit. This requires QPUs to coordinate, making it necessary to allocate resources for communication and establishing a trade-off: the more processors we wish to use to perform the computation, the larger the communication cost will be. In the extreme case, one could imagine each individual qubit being managed by a separate QPU, so every multi-qubit gate would require inter-QPU communication.

Quantum communication is performed more profitably by photonics, whereas in-processor communication is easier with cold matter or solid state architectures. Let us mention two examples:

- Some of the currently most advanced quantum architectures are hybrids, that connect small units of matter degrees of freedom (such as quantum dots, ion traps or nitrogen vacancy centres), into a network using photonic degrees of freedom [17–19].

- Part of the aim of the Quantum Internet Alliance [20] is to establish a network between several parties, each of whose nodes have limited quantum capabilities in the order of 10-20 qubits [21]. In this view, questions of routing information along the quantum network become important [22, 23].

Although distributed quantum computing is being discussed for scalability purposes [15], and experimental distributed architectures have been proposed [16, 17], the

standard approach of quantum programmers remains designing quantum algorithms as monolithic circuits. But how do you execute, for example, the quantum circuit in Figure 1, using a pair of 2-qubit QPUs? This document develops an automated method that distributes any circuit across any number of quantum processing units, while minimising the quantum communication between them. We reduce the problem to hypergraph partitioning, which has been extensively researched in computer science literature and has fast heuristic solvers [24, 25].

We first discuss distributed quantum computing in more detail, then we reduce the problem of distributing a circuit across multiple QPUs to hypergraph partitioning. We briefly discuss some implementation details, namely pre- and post-processing routines to improve the circuit distribution. Finally, we evaluate our results on five quantum circuits from the literature that are of interest to quantum computing.

II. DISTRIBUTED QUANTUM COMPUTING

This section describes the essential characteristics of any distributed architecture and identifies communication across QPUs as the main bottleneck. We then detail the problem at hand and discuss related work from the literature. Finally, we describe the standard way nonlocal multi-qubit gates are executed across QPUs, which will be required when building distributed circuits.

A. Distributed quantum architectures

We claim that any distributed quantum architecture (DQA for short) shares the following essential features:

- *Multiple QPUs,* each of which holds a limited number of workspace qubits. It should be possible to prepare these qubits to hold the input data of a program and read output from them through measurements.

- A *classical communication network* that the QPUs may send classical messages through when measuring their qubits and receive message over when applying corrections.

- *Ebit generation hardware.* An *ebit* is a maximally entangled bipartite quantum state shared across two QPUs. In this paper, we choose the Bell state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ as the initial state for every ebit. Each ebit comprises two qubits, called *ebit halves,* that are stored in different QPUs. An ebit should be understood as a resource that a QPU may use to communicate a single qubit to another QPU. Each QPU may have its own hardware to create and share ebits, or a separate device may generate ebits centrally. Depending on the technology,

this may involve entanglement distillation and error correction of a noisy quantum channel [26, 27]. A promising way to create ebits is to excite the qubits we wish to entangle so that they each release a photon, which are then made to interfere at a beam splitter; the outcome of the interference heralds the creation of entanglement between the qubits [28].

- *Ebit memory space* on each QPU, dedicated to the storage (and possibly generation) of ebit halves. These are the qubits that will interact with the rest of the QPUs, and are thus likely to require a different physical realisation than the rest of the qubits used as workspace for the computation. QPUs should support the application of two-qubit gates between ebit and workspace qubits, so that the entanglement can be spread within the QPU.

A DQA using ion traps has been proposed [16] where each ion trap holding up to 100 qubits acts as a QPU. Ebit generation is achieved either by creating Bell states locally in one QPU and shuttling one of the qubits towards another QPU, or by the photon-heralded entanglement generation routine previously described [28]. The authors argue that, to reduce undesired crosstalk, the ions used for ebit generation will likely need to be of a different atomic species than those used for workspace qubits. Moreover, while they assume the cost of classical communication to be negligible, the authors estimate that the generation of ebits will be roughly 300 times more costly than the application of a two-qubit gate locally within a QPU.

Other DQAs have been proposed using different technologies; for instance, via semiconductor nanophotonics [17] where qubits are encoded by quantum dots that interact with their nearest-neighbours within cavities. Each of these nearest-neighbour groups of quantum dots corresponds to the workspace qubits of a QPU, while entanglement across QPUs is generated by laser pulses through strategically positioned waveguides that affect the quantum dots closer to it. For this DQA too, the authors claim that ebit generation is the bottleneck of the architecture. In general, this is to be expected for any distributed architecture: entanglement across distant parties is harder to achieve than interactions between neighbouring qubits. Thus, our objective when implementing circuits on a distributed architecture will be to minimise the amount of ebits required; this is the focus of the present paper.

There is no clear choice of the technology to be used to implement the essential DQA features listed above. For generality's sake we therefore choose not to make any further assumptions on the specification of DQAs, ignoring details that could vary across different technologies.

B. Nonlocal quantum gates

Quantum circuits are built from devices known as gates that apply operations to the qubits (see Figure 1). A universal gateset is a collection of gates that can be used to implement any circuit up to arbitrary accuracy. The Solovay-Kitaev theorem [29] ensures that any circuit can be approximated (up to arbitrary precision ε) using only gates from any chosen universal gateset. This translation is efficient (poly-logarithmic with respect to $1/\varepsilon$) both in time and circuit depth. Therefore, without loss of generality, we assume that every circuit we are tasked to distribute is built up from one-qubit gates and CZ gates exclusively. This gateset is known to be universal, as it is locally equivalent to the Clifford+T gateset. We choose CZ over the more usual CNOT gate because of its symmetry on inputs, which simplifies some details of our algorithm.

When distributing a circuit, one-qubit gates require no communication across QPUs and are therefore trivial from our point of view. In contrast, communication is required to implement any CZ gate acting on a pair of qubits that live in different QPUs, in which case we call the gate nonlocal. To distribute quantum circuits we need to be able to implement nonlocal CZ gates.

We use an approach pioneered by Gottesman and Chuang [31] to implement multi-qubit gates across distant qubits using entangled mutipartite states and measurements. In particular, we will use the scheme proposed in [30] which, using a single ebit, implements any number of contiguous nonlocal CZ gates that act on a common qubit, as shown in Figure 2. To do so, a so-called cat-entangler first shares the state of the common qubit with the remote QPU, which requires the use of an ebit; then, the CZ gates are locally performed on the remote QPU and finally the cat-disentangler destructively measures the remaining ebit half to remove any residual entanglement.

Another option would be to use standard qubit teleportation [32] to send the qubit that the CZ gates share to the remote QPU; then the CZ gates may be applied locally, and afterwards the qubit can be sent back to its original QPU through another teleportation. The scheme in Figure 2 uses a single ebit, whereas the teleportation approach just described would use two ebits to accomplish the same result, as each teleportation consumes an ebit. However, if it so happens that the teleported qubit is not required in its original QPU any more, we could skip the second teleportation and thus use a single ebit. This rather trivial remark is relevant in the discussion of Appendix B.

C. The circuit distribution problem

Our objective is to minimise the number of ebits required to distribute a given quantum circuit. We will assume ebits may be generated to entangle any pair of QPUs in the architecture, i.e. we consider no restriction on the ebit connectivity between QPUs. In practice, constraints are likely to exist but, as shown in Figure 3, whenever two QPUs may not be directly entangled through the generation of an ebit, another QPU may act as intermediary and create the desired entanglement using two physically realisable ebits. A simple yet reasonable topological arrangement of QPUs is a hypercube, where N QPUs may be connected directly with $log\ N$ other QPUs, and indirectly through at most $log\ N$ physically realisable ebits by repeatedly using the method from Figure 3. Although relevant, this $log\ N$ factor (e.g. 7 ebits for a 128 QPU computer) is not the main bottleneck of the architecture; considering direct generation of ebits may already be up to 300 times more costly than local two-qubit operations (as estimated by the authors of the ion trap DQA [16]), the immediate concern is to reduce the overall use of ebits any distributed circuit requires, independently of whether each ebit can be generated directly or not.

On a similar note, we assume that the QPUs have no internal topological constraints, i.e. each one of them is capable of applying CZ gates upon any pair of qubits it holds; this idealisation can be accounted for at a later stage. Recent research has provided automated methods for efficiently simulating any circuit on topologically constrained QPUs, either by finding the least amount of qubit swappings required [33] or by redesigning the circuit from scratch using, for instance, Steiner trees [34, 35]. In practice, any of these methods may be used to simulate each of the circuits our algorithm (see Section III) allocates to each QPU, so the QPUs may actually be topologically constrained.

Although similar at first glance, the problem we focus on (quantum circuit distribution, QCD for short) is fundamentally different from that of simulation of circuits on topologically constrained QPUs [33–35] (TC for short). Let us stress the differences between these two problems:

- In TC, two-qubit operations are either realisable on the QPU topology or not. In QCD, the operations are all realisable, but are either local (cheap) or nonlocal (requiring expensive ebit communication). The outcome of TC is a circuit that only uses realisable operations, while QCD's outcome uses both local and nonlocal operations.

- TC attempts to find the shallowest equivalent circuit, whereas QCD only focuses on reducing the number of nonlocal (communication) operations. The most efficient circuit communication-wise may not be the smallest depth-wise.

- TC is a local optimisation problem: for each unrealisable operation, the optimal way to simulate it must be found, which may depend on the way the operations in its immediate neighbourhood are implemented. In QCD, rather than optimising each nonlocal operation separately, the focus is on the
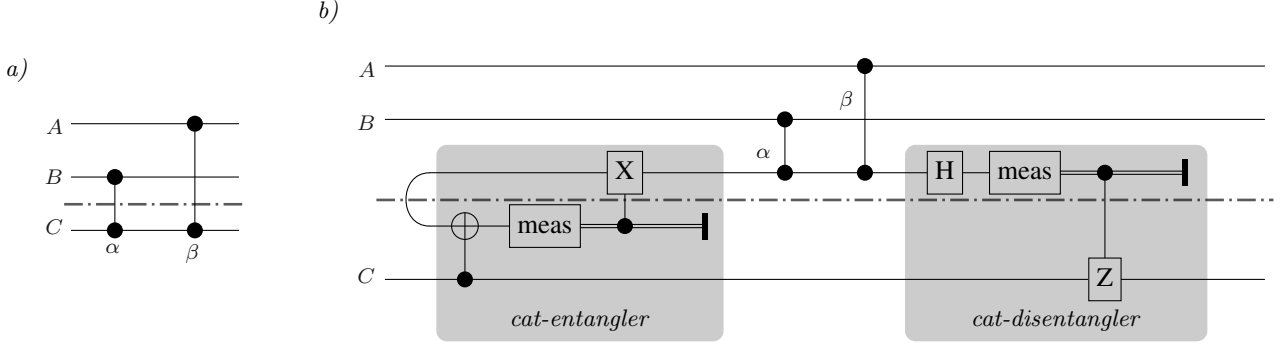
FIG. 2. Implementation, as in [30], of a group of nonlocal CZ gates that share a common qubit. Circuit *a)* is the original circuit, *b)* is the distributed version. The dashed line indicates how the circuit is separated into two QPUs. The bent wire on the left of *b)* represents an ebit. The measurement outcomes are communicated through a classical network.
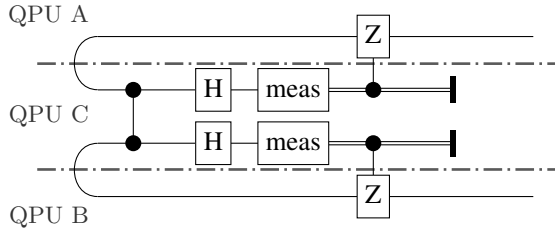

FIG. 3. Initially, a pair of ebits (bent wires) entangle QPU A with C and QPU B with C. At the end of the circuit, the two unmeasured qubits form an ebit between QPU A and B. Two ebits and two classical communications are used.

global interaction between qubits, as we need to group highly interacting qubits together, so that communication across QPUs is minimal.

The problem of distribution of quantum circuits has not received much attention in the literature. Some authors have previously proposed a solution using standard graph partitioning [36]. However, in that work, an additional exhaustive search is applied to decide how each two-qubit quantum gate should be implemented. This increases the runtime exponentially compared to the input size, making it futile in practice. The algorithm we propose in Section III encodes all possible choices of distribution in a hypergraph, and so our optimisation procedure relies solely on a hypergraph partitioner. The latter programs have been extensively studied and perfected in the computer science literature to perform efficiently even for large inputs [24, 25]. Unlike the former work [36], our approach may distribute circuits across any number of QPUs, thus answering an open problem proposed by the previous authors.

The idea of treating quantum circuits as graphs has been previously used in the literature. For instance, graphs have been employed to represent the causal structure of circuits for applications such as the recycling of circuit wires [37]. Certain results on classical simula-

tion of quantum circuits have been found through graph-theoretic approaches [38].

III. AUTOMATED CIRCUIT DISTRIBUTION

In this section we describe how the problem of quantum circuit distribution can be solved using hypergraph partitioning. We discuss some aspects of its practical implementation. Further technical details are given in Appendices A and B.

A. Reduction to hypergraph partitioning

A hypergraph consists of a set V of vertices and a set H of hyperedges, each hyperedge being defined as the subset of vertices it connects $\forall h \in H$, $h \subseteq V$. The hypergraph partitioning problem has as input a hypergraph (V, H), a parameter k giving the number of blocks (sub-hypergraphs) we wish to partition the hypergraph into, and a parameter ω known as the load-imbalance tolerance. The output is a labelling $f \colon V \to \{1, 2 \ldots k\}$ of vertices to blocks, satisfying the following two criteria:

- *load-balance*: for all $i = 1, 2, \ldots, k$:

$$|\{v \in V \mid f(v) = i\}| < (1+\omega)\frac{|V|}{k} \quad (1)$$

which implies that a valid labelling f must assign roughly the same amount of vertices to each block, with ω acting as a tolerance parameter.

- *minimal number of cuts*: given a way to assign a score $\chi_g \in \mathbb{N}$ to a labelling $g \colon V \to \{1, 2, \ldots, k\}$, the score of the output χ_f should be the lowest possible: $\forall g, \chi_f \leq \chi_g$. The score may be calculated in several ways, corresponding to variations of the hypergraph partitioning problem. We use $\chi_g = \sum_{h \in H} \lambda_g(h)$, where

$$\lambda_g(h) = |\{i \in \mathbb{N} \mid \exists v \in h, g(v) = i\}| - 1. \quad (2)$$

Equation (2) calculates, for each hyperedge $h \in H$, the number of different blocks its vertices have been assigned to, and substracts one in order to obtain the number of times the hyperedge is cut; e.g. if all the vertices of h are in the same block, $\lambda(h) = 0$.

We reduce the problem of efficiently distributing quantum circuits to the problem of hypergraph partitioning along the following intuition:

Hypergraph partitioning	Circuit distribution
vertices	wires (qubits)
hyperedges	groups of CZs
partition	distribution
blocks	QPUs
load-balance (1)	load-balance
fewest cuts (2)	fewest ebits used

The algorithm in Figure 4 encodes all information about how the circuit's CZ gates may be grouped together (i.e. when they may share the same ebit, see Figure 2) by representing such groups as a single hyperedge. The algorithm runs in time linear $\mathcal{O}(n)$ in the number n of gates of the input circuit. Figure 5 shows an example execution. Each vertex in the hypergraph corresponds to either a wire or a CZ gate; we will refer to them as wire-vertices and CZ-vertices respectively. The following theorem is the key insight that makes our approach successful.

Theorem *Given a circuit, each of its possible distributed implementations (without altering the gateset or the gate order) corresponds to a unique partition of its hypergraph (given by Figure 4) whose number of cuts equals the number of ebits required.*

The theorem implies that we may use third-party hypergraph partitioners to produce circuit distributions with low ebit count. We now explain the intuition behind the theorem. First, observe that any distribution is described by a hypergraph partition: assigning a wire-vertex to a block indicates in which QPU the corresponding wire is allocated. Similarly, assigning a CZ-vertex

```
1   input: circuit
2   output: (V,H)
3   begin
4     V ← ∅
5     H ← ∅
6     foreach wire in circuit do
7       V ← V ∪ {wire}
8       hedge ← {wire}
9       foreach gate in wire do
10        if isCZ(gate) then
11          V ← V ∪ {gate}
12          hedge ← hedge ∪ {gate}
13        else
14          H ← H ⊎ {hedge}
15          hedge ← {wire}
16      H ← H ⊎ {hedge}
17  end
```

FIG. 4. Pseudocode for the algorithm that translates a quantum circuit into a hypergraph containing all relevant qubit interaction information.

to a block determines which QPU will perform the CZ operation. Accordingly, the CZ gate will be local or require communication (i.e. ebits) to access its target wires. Notice that in Figure 5 each hyperedge connects a wire-vertex with multiple CZ-vertices: it represents all the locations where the wire's state is required. The number of cuts of a given hyperedge corresponds to the number of extra blocks it reaches (2), and for each of them an ebit is needed so the wire's state is accessible. Therefore, the number of cuts corresponds precisely to the number of ebits. Appendix A gives a detailed proof of the theorem.

To build the distributed circuit, add a cat-entangler and cat-disentangler for each cut, and then allocate all CZ gates to their corresponding QPU, connecting the relevant wires and ebit halves. This translation takes $\mathcal{O}(cuts + gates)$ steps. However, by construction of the hypergraph, we know that $cuts \leq 2\,gates$, and thus this transformation takes time $\mathcal{O}(n)$, i.e. linear in the number n of gates from the original circuit.

B. Implementation

Reducing the problem to hypergraph partitioning lets us use third-party solvers such as KaHyPar [24]. We implemented this approach in the quantum circuit description language Quipper [39]; the code is available at [40].

Apart from extracting a hypergraph out of the input circuit (Figure 4), and building the distributed circuit from the resulting partition, we include some additional pre-processing and post-processing phases:

- *Pre-processing 1*: transform the input circuit into an equivalent one using only one-qubit gates and CZ gates; Quipper provides specialised functionality to do so.

- *Pre-processing 2*: use the well-known rules from Figure 6 to reorder CZ gates and some 1-qubit gates, pulling CZ gates as early in the circuit as possible. This brings CZ gates closer together, letting our algorithm implement larger groups of non-local CZ gates using a single ebit. As shown in Figure 6, doing so may create new 1-qubit gates, namely Pauli X gates. Using the same rules, these byproduct gates can be pushed to the end of the circuit, where they will cancel out with other byproduct gates, so the overhead is bounded by at most a single pair of extra Pauli (X and Z) gates per wire.

- *Pre-processing 3*: in many circuits, the main group of qubits that another qubit interacts with varies between the different stages of the circuit. Then, if we were to use the hypergraph of the whole circuit, the different connectivities of each stage would be confounded, preventing the hypergraph partitioner from properly optimising them. To account for this, we first run a procedure that detects significant changes in the circuit's qubit connectivity
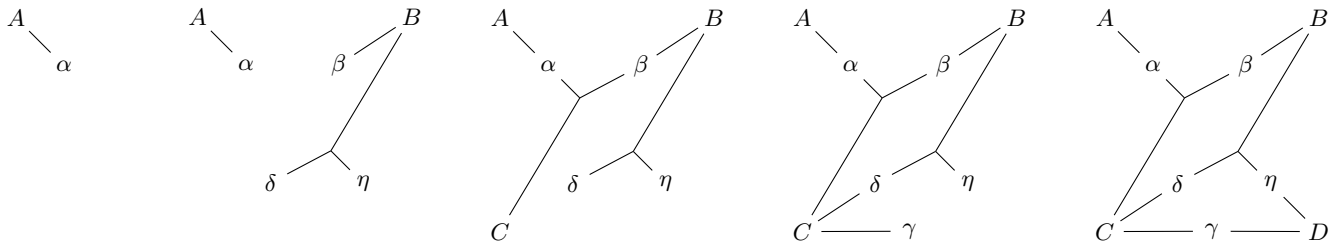
FIG. 5. Step by step execution of the algorithm in Figure 4 with input the quantum circuit of Figure 1. Each hyperedge is represented as a collection of line segments that all meet at one end, while their other ends reach each of the hyperedge's vertices. Greek letters represent CZ-vertices, latin letters represent wire-vertices.

and splits the circuit into multiple segments accordingly. Each of these segments is then distributed using the approach presented in Section III A. Appendix B details this extra pre-processing procedure.

- *Post-processing*: reduce the required ebit storage space by garbage management while building the distributed circuit: immediately after performing the last CZ gate of a group that involves an ebit, apply its cat-disentangler. This destroys the ebit so its space can be reused to store a newly created ebit.

IV. RESULTS

Our algorithm was evaluated on five quantum circuits provided by Quipper's library [39]. These circuits implement algorithms that have been discussed in the literature as examples where quantum computers achieve computational speedup. Their default configuration was used unless stated otherwise:

- *Boolean formula (BF)* [41]: the circuit fragment implementing the quantum walk, the core of the algorithm;

- *Binary welded tree (BWT)* [42]: tree height set to 200 (from 5 by default);

- *Ground state estimation (GSE)* [43]: number of precision qubits increased to 150 (from 3 by default);

- *Unique shortest vector (USV-R)* [12]: the subproblem called 'R', with lattice dimension 3 (from 5 by default);

- *Quantum Fourier transform (QFT)* [2]: using 200 qubits.

Table I shows the number of qubits and CZ gates of each circuit. These circuits require more qubits than

TABLE I. Original number of qubits and CZ gates of each of the circuits. We distributed each of them across k different QPUs, with $4 \leq k \leq 16$. Columns 'Ebit space overhead' and 'Time' show the worst value among these distributions.

Circuit	Qubits	CZ gates	Time	Ebit space overhead
BF	105	25,590	23.00s	4.8%
BWT	614	261,456	389.39s	1.1%
GSE	156	237,750	307.75s	2.6%
USV-R	842	377,695	282.32s	2.4%
QFT	201	199,000	327.48s	4.0%

the number a single near-term QPU can handle, making them meaningful examples on which to evaluate the distribution approach proposed in this paper. The times shown in the table indicate how long it took to obtain the distributed circuit using our implementation (available at [40]), running it on a standard desktop computer. The fact that circuits of this size can be distributed in a few minutes shows that our approach is useful in practice.

The last column from Table I shows the percentage of extra quantum memory required to store the ebit halves used for communication. The proportion is calculated by counting the maximum number of ebit halves stored simultaneously, and dividing it by the number of qubits in the original circuit. This overhead was considerably reduced from previous versions of our approach by limiting the number of gates allowed to be applied between two nonlocal CZ gates sharing an ebit, so that the corresponding ebit does not need to be stored over a long period of time.

Figure 7 shows the proportion of ebits required when each circuit was distributed. In all cases, over 60% of the CZ gates could be implemented either locally or 'for free' using already existing ebits. Naturally, as the number of QPUs used to distribute the circuit increases, more communication is required among them and a larger number of ebits is used. In practice, the number of QPUs each circuit should be distributed across will be determined by the circuit size; for instance, GSE may be distributed across 8 QPUs, each managing 20 qubits.
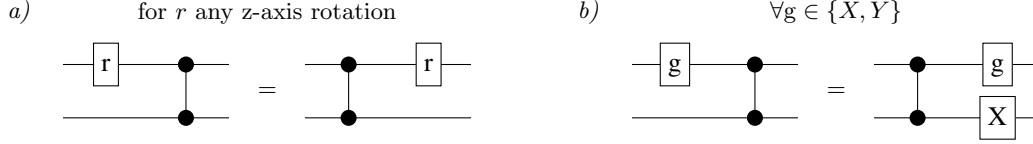
a) for r any z-axis rotation   b) $\forall g \in \{X, Y\}$

FIG. 6. Well-known cases where a 1-qubit gate can be pushed through a CZ gate. In case *b)* a byproduct gate is created. These byproduct gates can in turn be pushed through other CZs.
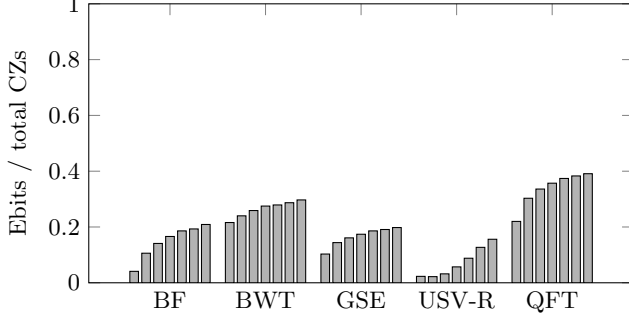

FIG. 7. Each bar shows the proportion of ebits required over the total number of CZs. For each circuit and from left to right, the bars correspond to distributing the circuit across 4, 6, 8, 10, 12, 14 and 16 QPUs with an equal number of qubits allocated to each.
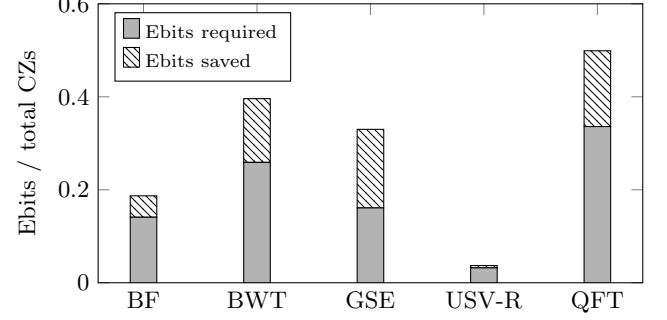

FIG. 8. The gray bar indicates the proportion of ebits required in the distribution found using hypergraph partitioning. The bar on top indicates the extra ebits if standard graphs were used instead. For all circuits, the data corresponds to distributing them across 8 QPUs.

To put the quality of these results into perspective, Figure 8 compares our approach with a simplified version using standard graph partitioning instead of hypergraph partitioning. The use of hypergraphs is the main contribution of our approach: in previous works [36] the optimisation of the number of nonlocal gates that are implemented 'for free' was achieved by exhaustively exploring the space of all possible configurations, which is exponential in size and therefore not workable in practice. Thanks to state-of-the-art hypergraph partitioners, this optimisation can be achieved in a practical amount of time by letting heuristics guide the search, instead of trying each possible configuration. The proportion of 'Ebits saved' as labelled in Figure 8 corresponds to the number of CZ gates that were implemented 'for free' using our approach. In some cases, such as GSE, we are saving approximately half of the number of ebits; in other cases, such as USV-R, the improvement is almost unnoticeable because most CZ gates are already implemented locally.

Bonus: Distributing CCZ gates

For certain DQAs, it has been discussed that local Toffoli gates could be computed at approximately the same cost as a local CZ gate [16]. Toffoli gates are three-qubit gates extensively used in quantum circuits and, in most architectures, they are implemented by decomposing each one into multiple one-qubit gates and 6 CNOT gates [44]. Interestingly, we can easily adapt our ap-

proach to distribute CCZ gates, which are locally equivalent to the Toffoli gate by applying one-qubit Hadamard gates.

To extend our approach to this setting, it suffices to realise that the approach from Figure 2 can be used to implement groups of nonlocal CCZ gates together with CZ gates; for instance, if CZ gate α from Figure 2 is replaced by a CCZ gate acting on the three qubits, the same cat-entangler and cat-disentangler allow us to implement both the CCZ gate and CZ gate β using a single ebit. After all, this cat-entangler and this cat-disentangler are compatible with the qubit basis the CCZ gate acts upon (which is the same as CZ). If a CCZ gate has each of its three wires allocated to different QPUs, two ebits will be required to implement it, so that the quantum information in each of the wires can be accessed by the QPU where the CCZ is actually applied. When building our hypergraph from the circuit (as in Figure 4), CCZ-vertices are created in the same way CZ-vertices are, but in this case each of them would be reached by three hyperedges: one per wire the gate acts upon. The rest of the approach works exactly the same as described before, consuming an ebit whenever a cut appears in the hypergraph partition, regardless of whether it reaches a CCZ-vertex.

If an architecture allows implementing local CCZ gates directly, this extended approach would yield distributed circuits requiring fewer ebits: the three qubits of a CCZ only interact once, whereas if the CCZ gate is implemented using 6 CZ gates, the communication required is
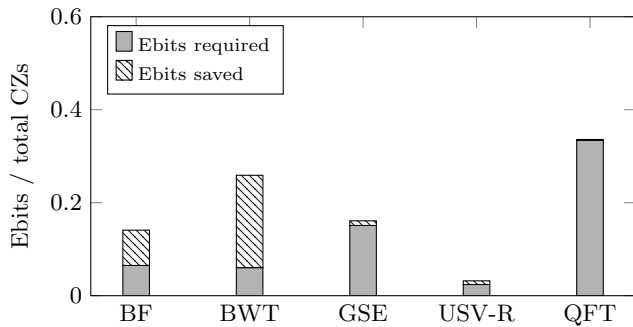
FIG. 9. The gray bar indicates the proportion of ebits required when using the extension where CCZ gates are distributed. The bar on top indicates the extra ebits if the CCZ gates are decomposed into CZ gates instead. In both cases, hypergraph partitioning is used. For all circuits, the data corresponds to distributing them across 8 QPUs.

increased. Figure 9 shows that distributing CCZ gates saves a remarkable proportion of ebits for circuits using n-qubit gates (with $n > 2$) extensively: BF and BWT. Naturally, this extension does not change the result when only two-qubit gates are applied between qubits, such as in QFT.

V. DISCUSSION AND FURTHER WORK

The lemma from Appendix A states that, in a similar way we may use hypergraph partitioning to solve the problem of quantum circuit distribution, the other way around is also feasible. This implies that if someone could devise an optimisation procedure that beats our distribution approach (i.e. gives better results and takes less time), we could immediately convert such a procedure into a hypergraph partitioner that beats KaHyPar [24], the state-of-art hypergraph partitioner we used. Considering that hypergraph partitioning has been extensively studied by experts in algorithm design [24, 25, 45], it is unlikely that a dramatically better approach to quantum circuit distribution exists unless some of the constraints we imposed are lifted. These constraints are described below; they constitute the open problems that should be addressed in order to reduce the communication cost of distribution even further:

- *Gateset.* Our chosen gateset contained every one-qubit gate and a single two-qubit gate: the CZ gate. Gatesets where other multi-qubit gates are allowed may bring better results. Our approach is easily adapted to use other gatesets, as shown in Section IV where the CCZ gate was included in our gateset. The question of which gateset is best for distribution is left as an open problem, and we point out that this may be architecture-dependent.

- *Rearranging multi-qubit gates.* The procedure labelled pre-processing 2 in Section III B rearranges one-qubit gates in the circuit to create larger groupings of CZ gates, which can reduce the number of ebits required to distribute the circuit. It is likely that, by rearranging multi-qubit gates, the connectivity of the circuit may be changed in a way that favours distribution.

It is important to stress that hypergraph partitioners are not expected to provide the optimal partition of the input hypergraph: that problem is intractable on a classical computer (namely, it is NP-hard [45]). Instead, we work with close to optimal solutions that can be found efficiently by classical computers. The results discussed in Section IV were obtained using such suboptimal partitions. These allow us to reduce the communication cost of distributing circuits, and thus help us compute problems that are not tractable in classical computers (not even suboptimally) and whose quantum circuits would require more qubits than the number a near-term QPU can handle.

Apart from the restricted number of qubits a QPU can manage, there is another fundamental limit to scalability we have overlooked up to this point: the short lifespan of qubits due to decoherence. NISQ computers will only be able to store and manipulate quantum information for a short period of time, which means we should not expect to be able to execute more than 1000 consecutive two-qubit gates [14]. This means that optimising the depth of the circuit (i.e. reducing the largest chain of consecutive gates) is considered essential. There are many different methods that reduce the depth of circuits [46, 47], and we propose these should be used to optimise the input circuit before distributing it with our approach. An interesting line of research is exploiting how parallelism may be employed to further reduce the circuit depth: if a circuit is distributed across different QPUs, the QPUs may perform simultaneous computations, reducing the total time the quantum information needs to be coherently stored.

We have seen that distributed quantum architectures [16, 17] have been proposed as a feasible approach to increase the size of quantum computers. Circuits that are too large to be performed in near-term quantum processing units may be run on distributed quantum architectures at the cost of quantum communication. We have presented an automated method for distributing quantum circuits across multiple agents, minimising the quantum communication between them. In this last section we have discussed the limitations of our approach and pointed out further lines of research that would improve it. Our approach was evaluated favourably on five test circuits of interest to the quantum computing literature. These circuits are too large to fit in a single near-term QPU and thus need to be distributed in order to be implemented.

[1] R. P. Feynman, Simulating physics with computers, Int. J. Th. Phys. **21**, 467 (1982).

[2] M. J. Nielsen and I. L. Chuang, *Quantum computation and quantum information* (Cambridge University Press, 2000).

[3] S. Lloyd, Universal quantum simulators, Science **273**, 1073 (1996).

[4] P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Review **41**, 303 (1999).

[5] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Annual ACM symposium on theory of computing* (ACM, 1996) pp. 212–219.

[6] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum algorithm for linear systems of equations, Phys. Rev. Lett. **103**, 150502 (2009).

[7] J. L. O'Brien, A. Furusawa, and J. Vučković, Photonic quantum technologies, Nature Photonics **3**, 687 (2009).

[8] M. H. Devoret and R. J. Schoelkopf, Superconducting circuits for quantum information: an outlook, Science **339**, 1169 (2013).

[9] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, Quantum entanglement, Rev. Mod. Phys. **81**, 865 (2009).

[10] R. Raussendorf and H. J. Briegel, A one-way quantum computer, Phys. Rev. Lett. **86**, 5188 (2001).

[11] D. Deutsch, Quantum computational networks, Proc. Roy. Soc. A **425**, 73 (1989).

[12] O. Regev, Quantum computation and lattice problems, SIAM J. Comput. **33**, 738 (2004).

[13] Extracted from Quipper USV-R implementation [39].

[14] J. Preskill, Quantum Computing in the NISQ era and beyond, Quantum **2**, 79 (2018).

[15] R. van Meter and S. J. Devitt, Local and distributed quantum computation, IEEE Computer **49**, 31 (2016).

[16] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects, Phys. Rev. A **89**, 022317 (2014).

[17] R. van Meter, T. D. Ladd, A. G. Fowler, and Y. Yamamoto, Distributed quantum computation architecture using semiconductor nanophotonics, Int. J. Q. Inf. **8**, 295 (2010).

[18] C. J. Ballance *et al.*, Hybrid quantum logic and a test of bells inequality using two different atomic isotopes, Nature **528**, 384 (2015).

[19] M. S. Blok, N. Kalb, A. Reiserer, T. H. Taminiau, and R. Hanson, Towards quantum networks of single spins: analysis of a quantum memory with an optical interface in diamond, Faraday Discuss. **184**, 173 (2015).

[20] quantum-internet.team.

[21] S. Wehner, D. Elkouss, and R. Hanson, Quantum internet: a vision for the road ahead, Science **362**, eaam9288 (2018).

[22] E. Schoute, L. Mancinska, T. Islam, I. Kerenidis, and S. Wehner, Shortcuts to quantum network routing, arXiv:1610.05238 (2016).

[23] F. Hahn, A. Pappa, and J. Eisert, Quantum network routing and local complementation, arXiv:1805.04559 (2018).

[24] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, Engineering a direct k-way Hypergraph Partitioning Algorithm, Proc. Alg. Eng. Exp. (ALENEX) **19**, 28 (2017).

[25] Ü. Çatalyürek and C. Aykanat, Patoh (partitioning tool for hypergraphs), in *Encyclopedia of Parallel Computing*, edited by D. Padua (Springer US, Boston, MA, 2011) pp. 1479–1487.

[26] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, Mixed state entanglement and quantum error correction, Phys. Rev. A **54**, 3824 (1996).

[27] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, Distributed quantum computation over noisy channels, Physical Review A **59**, 4249 (1999).

[28] C. Simon and W. T. M. Irvine, Robust long-distance entanglement and a loophole-free bell test with ions and photons, Physical review letters **91**, 110405 (2003).

[29] C. M. Dawson and M. A. Nielsen, The Solovay-Kitaev algorithm, arXiv:quant-ph/0505030 (2015).

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

[31] D. Gottesman and I. L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, Nature **402**, 390 (1999).

[32] 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, Physical review letters **70**, 1895 (1993).

[33] A. M. Childs, E. Schoute, and C. M. Unsal, Circuit transformations for quantum architectures, arXiv preprint arXiv:1902.09102 (2019).

[34] A. Kissinger and A. M.-v. de Griend, Cnot circuit extraction for topologically-constrained quantum memories, arXiv preprint arXiv:1904.00633 (2019).

[35] B. Nash, V. Gheorghiu, and M. Mosca, Quantum circuit optimizations for nisq architectures, arXiv preprint arXiv:1904.01972 (2019).

[36] M. Zomorodi-Moghadam, M. Houshmand, and M. Houshmand, Optimizing teleportation cost in distributed quantum circuits, International Journal of Theoretical Physics **57**, 848 (2018).

[37] A. Paler, R. Wille, and S. J. Devitt, Wire recycling for quantum circuit optimization, Physical Review A **94**, 042337 (2016).

[38] I. L. Markov and Y. Shi, Simulating quantum computation by contracting tensor networks, SIAM Journal on Computing **38**, 963 (2008).

[39] www.mathstat.dal.ca/∼selinger/quipper/doc.

[40] github.com/PabloAndresMartinez/Distributed.

[41] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang, Any and-or formula of size n can be evaluated in time $n^{1/2+o(1)}$ on a quantum computer, Found. Comp. Sci. (FoCS) **48**, 363 (2007).

[42] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, Exponential algorithmic speedup by quantum walk, Proc. Symp. Th. Comp. (STOC) **35**, 59 (2003).

[43] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik, Simulation of electronic structure hamiltonians using quantum computers, Molecular Physics **109**, 735 (2011).

[44] V. V. Shende and I. L. Markov, On the cnot-cost of toffoli

gates, Quantum Info. Comput. **9**, 461 (2009).

[45] L. Lyaudet, NP-hard and linear variants of hypergraph partitioning, Theoretical Computer Science **411**, 10 (2010).

[46] A. Cowtan, S. Dilkes, R. Duncan, W. Simmons, and S. Sivarajah, Phase gadget synthesis for shallow circuits, Proc. 16th International Conference on Quantum Physics and Logic (QPL, Orange, California, June 2019) (2019).

[47] M. Amy, D. Maslov, and M. Mosca, Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **33**, 1476 (2014).

APPENDIX A

In this appendix we prove the theorem presented in section III A and related results also discussed in that section.

Theorem *Given a circuit, each of its possible distributed implementations (without altering the gateset or the gate order) corresponds to a unique partition of its hypergraph (given by Figure 4) whose number of cuts is equivalent to the number of ebits required.*

Proof. First, we provide a bijection between the trivial configurations:

- a partition of the hypergraph where all vertices are in the same block corresponds one-to-one to

- the whole circuit being executed in a single QPU.

Then, we extend the bijection to any partition/distribution by defining two primitive transformations for each problems, which allow us to move vertices around. The wire-primitive moves wire-vertices:

- given a partition of the hypergraph, moving wire-vertex x to block i corresponds one-to-one to

- picking wire x and allocating it to QPU i.

The *CZ-primitive* moves CZ-vertices:

- given a partition of the hypergraph, moving CZ-vertex α to block i corresponds one-to-one to

- picking CZ gate α and allocating it to be carried out in QPU i.

Any partition/distribution can be described as a sequence of primitives: starting from the trivial configuration, first move all CZs to their corresponding block/QPU using the CZ-primitive once per CZ gate, then do the same for the wires using the wire-primitive. The one-to-one correspondence between primitives then gives us a bijection between the set of all possible distributions of the circuit and all possible partitions of its hypergraph. It remains to prove this bijection satisfies that the ebit count λ_e of a distribution and the cut count λ_c of its corresponding hypergraph partition are always equivalent $\lambda_c = \lambda_e$.

1. The trivial configuration of both problems has $\lambda_c = 0 = \lambda_e$. We impose that the block/QPU where all vertices are allocated on the trivial configuration is an auxiliary one that will not hold any vertices/wires on the final partition/distribution. Thus, it is just an artifact to simplify the proof.

2. By construction, each CZ-vertex is connected to exactly two hyperedges. When a CZ-primitive is applied, the number of cuts λ_c will increase by one if and only if, in the block where it is reallocated, there is no other CZ-vertex with whom it shares a hyperedge. The same happens for the ebit count λ_e: if, in the QPU where it is reallocated, there is no CZ with whom it shares a wire, then an ebit is required to remotely access it, otherwise the channel already exists and no additional ebit is required. Thus, we may reallocate all CZ gates while preserving $\lambda_c = \lambda_e$.

3. Applying wire-primitives to the current configuration will always decrease λ_c and λ_e. When wire-vertex x is reallocated to block i, the number of cuts λ_c will decrease by one per hyperedge x shares with a CZ-vertex in block i. The ebit count λ_e will decrease under the same circumstances, because the CZs corresponding to those CZ-vertices will be able to access the wire locally, and therefore will not require ebits to do so. Thus, we may reallocate all wires while preserving $\lambda_c = \lambda_e$.

The strategy of allocating all CZ gates first and then allocating wires is chosen for simplicity. Although tedious, it is straightforward to check case by case that the argument above holds independently of the order at which we allocate CZ gates and wires.

$\square$

The following lemma shows that, in a similar way as to how the problem of quantum circuit distribution can be reduced to hypergraph partitioning, the dual notion is also true: hypergraph partitioning can be solved using a quantum circuit distributer. This insight has no direct application in practice, but it is valuable from a theoretical point of view, as stated in the corollary that follows.

Lemma *The problem of hypergraph partitioning can be reduced to the problem of quantum circuit distribution.*

Proof. We need to show how an optimal partition of any hypergraph can be obtained by finding an optimal distribution of a dummy circuit. Given any hypergraph, create a circuit that has one wire per vertex in the hypergraph and, for each hyperedge h:

1. take the subset of vertices it reaches and the corresponding subset of wires $\mathcal{W}_h$;

2. pick (at random) one of these wires and apply CZ gates between it and each of the other wires in $\mathcal{W}_h$;

3. apply a Hadamard gate on each of the wires in $\mathcal{W}_h$.

Notice that this process takes a polynomial number of steps with respect to the number of vertices and hyperedges in the hypergraph. An example of a hypergraph and its resulting dummy circuit is given in Figure 10.

We then use the algorithm from Figure 4 to obtain a new hypergraph (see Figure 10c) which is similar to the original one, but not the same. We call this hypergraph the *extended* hypergraph. Notice that the only difference is the addition of CZ-vertices and standard edges. If we merge each CZ-vertex with the wire-vertex the standard edge connects it to, the resulting hypergraph is the same as the one given as input (see Figure 10c). It is trivial to check that this will always be the case due to the way we build the dummy circuit.

If an optimal distribution of the dummy circuit is found, we can use the bijection provided in the proof of the theorem above to obtain an optimal partition of the extended hypergraph. We can then remove the CZ-vertices by merging them, obtaining a partition of the input hypergraph. When the vertices to be merged are in the same block it is clear that merging does not affect the optimality of the partition. When they live in different blocks, reallocating the CZ-vertices so they are on the same block can never increase the cut count: either that cut is simply moved from the standard edge to the hyperedge, or the cut is no longer needed because another vertex in the hyperedge is already on that block. But because the partition of the extended hypergraph is optimal, this particular reallocation can not decrease the cut count either. Thus we may ignore the CZ-vertices altogether. The allocation of the rest of the vertices provides an optimal partition of the input hypergraph. □

Corollary *The problem of quantum circuit distribution is NP-hard.*

Proof. The previous lemma shows that hypergraph partitioning can be reduced to this problem, with all required transformations having polynomial time complexity. As hypergraph partitioning is NP-hard [45], it immediately follows by the definition of NP-hardness that the problem of quantum circuit distribution, as defined in this document, is NP-hard too. □

APPENDIX B

This appendix presents the procedure labelled *preprocessing 3* in Section III B, that informs how the input circuit should be split into segments before distributing. The goal is that, whenever the qubit connectivity within the circuit changes dramatically, the circuit is divided into two segments: one ending at some point previous to that change and the other starting from that point onwards. The different segments are then distributed using the approach described in Section III A.

The procedure requires two user-defined parameters $\omega \in \mathbb{N}$ and $\Delta \in [0, 1]$. First, the circuit is explored from left to right, splitting it into preliminary segments containing ω many CZ gates each. Then, for each segment:

1. obtain the hypergraph partitions of the current segment and the next one;

2. obtain their discrepancy score δ computed as in (3);

3. if the discrepancy δ is below the threshold Δ, view both segments as a single one (i.e. merge them) and return to step 1; otherwise obtain the distributed circuit of the current segment and continue the procedure until all segments have been distributed.

Once this process finishes, the distributed circuits are executed in the target DQA one after the other. This may require teleporting qubits [32] between QPUs when progressing from one segment to the next. Each qubit teleportation makes use of a single ebit; this cost has been taken into account in the figures and discussion of Section IV. The procedure may be modified so that parameter Δ is not required: apply step 3 only when δ is minimal and repeat the procedure until merging segments no longer decreases the ebit count. This modified version is the one used to obtain the results graphed in Section IV.

The the discrepancy score $\delta \in [0, 1]$ between two segments s and r is calculated as:

$$\delta = \sum_{w \in \mathcal{W}} \tau(w) \, \frac{\min\{h_s(w), h_r(w)\}}{\min\{H_s, H_r\}} \qquad (3)$$

where $\mathcal{W}$ is the set of all wires in the circuit, $\tau(w)$ returns 0 if the wire is allocated to the same QPU in both segments and 1 otherwise, H_s returns the total number of hyperedges in the hypergraph of segment s (similarly for segment r) and $h_s(w)$ returns the number of hyperedges that reach the vertex corresponding to wire w within the hypergraph of segment s (similarly for segment r).

Different discrepancy scores could be used without changing any other aspect of the algorithm. Equation (3) was the score that performed best among the different options we attempted. It moreover has an intuitive interpretation:

- if a wire is allocated to the same QPU in both segments, that wire's contribution to discrepancy is null, which is why we multiply by $\tau(w)$;

- $h_s(w)$ estimates the wire's relevance in the segment connectivity and hence should be proportional to the discrepancy score;

- if a wire is allocated to different QPUs in each segment, but is almost never used in one of them (i.e. $h_r(w) \approx 0$), it would be relatively cheap to reallocate that wire to match the other segment, justifying the use of $(\min\{h_s(w), h_r(w)\})$;

- to compare scores fairly they need to be normalized, which is why we divide by $(\min\{H_s, H_r\})$.
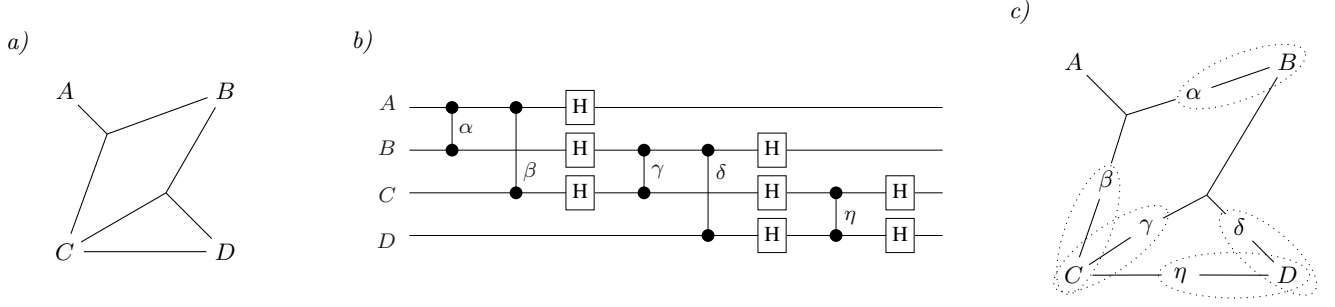
FIG. 10. *a)* An arbitrary input hypergraph. *b)* A (not unique) dummy circuit built from the hypergraph. *c)* The hypergraph obtained by applying the algorithm from Figure 4 on the dummy circuit, the hypergraph obtained is an extended version of the input one. We can retrieve the input hypergraph by merging vertices as indicated by the dotted ellipses.

This procedure uses the hypergraph partitioner multiple times. At first glance, that may seem to come at a great cost, as hypergraph partitioning is the most resource intensive routine in our approach. However, by splitting the circuit into segments, the hypergraph that is partitioned each time is much smaller than the hypergraph of the overall circuit. Considering that hypergraph partitioning is an NP-hard problem [45], the reduction of the input size improves performance dramatically. In practice, when using KaHyPar [24] (our choice of third-party hypergraph partitioner), we found out this performance improvement overcame the cost of running the partitioner multiple times.