Space-depth tradeoffs in parity synthesis for quantum computing

Arya Maheshwari

Department of Computer Science, Princeton University

Adviser: Dr. Ali Javadi-Abhari (IBM Quantum)

PACM Reader: Professor Noga Alon (Princeton University)

Abstract

We study circuits for the parity synthesis subroutine in quantum computing from a space-depth tradeoff perspective, motivated by applications in e.g. synthesis of Hamiltonian simulation circuits. We first overview two existing approaches at opposite ends of the space vs. depth spectrum. Our main contribution is to then provide a framework that fills in the gap between these two extremes, by enabling finer-grained control over the space-depth tradeoff using a tunable parameter c. Our framework builds upon the block extension algorithm of [BBV+21] by leveraging additional ancilla to proportionally parallelize the existing computations.

1 Introduction

Quantum computing (QC) is a paradigm with the potential to transform computer science, yet its practical capabilities are hindered by physical limitations of current quantum computers, leading to significant resource constraints. For instance, applying a gate to a qubit—the basic computational operation in QC—is a noisy process in practice. This noise compounds across multiple operations, leading to increasing errors as the number of gates required to perform a computation grows. Another particularly challenging problem is that qubits only stay *coherent* for short periods of time, due to inevitable interactions with the external environment that disturb the qubits' state. This *decoherence* limits the time for which a quantum computer can feasibly be run in practice before starting to incur significant errors.

Quantum compilation aims to bridge this gap between theoretical procedures and practical implementations by analyzing how an algorithm, expressed as a quantum circuit, can be optimized and implemented in a more resource-efficient way. More specifically, the goal is to optimize some metric of circuit resources, such as the number of qubits used; the total gate count; or the circuit depth (i.e. the number of *layers* of gates in a circuit). In this project, we will focus on circuit depth. This metric roughly captures the time required to execute a circuit, as each layer of gates represents a step of operations that act on disjoint qubits and thus can be executed in parallel. (Concretely, for instance, the circuits shown in Figure 2 and Figure 3 both have depth 3 despite the circuit in Figure 3 having more gates, as these gates can be implemented in parallel as shown.) As a result, optimizing circuit depth is an important method for coping with the challenge presented by qubit decoherence.

Some basic, myopic techniques in quantum compilation include removing groups of gates that cancel each other out or reordering commuting gate blocks to minimize depth, but more sophisticated and powerful techniques can be developed for specific classes of problems. One such class of problems of central importance in QC is Hamiltonian simulation. This is the task of implementing a circuit to model the evolution of a quantum system governed by a given Hamiltonian. Many Hamiltonians of interest can be expressed as a sum of *Pauli operators*, in which case the problem boils down to implementing a sequence of *Pauli rotations*. Modulo some gates required for basis changes, the main components of implementing each Pauli rotation are computing a corresponding parity between qubits through a ladder of CNOTs (a type of 2-qubit gate); applying a single-qubit rotation (e.g. via an R_Z gate) on that parity; and then uncomputing the parity through a reverse ladder of CNOTs.[1]

From a compilation perspective, the bottleneck in this procedure is the set of CNOT gates, as these 2-qubit gates are more costly to implement than the single-qubit rotation gates in common physical QC systems like superconducting- and trapped ion-based QC [MZ22, DLF+16]. Thus the optimization objective often boils down to the *CNOT depth* of the circuit, i.e. considering the network of CNOTs used to construct the required parities and ignoring the single-qubit rotations interspersed throughout the overall circuit. In particular, we can reduce the problem for our purposes to the task of synthesizing a *parity network*

[1] We will introduce some of the key concepts relevant to our specific project, like CNOT gates, in Section 2.

[AAM18, VMGDB22], or a CNOT circuit where each desired parity are present *at some point* throughout the course of the circuit the circuit. The optimization goal is then to construct a parity network with as low depth as possible.

In this project, we study a variant of this parity network synthesis task where we instead require all parities to appear *simultaneously* in the circuit. In particular, we will suppose that in addition to the original "data" qubits, we have access to a set of *ancilla* qubits, and we then want to synthesize the parities onto these ancilla such that all parities exist together at the same time on these ancilla (rather than being interspersed at various points in the circuit).[2]

Our motivation for studying this variant of parity synthesis comes from a broader agenda of applying techniques like *measurement-based quantum computing* (MBQC) to optimize Hamiltonian simulation circuits. Very briefly, conditional corrections that arise in MBQC induce a specific order in which Pauli rotations must be executed, which is then not compatible with the standard parity network problem, where there is no guarantee on the order in which parities arise. If we instead have all parities present simultaneously on ancilla, then rotations can be executed in *any* desired order. Another motivation for this variant is the case of "CNOT+T" circuits [BBV+21, AMM13], where it is very beneficial to execute the costly[3] T gates—each associated with a parity—in parallel, thus suggesting the same benefit from enforcing that all parities be present simultaneously. For the purposes of this project, however, we will abstract our problem away from this underlying motivation and focus our attention on the modular problem that we define formally in the next section.

1.1 Our Problem

Formally, we consider the ParallelParities problem, defined as follows.

Problem: ParallelParities	
Input:	n data qubits $\mathbf{x} = \{x_1, ..., x_n\}$; p parities $\{f_1(\mathbf{x}), ..., f_p(\mathbf{x})\}$ over the data qubits; m ancilla
Output:	In minimal depth, synthesize the p parities on ancilla *simultaneously*

Given n data qubits $\mathbf{x} = \{x_1, ..., x_n\}$, p parities $\{f_1(\mathbf{x}), ..., f_p(\mathbf{x})\}$ over the data qubits, and some number $m \geq p$ of available ancilla qubits, the goal is to construct a circuit of minimal depth in which the p parities are synthesized onto the ancilla such that they are present *simultaneously*, that is, all parities are stored on ancilla qubits at some single point in the circuit.

Observe that the number m of ancilla is an input to the problem here. In general, using more ancilla provides more resources for a quantum circuit to exploit: for instance,

[2] Note that, letting n be the number of data qubits, one could also consider synthesizing n of the parities on the data qubits (the number of parities required is often larger than n in practical use cases). We choose to state our model and problem in terms of synthesis only onto ancilla qubits for clarity and simplicity of the resulting ideas.

[3] T gates are highly costly in fault-tolerant settings. See [AMM13] for further discussion.

ancilla qubits can store intermediate computations and can enable executions of additional operations in parallel. It is thus natural to expect—or, at least, hope—that as the number of ancilla available increases (i.e. larger m), the depth necessary to implement a circuit achieving ParallelParities should decrease commensurately, if any parallelization across the ancilla can be leveraged. The problem formulation thus suggests a tradeoff between resources, with ancilla usage (which we refer to as a circuit's *space* usage) on one end and circuit depth (which is a proxy for circuit execution time) on the other.

Our specific goal in this project is to develop a better quantitative understanding of this tradeoff between space and depth in the ParallelParities problem. Our main contribution, presented in Section 4, is to provide a simple framework that allows this space-depth tradeoff to be *controlled* at a finer-grained level than before. This framework enables instance-specific allocation of space and depth costs, thereby making the optimization of circuit resources for this variant of the parity synthesis problem much more practically feasible than before.

2 Background and Preliminaries

Notation. We first fix some basic notation and conventions. All logs will be base 2. For a m-by-n matrix A, $A[S]$ denotes the subset of rows indexed by the set S (for some $S \subseteq [m]$). Similarly $A[r_1 : r_2]$ denotes the submatrix defined by rows r_1 through r_2 inclusive. $A[r_1 : r_2; c_1 : c_2]$ denotes the submatrix defined by rows r_1 through r_2 inclusive and columns c_1 through c_2 inclusive. We use $\oplus$ to denote addition in $\mathbb{F}_2$, i.e. the addition to compute a parity. We will denote a m-by-n Boolean (i.e. over $\mathbb{F}_2$) matrix A as $A \in \mathbb{F}_2^{m \times n}$. We will sometimes use "wires" to refer to the circuit wires (indicated by horizontal lines in e.g. Figures 1 through 3) that carry qubit values, as this is sometimes useful to distinguish from the original logical values of the data qubits themselves. Figures 1 through 3 are examples of *quantum circuit diagrams*: the qubits/wires are denoted by the horizontal lines, and gate operations that act on these qubits are then drawn left to right in order of application.

Model. We now describe the model in which we study the ParallelParities problem, following the same general model used in prior literature on CNOT circuit synthesis (e.g. [PMH03, BBV+21, MZ22]). As a high-level summary, our model is formulated at a level of abstraction where the synthesis task in ParallelParities turns into a classical algorithm design problem in terms of Boolean matrices. First, we use $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$ to denote the n data qubits. The key abstraction we exploit in our problem is that we only care about computing parities between the different qubits, and hence we will simply think of each x_i as an element of $\mathbb{F}_2$ (i.e. like classical bits). We will then denote a *parity* over the qubits $\mathbf{x}$ by $f_j(\mathbf{x})$. We slightly abuse notation in that we generally use $f_j(\mathbf{x})$ to refer to a sum (over $\mathbb{F}_2$) of some set of x_i's, e.g. $f_j(\mathbf{x}) = x_1 \oplus x_2 \oplus x_4$, but we occasionally use the same notation to represent a parity as a vector in $\mathbb{F}_2^n$ where the one values represent which x_i's are part of the parity sum. We do this to simplify notation, as the meaning will be clear from context.

Next, an *ancilla* qubit represents an additional "helper" qubit used as an extra resource in a circuit. Ancilla qubits start in the zero state, denoted $|0\rangle$. The *controlled-NOT* (CNOT) gate is a 2-qubit gate that acts on a *control* qubit and a *target* qubit, depicted in Figure 1. The action of a CNOT is to add the control qubit parity into the target qubit, i.e. it flips the target conditioned on the control being in the one state. Formally, for control qubit x_i and target qubit x_j, $\text{CNOT}(x_i, x_j) = (x_i, x_i \oplus x_j)$.[4]

Figure 1: A single CNOT gate with control qubit x_i and target x_j.

CNOT circuits and linear reversible transformations. In this model, we can then represent circuits of CNOT gates over n data qubits as *linear reversible transformations* (over $\mathbb{F}_2$), concretely specified by an invertible n-by$n$ Boolean matrix. First, observe that we can encode a single $\text{CNOT}(x_i, x_j)$ as a left-multiplication of the state $\mathbf{x}$ by the matrix $A_{ij} = I_n + e_j e_i^\top \in GL_n(\mathbb{F}_2)$, where $e_k \in \mathbb{F}_2^n$ denotes the one-hot vector with a one at index k and zeros everywhere else. In words, A_{ij} is the identity matrix with an additional 1 in row j, column i. The transformation $A_{ij}\mathbf{x}$ thus has the effect of changing x_j to $x_i \oplus x_j$, as desired. A CNOT circuit is then simply a product of such matrices, which overall will be some matrix $A \in GL_n(\mathbb{F}_2)$.

In fact, there is a stronger correspondence: conversely, *any* linear reversible transformation can be implemented concretely as a CNOT circuit. Consider any invertible Boolean matrix $A \in GL_n(\mathbb{F}_2)$. We think of each row i of A as encoding the final parity desired on wire i (where the value at column j specifies whether the parity includes the jth data qubit). Recall that any invertible matrix can be expressed as a product of *elementary* matrices, which are the matrices corresponding to the three elementary row operations: row multiplication (by a nonzero scalar); row addition; and row swapping. The first operation is vacuous over $\mathbb{F}_2$. Next, the key is that row addition operations are given exactly by the matrices corresponding to CNOTs, as defined above: $\text{CNOT}(i, j)$ is the addition of row i to row j. Finally, row swapping over $\mathbb{F}_2$ can simply be implemented as three row additions. This is visualized through an annotated circuit of CNOTs in Figure 2. Thus any A can be expressed as a product of row addition operations, that is, implemented as a sequence of CNOT gates.

Linear reversible synthesis routines over n qubits. As a result of this correspondence, such CNOT-only circuits over n data qubits are also called *linear reversible circuits*, and the task of synthesizing a given n-qubit linear reversible circuit can be abstracted into the problem of efficiently constructing the corresponding invertible Boolean matrix from

[4]Note that we do not discuss the standard general formalism in quantum computing where an n-qubit state is represented by a 2^n-dimensional vector of *amplitudes* and any n-qubit gate is then given by a 2^n-by-2^n unitary matrix. This is because our problem focuses only parity synthesis and CNOT gates, which can be expressed more concisely via n-by-n Boolean matrices, as [PMH03] discuss. This enables us to work in a classical model of parities rather than a general quantum model of qubits and quantum gates.

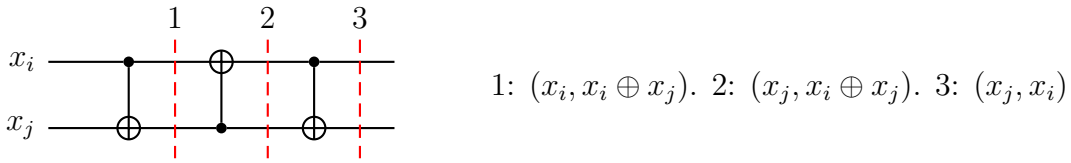

Figure 2: Row swapping with three row additions (CNOTs). The annotations show the parity states at each point in the circuit: $(x_i, x_j) \to (x_i, x_i \oplus x_j) \to (x_j, x_i \oplus x_j) \to (x_i, x_j)$.

row addition operations. This problem has been studied quite extensively in the quantum compilation/synthesis literature (e.g. [PMH03, BBV+21, JST+20, MZ22]) given the importance of optimizing blocks of CNOT gates that appear as subcircuits of various procedures. Particularly relevant to our project is the existing work on developing n-qubit linear reversible synthesis routines that minimize the depth of the resulting CNOT circuit. In this vein, [JST+20] present an algorithm for n-qubit linear reversible synthesis that achieves an asymptotically optimal depth of $\Theta(n \log n)$, matching the lower bound implied by a counting argument in [PMH03]. This procedure, however, is not efficient in practice due to relatively high constant factors; indeed, in the regimes of current applications where the number of qubits n is moderate, asymptotics do not necessarily capture concrete efficiency. Instead, an algorithm due to Maslov and Zindorf [MZ22] (hereafter MZ) achieves state-of-the-art depth performance in the regime where $70 \leq n \leq 1,345,000$, guaranteeing depth upper bounded by

$$d_{MZ}(n) := n + 2\log^2 n + 3\log n.$$[5]

Remark. We will use n-qubit linear reversible synthesis (instantiated with e.g. the MZ algorithm) as an important (blackbox) subroutine in our approach, but we emphasize that it does *not* just directly solve the same problem as our parity synthesis task in ParallelParities. For instance, note that the number p of parities we need to synthesize (onto ancilla) may be $\gg n$. In addition, the final parities on qubits in n-qubit linear reversible synthesis must all be *linearly independent* (by virtue of the reversibility, i.e. the invertibility of the corresponding Boolean matrix); we have no such guarantees on our set of parities over data qubits, which may be highly dependent.[6]

Logarithmic depth spreading. We conclude our preliminaries with a simple but handy building block that we will use for spreading some qubit x_i's parity value to a set of wires that start in the $|0\rangle$ state. That is, we want to map a set of wires starting in state $(x_i, 0, 0, \ldots, 0)$ to $(x_i, x_i, \ldots, x_i)$. (This can be thought of as a special case of so-called quantum FanOut, with the additional guarantee that all other wires start in the $|0\rangle$ state.)

If there are k wires, then this can be done naively with k sequential CNOTs from x_i to each of the $|0\rangle$ wires. This requires depth k, since clearly none of these CNOTs can be parallelized.

[5] This bound is slightly loser but cleaner in terms of coefficients relative to the exact bound given in [MZ22].

[6] One option would be to treat ancilla qubits as data qubits (with the knowledge that they start in state state $|0\rangle$ and hence wouldn't affect the actual desired parities), as the parities would be "independent" when viewed as including ancilla qubits. We could then directly run a $(n+p)$-qubit linear reversible synthesis. We will return to this in Section 3.1 and see why it is not a good idea.

Alternatively, we can spread x_i through a binary tree-like spreading procedure across the wires, by using CNOT target qubits as controls for future CNOT layers to parallelize the computation (rather than only controlling on x_i). An example is visualized in Figure 3, for $k = 7$ wires. In general, spreading to k wires in this way requires depth $\lceil \log(k+1) \rceil$. Observe that the same idea can be applied in reverse as a straightforward way to compute a single parity from a set of wires. Indeed we will use this idea in both directions in upcoming procedures.

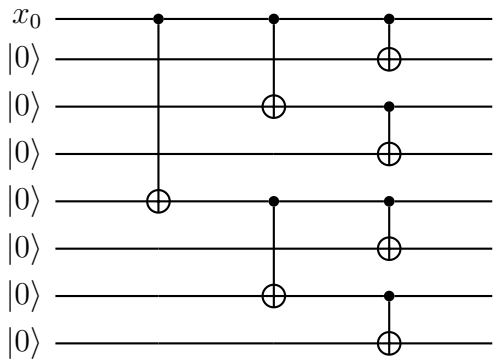

Figure 3: Log-depth spreading procedure through a binary tree-like CNOT structure: spreads a qubit x_i to k wires in $\lceil \log(k+1) \rceil$ depth. Here, $k = 7$.

3 Existing Approaches: Two Extremes

We now present two approaches to ParallelParities based on existing ideas at two ends of the space vs. depth spectrum. The first, in Section 3.1, can be viewed as optimizing for space usage, i.e. using a minimal number of ancilla, and the algorithm therein (due to [BBV+21]) is the main routine upon which we build our controllable framework. The second approach, in Section 3.2, is a simple idea for how one might try to achieve minimal depth given ample access to ancilla.

3.1 Minimizing ancilla usage

We first consider a minimal usage of ancilla. Given the requirement that all parities be present simultaneously on ancilla, we need at least one ancilla for each parity. Thus we suppose we have only $m = p$ ancilla available as a minimal amount. A straightforward idea would be to directly run a linear reversible synthesis routine over all $n + p$ qubits. In particular, this would involve treating each ancilla qubit as a "dummy" data qubit that is part of the corresponding parity to be stored on its wire: this would make the final parities "linearly independent" as needed for reversible synthesis, while the ancilla (starting in $|0\rangle$) would not affect the actual desired parities. However, using this naive approach with e.g. the MZ algorithm for linear reversible synthesis would incur depth linear in $(n+p)$, where p may be $\gg n$. We will see that we can do much better, even when restricted to just p ancilla, with

a fairly simple procedure due to [BBV+21] (Section 5) that involves parallel applications of linear reversible synthesis.

We start by viewing this task as an *isometry synthesis* from n data qubits to $n+p$ total qubits, with p ancilla qubits initialized to $|0\rangle$.[7] We frame the problem in terms of Boolean matrices similarly to the correspondence discussed Section 2—but with a modified representation that enables us to actually distinguish the n data qubits and p ancilla starting in $|0\rangle$, unlike in the strawman approach in the previous paragraph. In particular, we now view the state of the parities on wires as an $(n+p) \times n$ *rectangular* Boolean matrix, where each row i represents the parity stored on a wire i expressed over the n data qubits, with column j corresponding to jth data qubit. We will refer to this rectangular representation as the *parity state matrix* (or just *parity state*) of the qubits. In particular, the initial parity state is given by $A_{in} \in \mathbb{F}_2^{(n+p)\times n}$ where $A_{in}[1{:}\,n] = \mathbb{I}_n$ and $A_{in}[n+1{:}\,n+p] = \mathbf{0}$. The final parity state desired is given by $A_{out} \in \mathbb{F}_2^{(n+p)\times n}$ where $A_{out}[1:n] = \mathbb{I}_n$ and $A_{out}[n+i] = f_i(\mathbf{x})$.[8] These matrices are visualized in Figure 4. The goal is then to transform A_{in} into A_{out} with a CNOT circuit C applied over the $n+p$ qubits. Such a CNOT circuit can be represented as a matrix $C \in \mathbb{F}_2^{(n+p)\times(n+p)}$—based on row additions on the identity matrix, just as in Section 2—such that $A_{out} = CA_{in}$.

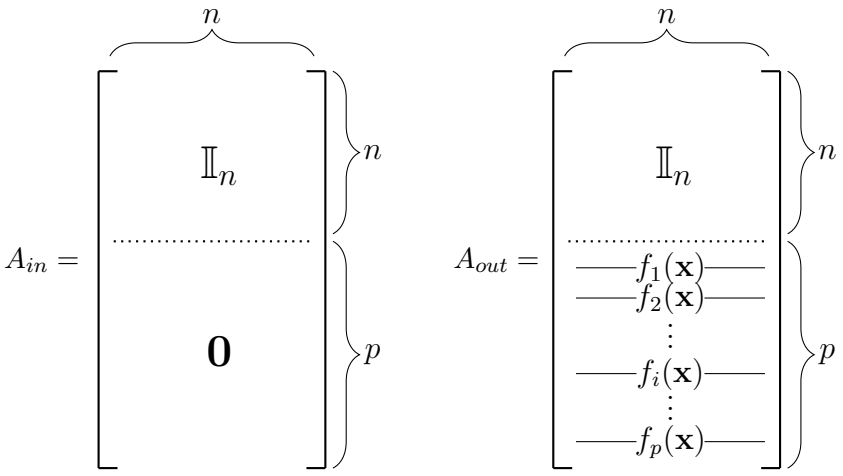

Figure 4: Synthesis with $m := p$ ancilla: initial and final parity states.

We can synthesize such a CNOT circuit C by applying a *block extension* algorithm proposed by [BBV+21], which we will refer to as the Block algorithm. This algorithm takes as input a routine for synthesizing an n-qubit linear reversible circuit (treated as a blackbox) and yields an upper bound on the depth for implementing C as a function of the depth of the input routine, as presented in the following lemma. Note that we present main

[7] For formal details on isometry synthesis, we defer the interested reader to e.g. [SYJ23].

[8] Depending on the setting it may or may not be necessary to return the data qubits to their original parity state as specified in A_{out} here. In the algorithm we describe this turns out not to affect the final depth bound. Furthermore, the order of the parities on the ancilla qubits does matter, so it may be possible to group parities in an instance-specific way to achieve slight depth gains.

results (both this existing algorithm and our new framework) under assumptions that various parameters are divisible as needed for simplicity of exposition. We will later briefly address in the Appendix how the general cases without divisibility assumptions can be handled (Appendix A.1 and A.2).

Lemma 3.1 (Block algorithm [BBV+21]). *Suppose n divides p. Let $d(n)$ be an upper bound on the depth required to implement an n-qubit linear reversible transformation. Then a CNOT circuit C that transforms $A_{in} \in \mathbb{F}_2^{(n+p)\times n}$ to $A_{out} \in \mathbb{F}_2^{(n+p)\times n}$ (that is, $C \in \mathbb{F}_2^{(n+p)\times(n+p)}$ such that $A_{out} = CA_{in}$) can be synthesized in depth upper bounded by $d(n) + 2\lceil \log(\frac{p}{n} + 1)\rceil$.*

We can apply the Block algorithm with the MZ routine for implementing linear reversible circuits, which has a depth upper bound of $d_{\mathrm{MZ}}(n) = \lfloor n + 2\log^2 n + 3\log n \rfloor$ for $70 \leq n \leq 1,345,000$. This immediately yields the following concrete bound.

Corollary 3.2 (Block algorithm [BBV+21] with MZ). *Suppose n divides p. For $70 \leq n \leq 1,345,000$, a CNOT circuit C that transforms A_{in} to A_{out} can be synthesized in depth at most $\lfloor n + 2\log^2(n) + 3\log(n) \rfloor + 2\lceil \log(\frac{p}{n} + 1)\rceil$.*

We review the Block algorithm here for completeness, as it also underlies our main contribution. The key idea is to divide the $(n+p) \times n$ parity state matrix into $n \times n$ blocks that can be efficiently operated on in parallel with a linear reversible transformation. In particular, with only logarithmic depth overhead, each $n \times n$ block of the input and output parity states can be made full-rank, and then we can transform each modified input block into the corresponding output block by synthesizing the appropriate CNOT circuit. The motivation for working with full-rank blocks comes from the fact that a linear reversible circuit is represented by an *invertible* matrix over $\mathbb{F}_2$ (as discussed in Section 2), and hence applying our input linear reversible synthesis routine on an input block can only yield an output block with the same rank. A priori the rank of each output block of the desired parities can be arbitrary, so we modify each input and output block into a full-rank version in order to ensure equality between ranks. Then the input blocks can be transformed into the corresponding output blocks through parallel CNOT circuits $C_i \in GL(\mathbb{F}_2)$ via linear reversible synthesis over the qubits of each block i. Putting all of these component circuits together yields the final desired circuit C. Overall, the point is that parallelizing over blocks leads to the depth being dominated by a $d(n)$ term (Lemma 3.1) rather than a much costlier $d(n+p)$ term.

Proof of Lemma 3.1. Let $b = \frac{p}{n} + 1$. For $1 \leq i \leq b$, let $Q_i = \{(i-1)\cdot n + 1, \ldots, i \cdot n\}$ represent a n-subset of the $n+p$ total qubits. We divide $A_{in} \in \mathbb{F}_2^{(n+p)\times n}$ into b blocks $A_{in}^1, \ldots, A_{in}^b \in \mathbb{F}_2^{n\times n}$, where each ith block is defined by $A_{in}^i = A_{in}[Q_i]$, $\forall i \in \{1, \ldots, b\}$. Analogously define blocks $(A_{out}^1, \ldots, A_{out}^b)$, with $A_{out}^i = A_{out}[Q_i]$, $\forall i \in \{1, \ldots, b\}$.

The main observation of the Block algorithm is the following: assuming only that $A_{in}^1 \in \mathbb{F}_2^{n\times n}$ (resp. A_{out}^1) is full-rank, the rest of the blocks A_{in}^i (resp. A_{out}^i) can also be made full-rank in $\lceil \log(b) \rceil$ depth. We will denote the full rank version of block A_{in}^i by $\hat{A}_{in}^i$, and similarly $\hat{A}_{out}^i$ for A_{out}^i, where $\hat{A}_{in}^1 = A_{in}^1$ and $\hat{A}_{out}^1 = A_{out}^1$. The key idea behind this

observation to add *partial permutations* of one block to another. Specifically, given any $A, B \in \mathbb{F}_2^{n\times n}$ such that A is full rank, there exists a partial permutation P such that $B + PA$ is full rank (Lemma 5.1 in [BBV+21]). Adding a partial permutation of A to B can be implemented by a depth-1 CNOT circuit, since controls are distinct qubits in A and targets are distinct qubits in B by definition of a permutation.

In particular, consider the input parity state A_{in}. Since we start with A^1_{in} as full-rank, the rest of the blocks can be made full-rank through a tree-like spreading procedure like in Figure 3, just on blocks instead of individual qubits and with partial permutations P instead of a single CNOT. Concretely, in the first layer, add the appropriate partial permutation P of $\hat{A}^1_{in}$ to A^2_{in} to obtain $\hat{A}^2_{in}$; in the next layer, add the appropriate P of $\hat{A}^1_{in}$ to A^3_{in} to obtain $\hat{A}^3_{in}$ and the appropriate P of $\hat{A}^2_{in}$ to A^4_{in} to obtain $\hat{A}^4_{in}$; and so on.[9]

It follows that $\lceil \log(b) \rceil$ layers are required to make all blocks full-rank. Since each layer involves qubit-disjoint CNOTs and is hence depth 1, the process requires at most $\lceil \log(b) \rceil$ total depth. The analogous procedure for the output parity state A_{out} yields the same result: we can obtain full rank versions $\hat{A}^i_{out}$ of each block A^i_{out} in $\lceil \log(b) \rceil$ depth. For each i, let G_i be the CNOT circuit of depth at most $\lceil \log(b) \rceil$ such that $G_i A^i_{out} = \hat{A}^i_{out}$.

Now consider the full-rank blocks $\{\hat{A}^i_{in}\}_{i=1}^b$ and $\{\hat{A}^i_{out}\}_{i=1}^b$, each of dimension $n \times n$. For each i, let $C_i = \hat{A}^i_{out} \cdot (\hat{A}^i_{in})^{-1} \in GL_n(\mathbb{F}_2)$. Then C_i is simply a n-qubit linear reversible transformation from $\hat{A}^i_{in}$ to $\hat{A}^i_{out}$ and hence can be implemented as a n-qubit CNOT circuit in depth $d(n)$, using the routine we consider as a blackbox input. The key efficiency of the Block algorithm is that circuits corresponding to each C_i can be applied *in parallel*, so the overall depth required to transform all $\hat{A}^i_{in}$ blocks into $\hat{A}^i_{out}$ blocks is $d(n)$ (rather than e.g. $d(n+p)$). We can then apply the *inverse* circuits $(G^i)^\dagger$ in parallel to map each $\hat{A}^i_{out}$ block to the final desired A^i_{out} block. Putting the pieces together, we have

1. Initial CNOT circuit to map each $A^i_{in} \mapsto \hat{A}^i_{in}$: $\lceil log(b) \rceil$ depth
2. Parallel CNOT circuits C_i to map each $\hat{A}^i_{in} \mapsto \hat{A}^i_{out}$ using input routine: $d(n)$ depth
3. Final CNOT circuit $(G^i)^\dagger$ to map $\hat{A}^i_{out} \mapsto A^i_{out}$: $\lceil log(b) \rceil$ depth.

Summing these depths yields the overall depth in the claim. Putting these subcircuits together (unifying the C_i into a block diagonal transformation, and then adding the initial and final CNOT circuits) yields the desired CNOT circuit C. □

3.2 Minimizing depth

At the other extreme, we now consider what depth reduction can be achieved under the assumption of ample access to ancilla. The following simple result shows that when we have

[9] We actually have a stronger assumption on the structure of A_{in} than in [BBV+21], where it is assumed only that A^1_{in} is full-rank. Indeed, for us, $A^1_{in} = \mathbb{I}_n$ and $A^i_{in} = \mathbf{0}$ for all $i > 1$. This conceptually simplifies the spreading for input blocks. In this case it suffices to let the partial permutation P always be $\mathbb{I}_n$. The spreading process then just involves spreading each of the n data qubit parities stored in A^1_{in} to the $b-1$ other blocks exactly as in Figure 3: qubit j is spread to qubits $\{(i-1)\cdot n + j\}_{i=2}^b$, for all $1 \le j \le n$.

at least $p \cdot n$ ancilla, the parities can be synthesized simultaneously in roughly $\log(p \cdot n)$ depth.

Lemma 3.3. *Given access to $m = p \cdot n$ ancilla, the p parities over n data qubits in* ParallelParities *can be synthesized in $\lceil \log(p+1) \rceil + \lceil \log n \rceil$ depth.*

Proof. The claim follows from a simple two-step procedure, where both steps use the idea of a binary tree CNOT structure for logarithmic depth spreading from Figure 3. To start, we split the $p \cdot n$ ancilla into p new registers of n ancilla each, with the jth register corresponding conceptually to jth of the p parities.

(1) First, spread the data qubit register to each of these ancilla registers, done in parallel for each of the data qubits. This can be done in $\lceil \log(p+1) \rceil$ total depth through the binary tree CNOT structure for spreading, since there are p ancilla registers to which to spread, and each qubit's spreading is disjoint and hence executed completely in parallel across the data qubits.

(2) Now, the p parities can be synthesized in parallel from their corresponding ancilla registers. Each parity, involving at most n data qubits, can be synthesized in $\lceil \log(n) \rceil$ depth again with a CNOT tree structure on its corresponding ancilla register (the opposite of spreading, as discussed in Section 2).

This yields depth $\lceil \log(p+1) \rceil + \lceil \log n \rceil$ overall. □

Remark: In practice, the number of ancilla used can possibly be reduced by only spreading the data qubits involved in each parity to the corresponding register. Each ancilla register's size will be equal to the *weight* $w_i \leq n$ of the corresponding parity $f_i(\mathbf{x})$, rather than n, resulting in $\sum_{i=1}^{p} w_i \leq np$ ancilla qubits used total. The depth is then $\lceil \log p \rceil + \lceil \max_i \log(w_i) \rceil$.

Intuitively, without heavier hammers from quantum computing like measurement-based quantum computing (MBQC),[10] achieving depth logarithmic in n and p seems to be the best we can do. A data qubit in general might need to be spread to p wires (for use in p parities); since we can only double the number of qubits that we can reach in each layer with a 2-qubit gate like a CNOT, this implies logarithmic depth in p. Similar reasoning for accumulating a parity from n wires via CNOTs implies logarithmic depth in n. This reasoning suggests that logarithmic depth (broadly speaking) is a minimal depth result in a standard model of parity synthesis via CNOTs.

4 Our Approach: Controlling the Space-Depth Tradeoff

Sections 3.1 and 3.2 thus provide two existing parity synthesis options at opposite ends of the spectrum in terms of space versus depth usage:

[10] In particular, [BKP09, BW24] together show that the "spreading" operation (i.e. the quantum FanOut gate) can be implemented in constant depth in an MBQC model, which implies that the whole ParallelParities procedure could be done in constant depth with MBQC (e.g. via an analogous procedure to the proof of Lemma 3.3). MBQC-based results are generally much more resource-intensive in terms of ancilla in practice, however.

1. Lemma 3.1 uses only p ancilla and has depth $d_a(n,p) := d(n) + 2\lceil \log(\frac{p}{n}+1)\rceil$, where $d(n)$ upper bounds the depth of implementing an n-qubit linear reversible circuit (e.g. $d(n) = n + O(\log^2(n))$ from MZ).

2. Lemma 3.3 uses np ancilla and reduces depth to $d_b(n,p) := \lceil \log(p+1)\rceil + \lceil \log(n)\rceil$.[11]

However, these existing methods provide no finer-grained control of the space-depth tradeoff to interpolate between these extremes. There is no clear way to use some intermediate number of ancilla between p and np to obtain some commensurately intermediate depth between $d_a(n,p)$ and $d_b(n,p)$. To the best of our knowledge, this gap has not been addressed so far in the literature on quantum compilation and parity synthesis.

Our main contribution addresses this issue through a simple extension of the Block algorithm from [BBV+21]. Given n data qubits and p parities, we provide a framework for ParallelParities that allows the space-depth tradeoff to be controlled by a chosen parameter c between 1 and n. This is presented formally in Theorem 4.1 and the associated algorithm in its proof. Morally, our framework can be thought of as a second layer of parallelization for parity synthesis: while the Block algorithm already uses parallel applications of a linear reversible synthesis routine, we now leverage parallel applications of the Block algorithm itself. We execute this second layer of parallelization to a degree controlled by the choice of c.

Like the Block algorithm, we express our bounds as a function of the depth required to implement an n-qubit linear reversible circuit; our technique does not require any assumptions about the routine used for this, which is treated as a blackbox input (indeed, because we use Block itself as a blackbox in our framework). As previously mentioned, we make certain divisibility assumptions regarding the parameter c in our main presentation and discussion of the c-controlled synthesis here. We describe options for handling the general case without divisibility assumptions in Appendix A.2.

Theorem 4.1 (**Main result: c-controlled synthesis**). *Suppose we have p parities defined over n qubits, and let c be any positive divisor of n such that n divides $c \cdot p$. Let $d(n)$ be an upper bound on the depth required to implement an n-qubit linear reversible transformation. Then* ParallelParities *can be implemented using $m = c \cdot p$ ancilla in depth at most*

$$d(n,p;c) = d\left(\frac{n}{c}\right) + 2\left\lceil \log\left(\frac{cp}{n}+1\right)\right\rceil + \lceil \log(c)\rceil.$$

We can apply Theorem 4.1 with the MZ routine for implementing linear reversible circuits to obtain a concrete bound.

Corollary 4.2. *Suppose we have p parities defined over n qubits, and let c be any positive divisor of n such that n divides $c \cdot p$. Then* ParallelParities *can be implemented using $m = c \cdot p$ ancilla in depth at most*

$$d(n,p;c) = \left(\frac{n}{c} + 2\log^2\left(\frac{n}{c}\right) + 3\log\left(\frac{n}{c}\right)\right) + 2\left\lceil \log\left(\frac{cp}{n}+1\right)\right\rceil + \lceil \log(c)\rceil.$$

[11] Note that indeed $d_b(n,p) < d_a(n,p)$ as $d(n) \gg 3\log(n)$ in practice and asymptotically, due to the lower bound of $d(n) = \Omega(n/\log(n))$ [PMH03, JST+20].

In comparison to Lemma 3.1 which uses p ancilla to obtain a depth dominated by the $d(n)$ term, Theorem 4.1 demonstrates a proportional tradeoff where using c times as many ancilla reduces the dominant depth term to $d(\frac{n}{c})$, albeit with some new overhead of smaller logarithmic terms. When applied with the MZ algorithm (Corollary 3.2 vs. Corollary 4.2), the dominant term is linear, and hence using c times as many ancilla reduces the dominant term by a factor of c. Furthermore, Theorem 4.1 recovers the bounds from Corollary 3.2 and Lemma 3.3 (up to small constant factors) at the boundary choices of $c = 1$ and $c = n$ respectively.[12]

Proof of Theorem 4.1. We start with n data qubits, $m = c \cdot n$ ancilla, and p parities to synthesize, for $1 \leq c \leq n$ such that c divides n and n divides $c \cdot p$. We prove the theorem constructively, by providing a synthesis algorithm that achieves the claimed depth. The crux of the algorithm is simple: first split each parity into c *parity pieces* such that each piece involves no more than $\frac{n}{c}$ qubits, and then run c instances of the Block algorithm from [BBV+21] in parallel over the parity pieces rather than parities themselves.

Concretely, for all $1 \leq i \leq p$ we split parity $f_i(\mathbf{x})$ into c pieces $\{f_i^{(1)}(\mathbf{x}), \ldots, f_i^{(c)}(\mathbf{x})\}$, such that $f_i^{(j)}(\mathbf{x})$ is a parity piece involving only qubits in the set $\mathbf{x}^{(j)} = \{x_{(j-1)\cdot\frac{n}{c}+1}, ..., x_{j\cdot\frac{n}{c}}\}$, for $1 \leq j \leq c$, and $f_i(\mathbf{x}) = \bigoplus_{j=1}^{c} f_i^{(j)}(\mathbf{x})$. Our main goal is then to synthesize the $c \cdot p$ parity pieces $\{f_i^{(j)}(\mathbf{x})\}_{i,j}$ on $c \cdot p$ ancilla. Once we can do this, observe that the pieces $\{f_i^{(j)}(\mathbf{x})\}_{i,j}$ can be combined into the desired parities $\{f_i(\mathbf{x})\}_{1\leq i\leq p}$ in $\lceil \log(c) \rceil$ total depth using an exactly analogous procedure to Lemma 3.3. Each parity can be created from its c pieces using a tree of CNOTs in $\lceil \log(c) \rceil$ depth, and this process can be executed *in parallel* for all parities since the sets $\{f_i^{(j)}(\mathbf{x})\}_{1\leq j\leq c}$ are stored disjointly (on separate ancilla) across i.

Now, the key observation is that the parity pieces $\{f_i^{(j)}(\mathbf{x})\}_{i,j}$ can be easily synthesized through c applications of the Block algorithm (Lemma 3.1) in parallel. Rather than viewing the n data qubits together, we can simply treat them as c separate groups each of $\frac{n}{c}$ qubits, namely $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(c)}$. We similarly divide up our allocation of $c \cdot p$ ancilla into c groups of p ancilla, one for each data qubit group. Indeed, the data qubit group $\mathbf{x}^{(j)}$ is involved in exactly p parity pieces $\{f_i^{(j)}(\mathbf{x})\}_{1\leq i\leq p}$. Thus we have c separate (qubit-disjoint) instances of the ParallelParities problem, where the jth instance ($1 \leq j \leq c$) involves data qubits $\mathbf{x}^{(j)}$, parity pieces $\{f_i^{(j)}(\mathbf{x})\}_{1\leq i\leq p}$, and the jth group of p ancilla. In particular, each of these ParallelParities instances has the following parameters: $\frac{n}{c}$ qubits, p parities, and p ancilla. Then, by applying the Block algorithm (Lemma 3.1), each instance can be implemented with a circuit of depth at most $d\left(\frac{n}{c}\right) + 2\lceil \log(\frac{cp}{n} + 1) \rceil$. Since these circuits are implemented in parallel (over $1 \leq j \leq c$), this is also the total depth of this step.

Thus there are just two components to the procedure: synthesizing the parity pieces in depth $d\left(\frac{n}{c}\right) + 2\lceil \log(\frac{cp}{n} + 1) \rceil$, and then combining them in depth $\lceil \log(c) \rceil$. Summing these depths together yields the claimed bound. □

[12] Specifically, the $c = 1$ case exactly recovers the bound from Corollary 3.2, while the $c = n$ case yields the bound from Lemma 3.3 up to an extra multiplicative factor of 2.

5 Discussion

In this project we formalize and study the ParallelParities problem, where p parities over n data qubits must be synthesized onto ancilla such that they are present simultaneously. This variant of parity network synthesis is motivated by recent approaches to synthesizing Hamiltonian simulation circuits. We review two existing ideas for approaching this problem, one that uses a minimal number of ancilla via [BBV+21]'s block extension algorithm (Lemma 3.1) and one that minimizes depth with a logarithmic depth spreading approach using far more ancilla (Lemma 3.3).

Given these existing approaches at opposite ends of the space vs. depth spectrum, our main contribution is a framework that enables significantly more control over the space-depth tradeoff in implementing ParallelParities (Theorem 4.1). The framework is parametrized by a value c, controlling the degree of parallelization, that can be chosen depending on the constraints and priorities of any particular application. In particular, this allows instance-specific allocation of space and depth costs. The resulting bounds recover the guarantees of Lemma 3.1 and Lemma 3.3 (up to a multiplicative factor of 2). Our framework is remarkably simple: the main idea is simply to run Block in parallel on pieces of the parities, and the proof fits on a page. We believe this theoretical simplicity will help to further augment the practicality of this tool, making for easy implementation in software.

One next step for this work include empirical evaluation of this framework. It would be useful to map out the specific depth obtained in a variety of parity synthesis applications—for instance, to understand the practical cost of the logarithmic overhead, and to understand which values of c seem to balance both space and depth costs best in practice. Another clear direction for future work is integrating this framework back into the broader agenda of measurement-based synthesis of Hamiltonian simulation circuits. Finally, a future direction of theoretical interest is to formally develop corresponding *lower* bounds for depth over varying numbers of ancilla. This would be useful in more thoroughly mapping out the space-depth tradeoff, to understand if any further optimizations are possible.

6 Acknowledgements

The majority of this work was done while I was an intern at IBM Research (IBM Quantum) in the summer of 2023. I thank my two IBM mentors, Dr. Ali Javadi-Abhari and Dr. Simon Martiel, for their wonderful mentorship and the stimulating research conversations we had together, both during the summer and over the following months. I would also like to thank Prof. Noga Alon for taking the time to serve as my PACM reader. Finally, I thank Prof. Paul Seymour, Ms. Victoria Beltra, and the PACM certificate organizers for the opportunity to present this work.

References

[AAM18] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. On the cnot-complexity of cnot-phase circuits. (arXiv:1712.01859), August 2018. arXiv:1712.01859.

[AMM13] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t-depth optimization of clifford+t circuits via matroid partitioning. (arXiv:1303.2042), December 2013. arXiv:1303.2042.

[BBV+21] Timothée Goubault de Brugière, Marc Baboulin, Benoît Valiron, Simon Martiel, and Cyril Allouche. Reducing the depth of linear reversible quantum circuits. *IEEE Transactions on Quantum Engineering*, 2:1–22, 2021.

[BKP09] Dan E. Browne, Elham Kashefi, and Simon Perdrix. Computational depth complexity of measurement-based quantum computation. In *Theory of Quantum Computation, Communication, and Cryptography*, 2009.

[BW24] Elisa Bäumer and Stefan Woerner. Measurement-Based Long-Range Entangling Gates in Constant Depth. 8 2024.

[DLF+16] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63–66, August 2016.

[JST+20] Jiaqing Jiang, Xiaoming Sun, Shang-Hua Teng, Bujiao Wu, Kewen Wu, and Jialin Zhang. *Optimal Space-Depth Trade-Off of CNOT Circuits in Quantum Logic Synthesis*, pages 213–229. 2020.

[MZ22] D. Maslov and B. Zindorf. Depth optimization of cz, cnot, and clifford circuits. *IEEE Transactions on Quantum Engineering*, 3(01):1–8, jan 2022.

[PMH03] K. N. Patel, I. L. Markov, and J. P. Hayes. Efficient synthesis of linear reversible circuits. (arXiv:quant-ph/0302002), February 2003. arXiv:quant-ph/0302002.

[SYJ23] Aaron Szasz, Ed Younis, and Wibe de Jong. Numerical circuit synthesis and compilation for multi-state preparation. (arXiv:2305.01816), September 2023. arXiv:2305.01816.

[VMGDB22] Vivien Vandaele, Simon Martiel, and Timothée Goubault De Brugière. Phase polynomials synthesis algorithms for nisq architectures and beyond. *Quantum Science and Technology*, 7(4):045027, Oct 2022.

A Modified bounds without divisibility assumptions

A.1 For Block algorithm

The case where n does not divide p slightly complicates the Block algorithm. Instead of having only $n \times n$ blocks, there will now have to be one $(n+r) \times n$ block in the parity state matrices, where $r = p \mod n$. The resulting bound, which is the one given in [BBV+21], is weaker but more general relative to Lemma 3.1.

Lemma A.1 ([BBV+21]). *Let $r = p \mod n$, and let $d(n)$ be an upper bound on the depth required to implement an n-qubit linear reversible transformation. Then a CNOT circuit C that transforms A_{in} to A_{out}, i.e. $C \in \mathbb{F}_2^{p \times p}$ such that $A_{out} = CA_{in}$, can be synthesized in depth upper bounded by $d(n+r) + 2\lceil \log(\lfloor \frac{p}{n} \rfloor + 1) \rceil$.*

The proof of this bound is largely the same as the proof of Lemma 3.1. Let $r = p \mod n$ and $b = \lfloor \frac{p}{n} \rfloor + 1$. We now define $Q_i = \{(i-1) \cdot n + 1, \ldots, i \cdot n\}$ for $1 \leq i \leq b-1$ while $Q_b = \{(b-1) \cdot n + 1, \ldots, np\}$, such that $A_{in}^1, \ldots, A_{in}^{b-1} \in \mathbb{F}_2^{n \times n}$ while $A_{in}^b \in \mathbb{F}_2^{(n+r) \times n}$. The blocks $A_{out}^1, \ldots, A_{out}^b$ are defined analogously. The procedure for making each input and output block full-rank remains the same; for the larger blocks A_{in}^b and $A_{out}^b \in \mathbb{F}_2^{(n+r) \times n}$, full-rank simply means making the block rank n, where the first n rows form an invertible matrix.

For $i < b$, we can use the same transformation $C_i = \hat{A}_{out}^i \cdot (\hat{A}_{in}^i)^{-1} \in GL_n(\mathbb{F}_2)$ to map $\hat{A}_{in}^i$ to $\hat{A}_{out}^i$. The key difference comes from the case of $\hat{A}_{in}^b \mapsto \hat{A}_{out}^b$. In this case, one can define an invertible matrix $C_b \in GL_{n+r}(\mathbb{F}_2)$ — in particular, a linear reversible transformation now over $n+r$ qubits — such that $C_b \hat{A}_{in}^i = \hat{A}_{out}^i$.[13] This $(n+r)$-qubit CNOT circuit can be implemented in depth $d(n+r)$, and so the total depth from applying the CNOT circuits C_i, $1 \leq i \leq b$, in parallel is equal to this dominant term of $d(n+r)$. Thus the overall depth achieved is $d(n+r) + 2\lceil \log(\lfloor \frac{p}{n} \rfloor + 1) \rceil$.

A.2 For our controlled framework

As with the Block algorithm, removing divisbility assumptions adds some complexity to our c-controlled synthesis framework. Here we present two straightforward options for handling this additional overhead. The first leads to an increase in depth (while preserving the number of ancilla required), and the second requires an increase in space (while preserving the depth required). Indeed, these options align with our goal of providing control of the space-depth tradeoff. Depending on the constraints of a particular instance in practice, one can pick which additional cost to tolerate in cases where divisibility assumptions are not satisfied.

Concretely, we continue to consider n data qubits, p parities, and c as a positive divisor of n, but now we no longer assume that n divides $c \cdot p$. Corollary A.2 states the increased depth option for handling this general case, while Corollary A.3 states the increased space option. The terms that change relative to Theorem 4.1 are boxed for emphasis. As in Theorem 4.1,

[13] The details are presented in [BBV+21] Section 5.

we present these corollaries as a function of the depth of an input routine for n-qubit linear reversible circuits. Corresponding concrete bounds can be derived easily by plugging in the MZ bounds, as in Corollary 4.2.

Corollary A.2 (**Handling divisibility with depth increase**). *Suppose we have p parities defined over n qubits, and let c be a positive divisor of n. Let $d(n)$ be an upper bound on the depth required to implement an n-qubit linear reversible transformation. Let $r = \left\lceil \left(p \mod \frac{n}{c}\right) / \lfloor p/(\frac{n}{c}) \rfloor \right\rceil$. Then* ParallelParities *can be implemented using $m = c \cdot p$ ancilla in depth at most*

$$d(n, p; c) = d\left(\frac{n}{c} + r\right) + 2\left\lceil \log\left(\frac{cp}{n} + 1\right)\right\rceil + \lceil \log(c) \rceil.$$

In particular, $d(n, p; c) \leq \boxed{d\left(2 \cdot \frac{n}{c}\right)} + 2\left\lceil \log\left(\frac{cp}{n} + 1\right)\right\rceil + \lceil \log(c) \rceil$.

Proof of Corollary A.2. This option follows from applying the same idea used in the general Block algorithm (Lemma A.1) to our c-instance parallel procedure, with a slight optimization. We will still run c instances of the Block algorithm in parallel, split up over the $\frac{n}{c}$-groups of qubits $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(c)}$. Consider the jth instance for some $1 \leq j \leq c$, involving the parity pieces $\{f_i^{(j)}(\mathbf{x})\}_{1 \leq i \leq p}$. After grouping these pieces into complete groups of size $\frac{n}{c}$ to form $\lfloor p/(\frac{n}{c}) \rfloor$ blocks, we will have $r_0 := p \mod \frac{n}{c}$ parity pieces remaining. Rather than appending all of these remainder pieces to one block as [BBV+21] do in the Block algorithm, we can slightly optimize the parallelization by evenly distributing the remainder across all blocks. This increases the number of rows in a given block by at most $r := \lceil r_0 / \lfloor p/(\frac{n}{c}) \rfloor \rceil$, to $\frac{n}{c} + r$. The procedure for making these input and output blocks full-rank remains the same as in the general Block algorithm; roughly, we can simply ignore the remainder rows that get appended for the full-rank spreading. In addition, one can similarly find a linear reversible transformation now over (at most) $\frac{n}{c} + r$ qubits to transform the new input block to the new output block as in the general case for Block. Thus the depth required for this stage of the Block algorithm increases to $d(\frac{n}{c} + r)$, while the rest of the procedure remains the same. This row-distributing can be done for all instances running in parallel, and the final step of combining the pieces remains the same. This yields the claimed depth bound. We remark that in the worst case r can be up to $\frac{n}{c}$, and so $d(\frac{n}{c} + r) \leq d(2 \cdot \frac{n}{c})$, but for many parameter values we expect r to be notably smaller than $\frac{n}{c}$. □

Corollary A.3 (**Handling divisibility with space increase**). *Suppose we have p parities defined over n qubits, and let c be a positive divisor of n. Let $d(n)$ be an upper bound on the depth required to implement an n-qubit linear reversible transformation. Then* ParallelParities *can be implemented using* $\boxed{m = n \cdot \lceil \frac{cp}{n} \rceil}$ *ancilla in depth at most*

$$d(n, p; c) = d\left(\frac{n}{c}\right) + 2\left\lceil \log\left(\frac{cp}{n} + 1\right)\right\rceil + \lceil \log(c) \rceil.$$

Relative to Theorem 4.1, notice that this option can lead to an extra space overhead of up to n additional ancilla. The space bound is not particularly optimized and is very easy to

achieve, by padding the number of parities to a multiple of $\frac{n}{c}$. The main thing to understand is why we need such padding at all. To achieve the same depth as Theorem 4.1, specifically with the leading term of $d\left(\frac{n}{c}\right)$, we need to construct blocks of dimension (at most) $\frac{n}{c} \times \frac{n}{c}$ on which to run the Block algorithm. But now we may have a nonzero remainder $r := (p \bmod \frac{n}{c}) < \frac{n}{c}$ of parity pieces, yet each parity piece can in general involve all $\frac{n}{c}$ qubits from its data qubit group. In particular the number of columns in the remainder parity block in general cannot be reduced from $\frac{n}{c}$, and hence we cannot somehow synthesize a block of dimension less than $\frac{n}{c} \times \frac{n}{c}$ for the remainder parities. The reason we need to synthesize square blocks, again, results from correspondence between linear reversible synthesis routines and *invertible* Boolean matrices. The additional ancilla are thus used to pad the remainder parities to obtain a block with $\frac{n}{c}$ rows.

Proof of Corollary A.3. Simply pad the number of parities to the nearest multiple of $\frac{n}{c}$: let $p' = \frac{n}{c} \cdot \lceil \frac{cp}{n} \rceil \geq p$. It does not matter what the parities added for padding are for the depth bound; to minimize gates used we can simply consider these to be the null parities (i.e. $f(\mathbf{x}) = \vec{0}$). With n qubits and p' parities, Theorem 4.1 can now be applied as the divisibility assumptions are satisfied, and it directly yields the claimed space and depth bounds. □