Automated distribution of quantum circuits

Pablo Andrés-Martínez* and Chris Heunen†
University of Edinburgh
(Dated: November 27, 2018)

Quantum algorithms are usually monolithic circuits, becoming large at modest input size. Near-term quantum architectures can only execute small circuits. We develop automated methods to distribute quantum circuits over multiple agents, minimising quantum communication between them. We reduce the problem to hypergraph partitioning, which is NP-hard but has advanced heuristic solvers. Our implementation is evaluated on five quantum circuits. Four are amenable to distribution. The distribution cost is more than halved when compared to a naive approach.

PACS numbers: 03.67.Ac; 03.67.Lx

Quantum computation [1–3] leverages the laws of quantum mechanics to design physical systems capable of outperforming devices based on the laws of classical mechanics [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 *unique shortest vector* (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. It is therefore desirable to split a circuit into smaller pieces – each of which is executed on a different quantum processing unit (QPU) – in a way that computes the same solution. This approach is known as distributed quantum computing [14]. It requires QPUs to coordinate, making it necessary to allocate resources for communication. This establishes 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 requires a communication call.

Quantum communication over long distances is performed more profitably by photonics, whereas inprocessor communication is easier with cold matter or solid state architectures. Let us mention two examples:

- Two of the currently most advanced quantum architectures are hybrids, that connect small units of matter degrees of freedom (such as ion traps or nitrogen vacancy centres), into a network using photonic degrees of freedom [15, 16].

- Part of the aim of the Quantum Internet Alliance [17] is to establish a network between several parties, each of whose nodes has limited quantum capabilities in the order of 10-20 qubits [18]. In this view, questions of routing information along the quantum network become important [19, 20].

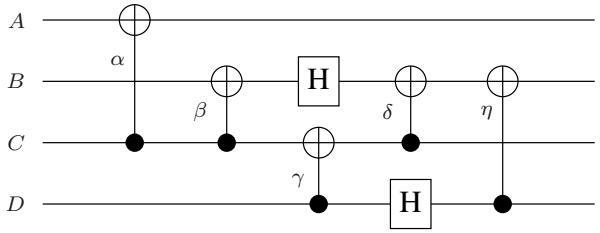

FIG. 1. Example quantum circuit, applying Hadamard gates H and CNOT gates $\alpha, \beta, \gamma, \delta$ to four input qubits A, B, C, D.

Thus distributed quantum computing may be regarded as a viable contribution towards scalability. However, the standard approach of quantum programmers is to design quantum programs as monolithic circuits [21]. But how do you execute, for example, the quantum circuit in Figure 1, using a pair of 2-qubit quantum processors? This article develops an automated method that distributes any circuit across any number of quantum processing units, while minimising the quantum communication between them.

We first discuss nonlocal quantum gates and distributed quantum computing in more detail. Then we reduce the problem to hypergraph partitioning. The latter problem is NP-hard [22], but has highly advanced heuristic solvers [23] (which compute good, though perhaps not optimal, solutions in efficient time). This leads to an algorithm that distributes any user-defined quantum circuit. We briefly discuss its implementation details, namely pre- and post-processing routines to improve the circuit distribution. Finally, we evaluate the implementation on five standard quantum algorithms.

Distributed quantum computing architectures (DQCAs) are characterised by the following features [14]:

- *Multiple quantum processing units (QPUs)*, each of which holds a number of workspace qubits, and can perform classically controlled universal quantum computation on them.
- 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. Each ebit thus comprises two qubits (each one stored in a different QPU), called *ebit halves*. An ebit contains the information needed within one QPU 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 [24].

For example, experimental DQCAs have used cavities to trap particles encoding the qubits, and laser pulses to entangle cavities across different QPUs [25].

Any DQCA strikes a compromise between ebit quality and the resources dedicated to preparing them [26]. Fortunately, efficient distributed quantum computation using noisy ebits is feasible [27]. Ebit generation remains the main bottleneck of DQCAs: it is far more expensive than classical communication and any local operation, as creating or using an ebit takes multiple local operations.

Our objective is therefore to *minimise the number of ebits* required to distribute a given quantum circuit. Besides, each QPU in a DQCA will have a limited amount of workspace that, without loss of generality, we may assume to be the same across all of the QPUs. Ideally, the number k of QPUs to distribute the circuit across is $k \approx N/C$, where N counts the qubits in the circuit, and C measures the workspace of a QPU. To reduce idle workspace, we will therefore impose an additional *load-balance* requirement: the qubits of the quantum circuit must be allocated evenly across the QPUs.

Nonlocal quantum gates Quantum circuits are constructed by connecting *gates* that apply operations to the qubits. A universal gateset is a collection of gates that can be used to implement any circuit up to arbitrary accuracy. One of the most commonly used universal gatesets is Clifford+T [28]. This finite gateset contains several gates acting on a single qubit, and a single 2-qubit gate called CNOT. When distributing a circuit, every 1-qubit gate can be implemented locally in the QPU holding the qubit; hence without inter-QPU communication. In contrast, communication is required to implement any CNOT gate acting on a pair of qubits that live in different QPUs, in which case we call the gate *nonlocal*.

Only considering the Clifford+T gateset is not restrictive. Given any circuit, the Solovay-Kitaev theorem [29] approximates it (up to arbitrary precision ε) using gates exclusively from a chosen universal gateset. This translation is efficient ($\mathcal{O}(\log^c(1/\varepsilon))$ for small c) both in time and circuit length. Thus, we may assume that all quantum circuits are implemented in the Clifford+T gateset. To distribute quantum circuits, it therefore suffices to implement the CNOT gate nonlocally.

To do so, we build on the known scheme [30] shown in Figure 2. It implements any number of contiguous nonlocal CNOT gates that act on the same control qubit and whose target qubits reside in the same QPU: first apply a so-called cat-entangler (defined in Figure 2) to share the state of the control qubit with the target QPU; then, perform the CNOT gates locally at the target QPU; finally, the cat-disentangler destructively measures the remaining ebit half. The communication between the two QPUs uses a single ebit, apart from two classical bits (the measurement outcomes).

We call this scheme the *remote-control* method, as it shares the state of the control wire with a remote QPU. In the dual *remote-target* method, the CNOT gates act on a common target wire instead, and the state of the target wire is shared with a remote QPU. The scheme for the remote-target method is the same as that for remote-control, but both cat-entangler and cat-disentangler are preceded and followed by Hadamard gates on each wire.

Depending on the layout of the circuit, some groups of nonlocal CNOT gates will be realised more efficiently (i.e. using fewer ebits) by the remote-control method, whereas others will benefit from the remote-target method. Therefore part of the challenge of distributing a circuit is deciding which method to use when implementing each nonlocal CNOT.

Hypergraph partitioning A hypergraph consists of a set V of vertices, and a family $H \subseteq 2^V$ of subsets of vertices called hyperedges. 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 should be a labelling $f\colon V \to \{1, 2 \dots k\}$ of vertices by blocks, satisfying the following two criteria:

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

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

- *minimal number of cuts*: given a way to assign a score $\chi_g \in \mathbb{N}$ to a labelling $g\colon V \to \{1, 2, \dots, k\}$, ensure that $\chi_f \leq \chi_g$ for every labelling g. The

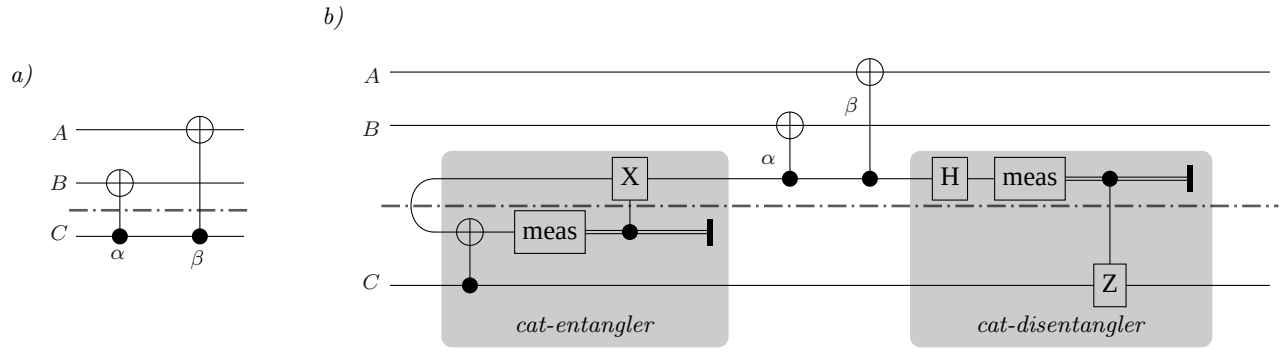

FIG. 2. Implementation, as in [30], of a group of nonlocal CNOT gates that have a common control wire. Circuit *a)* is the original circuit, *b)* is the distributed version. The dashed line indicates how the circuit is separated into two QPUs. The bent wire on the left of *b)* represents the generation of an ebit.

score χ_g may be calculated in several ways, corresponding to variations of the hypergraph partitioning problem. This paper uses $\chi_g = \sum_{h \in H} \lambda_g(h)$, where

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

This scoring not only takes into account the number of hyperedges cut, but also how many different blocks they reach.

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

Hypergraph partitioning	*Efficient distribution*
vertices	wires
hyperedges	groups of CNOTs
partition	distribution
blocks	QPUs
load-balance (1)	load-balance
fewest cuts (2)	fewest ebits used

The pseudocode in Figure 3 creates a hypergraph that summarises all the communication constraints between the qubits of the input circuit. This algorithm runs in time $\mathcal{O}(n)$ linear in the number n of gates of the input circuit. Figure 4 shows an example execution. Each vertex in the hypergraph corresponds to either a wire or a CNOT; below we refer to them as wire-vertices and CNOT-vertices respectively.

Theorem *Distributions of a quantum circuit with c ebits correspond bijectively to partitions of its hypergraph generated by Figure 3 with c cuts. A partition can be converted into a distributed circuit in time $\mathcal{O}(n)$, where n is the number of gates in the original circuit.*

We now explain the intuition behind this theorem; for the formal proof see Appendix A. 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 CNOT-vertex to a block determines which QPU will perform the CNOT operation. Accordingly, the CNOT will be local or require communication (i.e. ebits) to access its control and target wires.

Notice that each hyperedge connects a wire-vertex with multiple CNOT-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.

```
1  input: circuit
2  output: (V,H)
3  begin
4    V ← ∅
5    H ← ∅
6    hedge ← ∅
7    foreach wire in circuit do
8      V ← V ∪ {wire}
9      hedge ← {wire}
10     hType ← unknown
11     foreach gate in wire do
12       if gate == CNOT then
13         V ← V ∪ {labelOf(gate)}
14         if controlOf(gate) == wire then
15           if hType == target then
16             H ← H + {hedge}
17             hedge ← {wire}
18           hType ← control
19         if targetOf(gate) == wire then
20           if hType == control then
21             H ← H + {hedge}
22             hedge ← {wire}
23           hType ← target
24         hedge ← hedge ∪ {labelOf(gate)}
25       else
26         H ← H + {hedge}
27         hedge ← {wire}
28         hType ← unknown
29     H ← H + {hedge}
30 end
```

FIG. 3. Pseudocode translating a quantum circuit into a hypergraph.

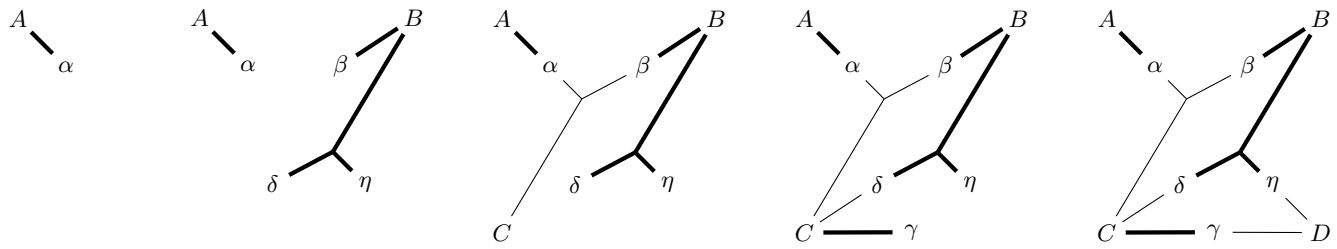

FIG. 4. Step by step execution of the algorithm in Figure 3 with input the quantum circuit of Figure 1. Each hyperedge is represented as a collection of line segments that all meet at one end, while their other ends reach each of the hyperedge's vertices. Thick lines represent hyperedges of *target* type, thin lines are used for *control* type (see Figure 3).

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

Implementation Hypergraph partitioning has been extensively studied in the computer science literature. Reducing our problem to it lets us use solvers such as KaHyPar [23]. We implemented our approach in the quantum circuit description language Quipper [31] (see [32]).

Apart from extracting a hypergraph out of the input circuit (Figure 3), and building the distributed circuit from the resulting partition, there are additional pre-processing and post-processing phases:

- *Pre-processing 1*: transform the input circuit into an equivalent one using only Clifford+T gates; Quipper provides specialised functionality to do so.
- *Pre-processing 2*: use the well-known rules [33] from Figure 5 to swap the order between CNOTs and the 1-qubit gates from Clifford+T, pulling all CNOTs as early in the circuit as possible. This brings CNOTs closer together, letting our algorithm implement larger groups of nonlocal CNOTs using a single ebit.
- *Post-processing*: reduce the ebit storage space by garbage management while building the distributed circuit: immediately after performing the last CNOT of a group that involves an ebit, apply its cat-disentangler. This destroys the ebit so its space can be reused to store another ebit.

Evaluation Our algorithm was evaluated on five standard example quantum circuits in Quipper [31]:

- *Boolean formula (BF)* [34]: the circuit implementing the quantum walk, the core of the algorithm;
- *Binary welded tree (BWT)* [35]: twice the default tree height;
- *Ground state estimation (GSE)* [36]: twice the default number of basis functions and occupied orbitals;
- *Unique shortest vector (USV-R)* [12]: the subproblem called 'R', with lattice dimension 3;
- *Quantum Fourier transform (QFT)* [2]: 200 qubits.

Figure 6 shows the proportion of CNOTs that become nonlocal after distribution, and the proportion of ebits needed. A naive distribution would use one ebit per nonlocal CNOT, so larger differences between these proportions mean larger reductions in ebit count. Our approach reduces the amount of communication needed by more than a half.

Distributing a circuit across more QPUs renders more CNOTs nonlocal and increases the ebit count. This increase is not uniform. For example, distributing GSE for $k \geq 13$ makes all CNOTs nonlocal, so the ebit count barely changes. This is undesirable, because nonlocal CNOTs cost more than local ones. For each problem, one wishes to choose the largest k with feasible communication cost; for GSE, $k = 11$ is reasonable.

Figure 7 shows the ratio between QPU space dedicated to communication (ebit halves) and space dedicated to computation (workspace qubits). Again this trade-off is circuit-dependent. In particular, QFT does not distribute well: $k = 3$ already requires as much ebit space as workspace. Figures 6 and 7 suggest the other four circuits do benefit from distribution, particularly BF and USV-R.

Discussion Our Theorem ensures that, if a hypergraph partition with minimal number of cuts is found, the distribution we provide has the least number of ebits.

- Our method is optimal under the choice of Clifford+T gateset and implementing nonlocal CNOTs as in [30]. However, some quantum circuits may be distributed more efficiently under a

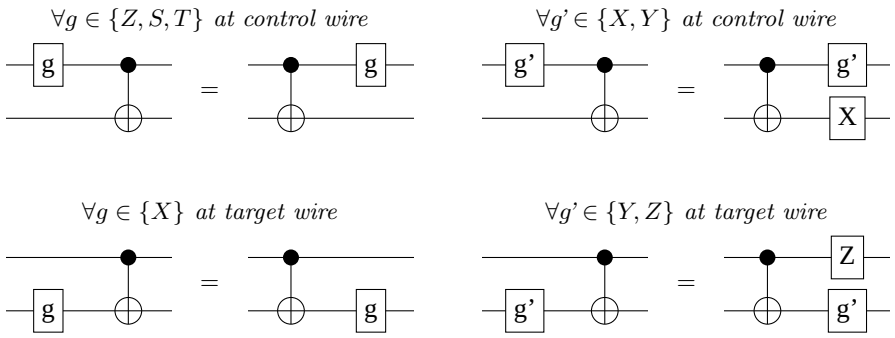

FIG. 5. Different cases when a 1-qubit gate from Clifford+T can be pushed through a CNOT gate.

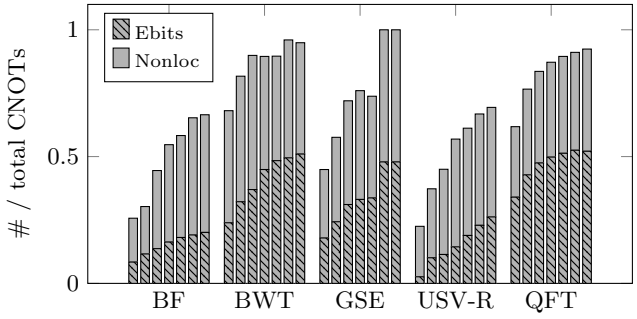

FIG. 6. Each bar shows the number of nonlocal CNOTs and ebits required, normalised by the original number of CNOTs. For each circuit, the bars from left to right correspond to distributing across 3, 5, 7, 9, 11, 13 and 15 QPUs.

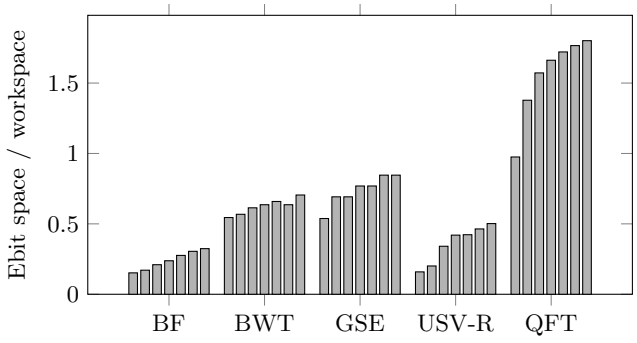

FIG. 7. Each bar shows the ratio between the space dedicated to communication and the workspace dedicated to computation. For each circuit, the bars from left to right correspond to distributing across 3, 5, 7, 9, 11, 13 and 15 QPUs.

different gateset. The algorithm is easily adapted to use other gatesets, as it deals with abstract operations requiring inter-QPU communication. We leave open the question of which gateset is best.

- Phase *pre-processing 2* rearranges gates in the circuit, but does not reorder CNOTs acting on the same wire, which could reduce the ebit count further. One might use a complete set of axioms for rearranging CNOTs [37] to explore equivalent circuits with the least ebit count. This is a non-trivial optimisation problem, because rearranging CNOTs creates new ones as byproducts (see axiom CNT.8 [37]), which affect the ebit count.

- Hypergraph partitioning is an NP-hard problem [22]. Therefore no solver can guarantee optimal solutions in feasible time, and one cannot expect an optimal distribution in practice. Distributing quantum circuits is also NP-hard, which follows from the Theorem. On the bright side, NP-hardness is often addressed in practice [38], using heuristics that exploit properties of the specific problem. For instance, knowing that all CNOT-vertices have degree two may help developing search strategies that work better than KaHyPar's generic ones.

Our implementation may be improved technically, by taking better advantage of Haskell's scalable datatypes, or by merging into Quipper's core itself.

Conclusion We have presented an automated method to distribute quantum circuits across multiple agents, minimising the quantum communication between them. Its implementation was evaluated favourably on four test circuits; the fifth does not seem amenable to distribution. Our method could thus practically inform larger experimental realisations of quantum devices [15–17]. Furthermore, various cryptography protocols based on computational lattice problems like USV are proposed to be safe against quantum adversaries [12, 39, 40], and any advantage an attacker could gain by distributing quantum circuits changes the security parameters.

* p.andres-martinez@sms.ed.ac.uk; Supported by the CDT in Pervasive Parallelism, funded by the EPSRC (grant EP/L01503X/1) and the University of Edinburgh.

† chris.heunen@ed.ac.uk; Supported by EPSRC Research Fellowship EP/L002388/2. We thank Petros Wallden for useful feedback.

[1] R. P. Feynman, 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, Science **273**, 1073 (1996).
[4] P. Shor, SIAM Review **41**, 303 (1999).
[5] L. K. Grover, in *Annual ACM symposium on theory of computing* (ACM, 1996) pp. 212–219.
[6] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).
[7] J. L. O'Brien, A. Furusawa, and J. Vučković, Nature Photonics **3**, 687 (2009).
[8] M. H. Devoret and R. J. Schoelkopf, Science **339**, 1169 (2013).
[9] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, Rev. Mod. Phys. **81**, 865 (2009).
[10] R. Raussendorf and H. J. Briegel, Phys. Rev. Lett. **86**, 5188 (2001).
[11] D. Deutsch, Proc. Roy. Soc. A **425**, 73 (1989).
[12] O. Regev, SIAM J. Comput. **33**, 738 (2004).
[13] Extracted from Quipper USV-R implementation [31].
[14] R. van Meter and S. J. Devitt, IEEE Computer **49**, 31 (2016).
[15] C. J. Ballance *et al.*, Nature **528**, 384 (2015).
[16] M. S. Blok, N. Kalb, A. Reiserer, T. H. Taminiau, and R. Hanson, Faraday Discuss. **184**, 173 (2015).
[17] quantum-internet.team.
[18] S. Wehner, D. Elkouss, and R. Hanson, Science **362**, eaam9288 (2018).
[19] E. Schoute, L. Mancinska, T. Islam, I. Kerenidis, and S. Wehner, arXiv:1610.05238 (2016).
[20] F. Hahn, A. Pappa, and J. Eisert, arXiv:1805.04559 (2018).
[21] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, in *ACM SIGPLAN Notices*, Vol. 48 (2013) pp. 333–342.
[22] L. Lyaudet, Theoretical Computer Science **411**, 10 (2010).
[23] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, Proc. Alg. Eng. Exp. (ALENEX) **19**, 28 (2017).
[24] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wooters, Phys. Rev. A **54**, 3824 (1995).
[25] R. van Meter, T. D. Ladd, A. G. Fowler, and Y. Yamamoto, Int. J. Q. Inf. **8**, 295 (2010).
[26] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, Phys. Rev. Lett. **76**, 722 (1996).
[27] J. I. Cirac, A. K. Ekert, S. F. Huelga, and C. Macchiavello, Physical Review A **59**, 4249 (1999).
[28] N. J. Ross and P. Selinger, Q. Inf. Comp. **16**, 901 (2016).
[29] C. M. Dawson and M. A. Nielsen, arXiv:quant-ph/0505030 (2015).
[30] A. Yimsiriwattana and S. J. Lomonaco Jr, AMS Cont. Math. **381**, 131 (2005).
[31] www.mathstat.dal.ca/∼selinger/quipper/doc.
[32] github.com/PabloAndresMartinez/Distributed.
[33] B. Coecke and R. Duncan, New J. Phys. **13**, 043016 (2011).
[34] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang, Found. Comp. Sci. (FoCS) **48**, 363 (2007).
[35] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman, Proc. Symp. Th. Comp. (STOC) **35**, 59 (2003).
[36] J. D. Whitfield, J. Biamonte, and A. Aspuru-Guzik, Molecular Physics **109**, 735 (2011).
[37] R. Cockett, C. Comfort, and P. Srinivasan, in *Quant. Phys. Log.*, Elec. Proc. Theo. Comp. Sci., Vol. 266 (2018) pp. 258–293.
[38] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, Ph.D. thesis (1987).
[39] M. Ajtai and C. Dwork, Proc. Symp. Th. Comp. (STOC) **29**, 284 (1997).
[40] O. Regev, J. ACM **51**, 899 (2004).

A. Proof of Theorem

Theorem *Distributions of a quantum circuit with λ_e ebits correspond bijectively to partitions of its hypergraph generated by the pseudocode of Figure 3 with λ_c cuts. A partition can be converted into a distributed circuit in time $\mathcal{O}(n)$, where n is the number of gates in the original circuit.*

Proof First, we provide the 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 define two *primitive transformations* for both 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 *CNOT-primitive* moves CNOT-vertices:

- given a partition of the hypergraph, moving CNOT-vertex α to block i corresponds one-to-one to
- picking CNOT 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 CNOTs to their corresponding block/QPU using the CNOT-primitive once per CNOT, then do the same for the wires using the wire-primitive. Because all partition/distributions can be described this way, and they correspond one-to-one to each other, it is clear that we have a bijection. It remains to prove that the bijection preserves $\lambda_c = \lambda_e$.

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/-circuit on the final partition/distribution. Thus, it is just an artifact to simplify the proof.

2. By construction, each CNOT-vertex is connected to exactly two hyperedges: one of control type and another of target type. When a CNOT-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 CNOT-vertex with whom it shares the same control hyperedge; similarly for the target hyperedge. The same happens for the ebit count λ_e: if, in the QPU where it is reallocated, there is no CNOT with whom it shares a control wire, then an ebit is required to remotely access it, otherwise the channel already exists and no additional ebit is required; the same applies for the target wire. Thus, we may reallocate all CNOTs 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 CNOT-vertex in i. The ebit count λ_e will decrease under the same circumstances, because the CNOTs corresponding to those CNOT-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 $\lambda_c = \lambda_e$.

To complete the proof, we just need to show that a partition can be converted into a distributed circuit in time $\mathcal{O}(n)$, where n is the number of gates in the original circuit. Given a partition, we may apply the method above backwards, reaching the trivial configuration and taking note of the sequence of primitives required to do so, which will be less than $3n$ (at most one per CNOT and each of the wires it acts upon). Then, we just need to apply the sequence forwards to build the corresponding distributed circuit. In total, the process takes $\mathcal{O}(2 \cdot 3n)$ primitives, which is equivalent to $\mathcal{O}(n)$.