Clifford Gate Optimisation and T Gate Scheduling: Using Queueing Models for Topological Assemblies

Alexandru Paler
Linz Institute of Technology
Johannes Kepler University, Linz, Austria
alexandrupaler@gmail.com

Robert Basmadjian
Chair of Sensor Technology
University of Passau, Passau, Germany
robert.basmadjian@uni-passau.de

Abstract—**Clifford gates play a role in the optimisation of Clifford+T circuits. Reducing the count and the depth of Clifford gates, as well as the optimal scheduling of T gates, influence the hardware and the time costs of executing quantum circuits. This work focuses on circuits protected by the surface quantum error-correcting code. The result of compiling a quantum circuit for the surface code is called a topological assembly. We use queuing theory to model a part of the compiled assemblies, evaluate the models, and make the empiric observation that at least for certain Clifford+T circuits (e.g. adders), the assembly's execution time does not increase when the available hardware is restricted. This is an interesting property, because it shows that T gate scheduling and Clifford gate optimisation have the potential to save both hardware and execution time.**

Index Terms—**quantum computing, surface code, topological assembly**

I. Introduction

The efficient design of error-corrected quantum circuits is an open problem, because there is a trade-off between hardware cost and execution time that needs to be accounted for. Very often, for Clifford+T circuits, the goal is to optimise T-count and/or T-depth. However, compiling and optimising error-corrected quantum circuits focuses on more than just the T gates. This work presents preliminary evidence supporting the goal of Clifford gate optimisation. We show that such optimisation methods can have a significant influence on the overheads of error-corrected quantum circuit execution.

A. Surface Code Assembly Volume

The surface quantum error-correcting code (QECC) is at the foundation of the most promising quantum computer architectures. The code tolerates high hardware error rates, and has straightforward architectural requirements [1]. A *topological assembly* (e.g. Fig. 1) is the result of compiling a quantum circuit to surface QECC elements. Compilation requires the circuit being transformed into Clifford+T gates. The surface code operates at a so-called logical layer, and the physical layer can be for example a topological cluster state (3D variant of the code), or a lattice of physical qubits arranged according to well defined 2D nearest neighbour interactions (planar surface code).

There are different surface code techniques (e.g. braiding, lattice surgery [2]). The discussion herein, and the accompanying figures, are based on the assumption that the code is
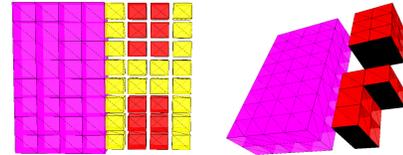

Fig. 1. 3D view of a topological assembly with two layers. (a) A region is for distillations (magenta), data patches are in columns (red), ancilla patches (yellow) are between the data patches. (b) Patches were placed in two layers. The ancilla are not used, and the distillation procedure is being executed. Execution time flows from image background to foreground.

implemented through lattice surgery. Nevertheless, our work is valid for the braided version of the code, too. In lattice surgery, logical qubits are encoded into code patches (e.g. squares in Fig. 2), which have two boundary types (opposite boundaries are of the same type). A quantum computation is executed by choosing on which boundary to interact patches (merged and split). Patch dimensions are dictated by the distance of the surface code.

The overheads (costs) of applying the surface code (in fact any QECC) is quantified by the utilised number of physical qubits, and the generated time overhead. The goal of fault-tolerant quantum circuit optimisation is to reduce the overheads (hardware and time) without sacrificing the strength of the error-correction. The overhead of a circuit's assembly can be abstracted through a 3D spacetime volume (qubits × time). In 3D, one of the dimensions is abstracted time. For the lattice surgery planar code, space (hardware, cf. Fig. 2) is the two-dimensional area necessary for storing all the logical qubit patches.

B. Distillation

The Clifford gates can be implemented fault-tolerantly in the surface code, but the T gates require the support of high fidelity T states. Such states need to be distilled first. A distillation procedure is a sub-circuit which takes multiple low-fidelity instances of the same state and outputs probabilistically a higher-fidelity state – a distilled state. The complexity of implementing surface QECCs originates from the necessary *distillation* procedures [1]. Executing a T gate comes at the additional cost of executing a distillation, which in turn is a computation with relatively high assembly volume. It is an open question how to design low volume distillations [3].
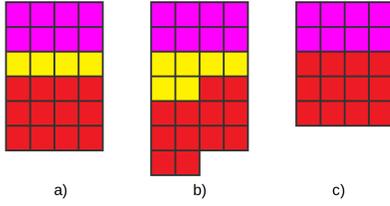
Fig. 2. Assembly layer. Distillery region is magenta. The yellow region stores distilled states (e.g. capacity 4). Clifford+T computation is executed using buffered T states in the red region (e.g. 12 logical qubits). Considering magenta and red regions of fixed sizes, the number of physical qubits is reduced by decreasing the size of the yellow region.

Each distillation has a known duration in time, which is assumed to be equal to a multiple of the code distance. The manipulation of patches in time is represented by a cube, and a distillation is a large cuboid whose volume equals multiple cubic volume units (e.g. in Fig. 1 the magenta cubes are volume units of a *single* distillation that was executed for two time steps, and will finish after additional steps, which are not illustrated).
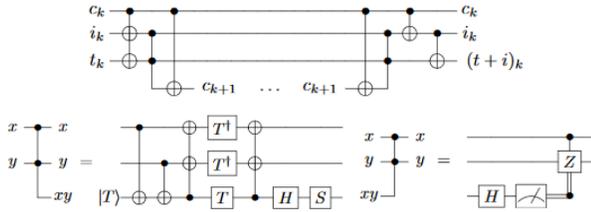

Fig. 3. The considered circuits are n-qubit carry ripple adders. Due to how gates are decomposed into Clifford+T, the resulting circuits include $4n$ T gates. All the T gates appear in the first half of the circuit. However, due to the carry ripple structure, some T gates can be delayed without delaying the overall computation.

C. Worst-Case: T-count equals T-depth

The current generation of quantum machines (almost certain the next generation, too) are resource restricted, and will support only sequential distillations (parallel would require too many physical qubits). This effectively calls for the adaptation of the Clifford+T circuits: the initially parallel T gates (e.g. Fig. 3) have to be *scheduled such that their sequential execution introduces a minimal time overhead into the assemblies*.

In hardware restricted scenarios it is assumed that the worst case execution time of a circuit is bounded by the T-count (when all T gates are executed sequentially T-depth equals T-count). In general, an assembly's depth (time, cf. *circuit depth*) is a function of the number of Clifford+T gates in the circuit. Worst-case estimations of assembly volumes use, as an indication for depth, the circuits' T-count (the number of T gates in the circuit) [4].

D. Problem Statement

The following definitions are used to state the problem. Considering Fig. 1, a *layer* (e.g. Fig. 2) is a slice of the assembly made at a given time coordinate. A *distillery* is the assembly region (cf. magenta in Fig. 2) where distillations are

executed. T gates are executed when a distillation succeeded, and a distilled T state was obtained. It is reasonable to store distilled states into a *buffer*. Buffering distilled T states has the potential of shortening the depth of the topological assembly, because T gates would be parallelised. However, this comes at the expense of hardware overheads (buffer sizes).

Patches (red or yellow) are interacted by manipulating their boundaries (merges and splits). In Fig. 1, red patches represent logical qubits, and yellow patches are ancillae (used to intermediate between red patches). This work focuses on distilled states, and we consider that the yellow patches represent distilled T states, and that computational ancillae are red. This graphical simplification (cf. Figs. 1 and 2) does not affect the generality and conclusion of the analysis. Thus, the yellow region is a visualisation of the buffer. The number of yellow patches in the region indicates the buffer size.

How can the trade-off between hardware and time be explained through the layout of the topological assembly? Does the buffer size influence the execution time? The same question can be reformulated as: Is T-depth always the worst case? This work will provide evidence that for some circuits the assembly depth is larger than T-depth.

II. METHODS

We present a methodology based on a straightforward queuing model of the distilled states buffer. The methodology is general and applicable to any Clifford+T quantum circuit. We use quantum adders to collect empirical evidence that scheduling T gates, as well as smart distillery control mechanisms lead to computational resource savings. The analysis will show that buffers are not necessary as long as *T gates are perfectly scheduled (on average, one T gate per distillation execution)*. If perfect scheduling is not possible, then the proposed methodology is applicable to:

- determine an optimal buffer size;
- compute the optimal point in time to turn off a distillery, and use the extra available space to perform the Clifford gate optimisation.

The later application implies that the distillery can be stopped before the buffer reaches full capacity, and the available space (yellow patches) is used to speed-up the Clifford part, such that the next T gate will be executed sooner. Thus, the queue based methodology offers quantitative insights into the necessity of future work on Clifford gate optimisation and T gate scheduling.

A. Related work

Although not explicitly stated, the need for Clifford gate optimisation was observed in [5]. The therein analysed circuits were Clifford dominated, and the T-count was not the upper bound of the worst case resource estimation. Clifford dominated circuits appeared in [6], too. Because of their structure, efficient scheduling of T gates became a necessity: distribute (commute where possible) T gates, such that there is no need for buffering distilled T states – the buffers would have occupied hardware resources. T gate scheduling implied
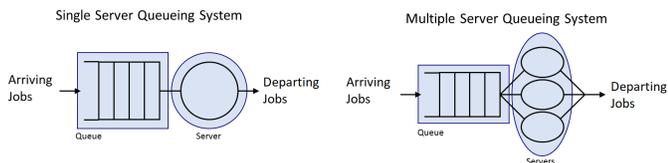
Fig. 4. An example of a single and multiple servers queuing systems.

Fig. 5. DTMC for distillery buffer capacity.

packing Clifford gates between the sequentially executed T gates (see worst case above).

The need for T gate scheduling resulted in the need of controlling distilleries, and the first investigations have been presented in [7]. Therein, the buffer had a fixed size, and could store seven distilled states. Once the buffer was full, the distillery was stopped (no distillations were started until the first buffered state was not consumed by a T gate). Otherwise, the buffer would have overflown. In that work a simplistic look-ahead strategy for calculating the number of next T gates, and necessary distilled T states, was presented. That strategy was used to predict when the buffer would be full.

Nevertheless, previous work has not answered the following questions: 1) are T state buffers always needed? In case they are, what will be the optimal buffer size? How can the buffer size be computed? Can the need for buffering be circumvented? In [7], once the buffer capacity was capped, the distillery control did not generate additional time overheads in the resulting assemblies.

B. Distillery and Buffer

We introduce an analogy, based on queuing theory, for capturing how the management of distillation procedures influences the arrangement of patches in the assembly. The analogy will help answer the trade-off question by looking at buffer occupancy.

In the case of queuing systems, *jobs* (also known as requests, customers, or calls) are stored temporarily in a *buffer* (also known as queue) until the server finishes executing the current job. Thus, a server can execute one job at a time. Buffers can be classified based on their capacity: *finite* or *infinite*. Unlike infinite buffer capacity, the finite one has a limited buffer size.

A *server* is a job processing unit. Upon finishing the execution of a job, a server retrieves the next job from the buffer based on a given *strategy* (e.g. FCFS – first come, first serve), and executes the job. Parallel job execution is achieved through multiple servers queuing system. Thus, a queuing system consists of one buffer and at least a single server. Figure 4 illustrates a graphical presentation of queuing systems consisting of a single and multiple servers. This work considers simple systems with one buffer and one server (e.g. single server queuing system).

In the case of topological assemblies, the job analogue is the *distilled T state*. The buffer's analogue is the *buffer region*. The capacity is the number of distilled states that fit into the buffer. Thus, infinite capacities are not realistic. The equivalent of the server is the region that consumes distilled T states.
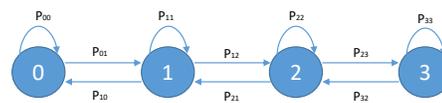
Whenever the buffer reaches its full capacity (e.g. overflow), then either the new arriving job is lost or the job producer (e.g. distillery) is notified about not to send new jobs until there is enough available capacity. In this paper, if such a buffer overflow happens, we consider the second option of stopping the production of new jobs by the distillery.

We propose the following methodology to identify the optimal buffer size for a given circuit: 1) the yellow region is modelled as an infinite buffer; 2) the behaviour of the distillery is emulated with maximum one distillation at a time; 2a) buffer occupancy increases by maximum one at a time, because of the sequential distillation process; 2b) buffer occupancy decreases by maximum one at a time, because circuit gates are not parallel (in this work).

The methodological steps are based on Markov chains, where each state is *the number of jobs in the system* (the ones kept in the buffer, and the one used for T gate). This adoption is valid since our problem satisfies the memory-less property: *future depends only on the current state and not: (1) on the history of states, and (2) how much time is spent in the current state*. The topological assembly has a discrete structure along the time axis, and this leads to the formulation of Discrete Time Markov Chains (DTMC, cf. Fig. 5): the state-space (e.g. $0, 1, 2, \ldots, n$ jobs) and the time-space (e.g. $1, 2, \ldots, n$ time instance) are discrete.

The resulting DTMCs are *ergodic* because (1) these have a finite number of states N, (2) their states are pairwise reachable, (3) the chain is aperiodic (e.g. no periodicity of a certain state exist). The ergodicity of a given DTMC ensures the fact that there exists a unique steady-state probability.

C. Buffer Size, Number of Transitions

Quantum hardware is not an abundant resource, and we analyse the trade-off between optimal buffer capacity, denoted by B, and the time overhead. This is an iterative process (cf. Fig. 6) in which circuit execution is re-emulated for values in the range $b \in [0, B)$, which will lead to *different number of transitions and different transition probabilities* (incorporating the resulting time overheads, e.g. increased assembly depth). *The worst-case execution time is obtained for the most hardware efficient setting when $b = 0$ (e.g there is no buffer).*

The values of transition probabilities $P_{i,j}$ (in matrix form) are obtained through a process which we call *(re)-emulation* (straightforward, linear procedure in which the usage of the buffer is simulated, as distilled states are produced, stored and used). Emulation is not quantum circuit execution or simulation. The methodology iterates the emulation of the corresponding circuit for different number of qubits for infinite and fixed (e.g. 7, see next Section) buffer capacities.
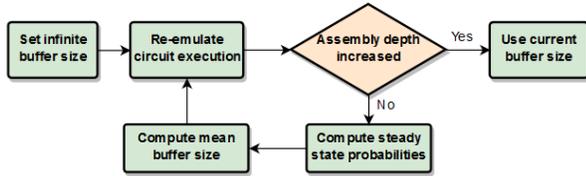
Fig. 6. The process of determining optimal buffer size.

TABLE I
RESULTS FOR INFINITE AND SIZE-7 BUFFERS.

Qubit	Mean # of Jobs		Total # of States (Infinite)	Utilisation	Number of Transitions
	Size-7	Infinite			
16	2.80	2.96	9	69%	270
32	3.85	6.51	19	73%	558
64	4.35	13.61	37	76%	1134
128	4.59	27.83	73	77%	2286
256	4.71	56.28	147	77%	4590
512	4.77	113.17	293	78%	9198
1024	4.80	226.94	585	78%	18414
1536	4.80	340.72	878	78%	27630
2048	4.82	454.5	1171	78%	36846

The $P_{i,j}$ transition probabilities are used to compute the *unique steady-state probabilities*, based on the classic system of equations $\vec{\nu} = \vec{\nu}P$, such that $\sum_{\forall i} \nu_i = 1$ ([8, p. 41]). The transition probability $P_{i,j}$ denotes the probability of reaching state j from state i.

Once those steady-state probabilities are calculated [9], [10], it is possible to compute among other the following metrics, which are at the foundation of evaluating the performance of a queuing system:

1) the probability of the system being idle v_0 or full v_n,
2) mean number of jobs in the system $\bar{K} = \sum_{i=1}^{\infty} k \cdot v_i$, where k denotes the current state,
3) the average utilisation of the whole system $\bar{U} = 1 - v_0$, where v_0 is the steady-state probability of an idle system.

III. RESULT: BUFFERING DOES NOT SHORTEN DEPTH

The influence of the buffer size on practical and commonly used quantum addition circuits [11] is presented in Table I. It should be noted that, as mentioned in the caption of Fig. 3, the distribution of T gates in the original adder is not perfect. *The circuits we analysed in this work have been previously partially optimised by hand with respect to their T gate scheduling. Nevertheless, it was not obvious from their structure what the optimal buffer capacity will be, or what their maximum assembly depth is.*

To evaluate and analyse the impact of buffer sizes, we setup two configurations for the queuing system: finite buffer size of 7, and infinite capacity. The raw information about distillation behaviour and buffer usage were calculated using SurfBraid[1], and a customised Python tool[2] to compute the steady states and the average buffer sizes.

Both cases (finite and infinite) have the same total number of transitions as well as utilisation rate. Regarding the number

of transitions: it almost doubles with each doubling of qubit numbers. As circuits get larger, these include more gates. The surprising observation is that irrespective of the buffer size, the execution time of the adders stays the same for the same number of qubits – no delays by executing sequential T gates. This same doubling behaviour is also evident for the total number of states in the case of infinite buffer capacity. Note that for the case of finite buffer size, this is not happening because the maximum number of states equals buffer capacity plus one (from 0 to 8).

Regarding utilisation, it is almost constant and ranging between 70% and 78% both for finite and infinite buffer size independent of the qubits. This indicates that the buffer is on average empty for around 20% of the time. Furthermore, the probability that the buffer is full (e.g. v_n) never passed 0.05 (e.g. for 16 qubit adder). This denotes that the system never reached to a buffer overflow situation.

By further investigating the mean number of jobs, which is calculated based on steady-state probabilities (omitted from Table I), the obtained results show that for a finite buffer size of 7, the average was almost constant for all circuit sizes (e.g. ranging from 2.8 till 4.8) for any number of considered qubit. This further motivates the fact that for the considered circuit there is no need to allocate large buffer sizes.

Using the methodology from Fig. 6, we reached the conclusion that, for the investigated adders, no buffer is in fact necessary because: a) no buffer overflows were recorded while re-emulation the circuit, b) the assembly depth did not increase even for capacity zero buffers. Consequently, the hardware requirements from buffer perspective can be reduced and optimised adequately.

IV. CONCLUSION

The analysed adder has the property that the sequential execution of T gates does not influence the depth of the resulting assembly (execution time). We conjecture that this property is because of the advantageous scheduling of the T gates inside the adders: *the distillery produces states faster than the main computation is consuming them.*

Another interesting property of the adder is that its *depth is not dominated by the T-count*. The assembly depth is larger than the one computed by looking at the T-count. Thus, *Clifford gate optimisation* is a realistic option for circuit optimisation. Future work directions are:

1) Optimal scheduling methods for T gates;
2) Optimisation of the Clifford part (reduce average number of Clifford gates between two subsequent T gates ≡ adapt to distillery speed);
3) Queuing-based models to assist in the automatic design and analysis of the resulting assemblies.

References

[1] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[2] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[3] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.

[4] Craig Gidney and Austin G Fowler. Efficient magic state factories with a catalyzed $|CCZ> $ to 2 $|T>$ transformation. *arXiv preprint arXiv:1812.01238*, 2018.

[5] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on sha-2 and sha-3. In *International Conference on Selected Areas in Cryptography*, pages 317–337. Springer, 2016.

[6] Ryan Babbush, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear T complexity. *Physical Review X*, 8(4):041015, 2018.

[7] Alexandru Paler. Controlling distilleries in fault-tolerant quantum circuits: problem statement and analysis towards a solution. In *2018 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–6. IEEE, 2018.

[8] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. WileyBlackwell, 2nd edition edition, May 2006.

[9] Robert Basmadjian, Florian Niedermeier, and Hermann de Meer. Modelling performance and power consumption of utilisation-based dvfs using m/m/1 queues. In *Proceedings of the Seventh International Conference on Future Energy Systems*, e-Energy '16, pages 14:1–14:11, New York, NY, USA, 2016. ACM.

[10] Robert Basmadjian and Hermann de Meer. Modelling and analysing conservative governor of dvfs-enabled processors. In *Proceedings of the Ninth International Conference on Future Energy Systems*, e-Energy '18, pages 519–525, New York, NY, USA, 2018. ACM.

[11] Craig Gidney. Halving the cost of quantum addition. *Quantum Journal*, 2, 2018.