Unitary entanglement construction in hierarchical networks

Aniruddha Bapat,[1,2] Zachary Eldredge,[1,2] James R. Garrison,[1,2] Abhinav Deshpande,[1,2] Frederic T. Chong,[3] and Alexey V. Gorshkov[1,2]

[1]*Joint Center for Quantum Information and Computer Science, NIST/University of Maryland, College Park, Maryland 20742, USA*
[2]*Joint Quantum Institute, NIST/University of Maryland, College Park, Maryland 20742, USA*
[3]*Department of Computer Science, University of Chicago, Chicago, Illinois 60637, USA*

(Received 23 August 2018; published 26 December 2018)

The construction of large-scale quantum computers will require modular architectures that allow physical resources to be localized in easy-to-manage packages. In this work we examine the impact of different graph structures on the preparation of entangled states. We begin by explaining a formal framework, the hierarchical product, in which modular graphs can be easily constructed. This framework naturally leads us to suggest a class of graphs, which we dub hierarchies. We argue that such graphs have favorable properties for quantum information processing, such as a small diameter and small total edge weight, and use the concept of Pareto efficiency to identify promising quantum graph architectures. We present numerical and analytical results on the speed at which large entangled states can be created on nearest-neighbor grids and hierarchy graphs. We also present a scheme for performing circuit placement—the translation from circuit diagrams to machine qubits—on quantum systems whose connectivity is described by hierarchies.

DOI: 10.1103/PhysRevA.98.062328

I. INTRODUCTION

As quantum computers grow from the small, few-qubit machines currently deployed to the large machines required to realize useful, fault-tolerant computations, it will become increasingly difficult for every physical qubit to be part of a single contiguous piece of hardware. Just as modern classical computers do not rely on a single unit of processing and memory, instead using various components such as CPUs, GPUs, and RAM, we expect that a quantum computer will likewise use specialized modules to perform different functions. At a higher level, computers can be organized into clusters, data centers, and cloud services which allow for a distributed approach to computational tasks, another paradigm quantum computers will no doubt emulate. Already there has been significant interest in how quantum algorithms for elementary operations such as arithmetic perform in distributed-memory situations [1,2] and how to automate the design of quantum computer architectures [3]. In addition, the construction of a fault-tolerant quantum computer naturally suggests a separation of physical qubits into groups corresponding to logical qubits, which makes modularity an attractive framework for building fault-tolerant computers [4]. Modular and scalable computing architectures have been explored for both ion trap [5,6] and superconducting platforms [7–9].

In this paper we use tools from graph theory to discuss benefits and drawbacks of different potential architectures for a modular quantum computer. A graph-theoretic approach allows us to flexibly examine a wide range of possible arrangements quantitatively and allows for convenient numerical simulation using existing software packages designed for network analysis [10]. We especially wish to focus on families of graphs that can scale with the desired number of qubits. In general, we assume that connectivity, i.e., being able to quickly perform operations between nodes, is desirable in an architecture, but that building additional graph edges is in some way costly or difficult, and so will try to minimize the number of needed edges to achieve a highly communicative graph.

We will make use of a previously described graph-theoretic binary operation known as the hierarchical product [11,12]. We will use this iteratively to describe a new family of graphs we dub "hierarchies." We will show that hierarchies perform well by many commonsense graph metrics and argue that they would serve as a plausible and efficient basis for a quantum computing architecture. Furthermore, we will demonstrate that these graphs allow for easily implemented heuristic procedures to assist in the compilation of quantum algorithms.

We will examine the performance of graphs in generating large entangled states such as the multiqubit Greenberger-Horne-Zeilinger (GHZ) state (also known as a cat state). The GHZ state has perfect quantum correlations between different qubits; it thus can be used to perform high-precision metrology [13,14]. In addition, the creation of a GHZ state can be used as part of a state-transfer protocol, which may be useful as part of large quantum computations [15].

An additional property of GHZ state preparation and state transfer which makes them a useful starting point is that, in nearest-neighbor connected systems, performing these tasks using unitary processes from an initial product state is limited by the Lieb-Robinson bound [16,17]. It takes a time proportional to the distance between two points to establish maximal quantum correlation between them. By examining these tasks on a range of different graphs, we hope to understand how the graph structure can affect the limitations on quantum processes caused by locality considerations. Prior work has characterized the difficulty of creating graph states [18], but preparation of such states is not limited by Lieb-Robinson considerations.

Our work in this paper should be contrasted with work on entanglement percolation [19,20]. Entanglement percolation describes the process of using low-quality entanglement between adjacent nodes on a graph to create one unit of long-range, high-quality entanglement (e.g., a Bell pair). The use of entanglement percolation to prepare large cluster states on a lattice was considered in Ref. [21]. The nature of entanglement growth in complex networks was considered in Refs. [22,23], showing that so-called "scale-free" networks are particularly easy to produce large entangled states in. We are interested in the overall capability of different graph structures to perform large computations and in the use of graph eigenvalue methods to understand the spread of quantum information [24]. GHZ state preparation and state transfer are just two possible benchmark tasks, and it is possible that other tasks would result in different evaluations of relative performance between graphs.

Our work should also be considered in the context of classical network theory, where much is known about complicated graph structures [25–27]. It remains to be seen to what degree classical network theory can be easily exported to the quantum domain. Quantum effects such as the no-cloning theorem may limit our ability to distribute information, or conversely we can take advantage of teleportation by distributing quantum bandwidth in anticipation of it actually being needed. As further examples of how quantum and classical networks differ, it has been shown that entanglement swapping may be used to permit quantum networks to reshape themselves into interesting and useful topologies [28]. It has also been shown that, in general, the optimal strategy for entanglement generation in quantum networks can be difficult to calculate because many aspects of classical control theory do not apply [29].

The structure of this paper is as follows. In Sec. II we will introduce a binary operation on graphs known as the hierarchical product, describe how it can be used to produce families of graphs we call hierarchies, and discuss the properties of these hierarchies. In Sec. III we will compare hierarchies to other families of graphs, examining how certain graph-theoretic quantities scale with the total number of included qubits. Readers who are not interested in graph theoretic details may wish to skip much of these first two sections. In Sec. IV we will use analytic and numerical methods to examine how long is required to construct GHZ states spanning our graphs or to transfer states across them, using Lieb-Robinson bounds to connect graph-theoretic quantities to bounds on quantum computing performance. Finally, in Sec. V we will show how the unique structure of hierarchies allows for simple heuristics to map qubits in an algorithm into physical locations in hardware.

II. HIERARCHICAL PRODUCTS OF GRAPHS

A. Background and notation

One of the defining features of modularity in a network is the presence of clusters of nodes that are well connected. Qualitatively, a modular network can be partitioned into such node clusters, or *modules*, that have a sparse interconnectivity. In quantum networking, it is believed that fully connected architectures will suffer greatly decreasing performance or increasing costs as the number of nodes becomes larger, and this motivates the search for alternative network designs. For instance, Ref. [30] estimates that a single module of trapped-ion qubits will likely contain no more than 10 to 100 ions, noting that the speed at which gates are possible becomes slower as the module is expanded. On the network scale, we might imagine a network of nodes over longer distances connected by quantum repeaters [31]. In such a network, establishing direct links between every possible pair of N nodes would require $\Theta(N^2)$ sets of quantum repeaters, a prohibitive cost as N becomes large.

The state of the art in quantum technologies, such as ion traps and superconducting qubits, is the ability to control a small number ($\approx$10–100) of physical qubits using certain fixed sets of one- and two-qubit operations. Instead of increasing the size of these modules, one could instead build a network out of many small modules that are connected at a higher level in a sparse way, perhaps by optical communication links [30].

Our first goal will be to describe modular architectures in the language of graph theory. This will then allow us to quantify and compare their connectivity properties against other network designs, notably the nearest-neighbor grid architecture.

Our detour into graph theory in this paper serves two purposes. First, it will allow us to develop a rigorous way to construct families of graphs which we believe are promising quantum computing architectures. Second, we will later (beginning in Sec. IV) use these graph properties to connect directly to physical bounds on the generation of states with long-range quantum correlations; phrasing the properties of quantum architectures as graphs allows us to make a direct application of the Lieb-Robinson bound to these cases.

An unweighted graph $G = (V, E)$ is conventionally specified by a set of vertices V and a set of edges between the vertices E, where an edge between distinct vertices i and j will be denoted by the pair (i, j). In this paper, we use the terms "vertex" and "node" synonymously. The *order* of a graph is the total number of vertices in the graph $|V|$. It will be useful for the purposes of this paper to work with *weighted* graphs, where we specify a weight $w_{ij} \in \mathbb{R}$ for each pair of vertices $(i, j) \in V \times V$. Two vertices i and j are said to be *disconnected* if $w_{ij} = 0$, and connected by an edge with weight $w_{ij} \neq 0$ otherwise. Thus, unweighted graphs may be thought of as graphs with unit weight on every edge.

Finally, the graphs we consider here will be *simple*, meaning:

(i) The edges have no notion of direction. In other words, $w_{ij} = w_{ji}$ for all $i, j \in V$.

(ii) There are no self-edges, i.e., $w_{ii} = 0$ for all $i \in V$.

(iii) Any two vertices have at most one edge between them.

Henceforth, graphs will be simple and weighted, unless otherwise specified.

The information contained in a graph can be represented as a matrix known as the *adjacency matrix*, whose rows and columns are labeled by the vertices in V and whose entries hold edge weights. Thus, the adjacency matrix is an $n \times n$ matrix where $|V| = n$. The adjacency matrix A_G (or simply

A for shorthand) for a graph G is given by

$$A_{ij} = \begin{cases} 0, & \text{if } i = j, \\ w_{ij}, & \text{if } i \neq j. \end{cases} \tag{1}$$

An important measure of local connectivity is given by the *valency* v_i of a node i, with $v_i = \sum_{j=1}^{n} w_{ij}$. For unweighted graphs, the valency of any node is simply the number of edges incident at that node, otherwise known as the *degree* of the node. We will also define the graph diameter $\delta(G)$ as the maximization of the shortest distance between two nodes on the graph over all pairs of nodes.

Graphs may also be described by the *Laplacian*. The algebraic Laplacian L is given by

$$L_{ij} = \begin{cases} v_i, & \text{if } i = j, \\ -w_{ij}, & \text{if } i \neq j. \end{cases} \tag{2}$$

The algebraic Laplacian is closely related to the adjacency matrix, since we may write $L = \Delta - A$, where $\Delta = \text{diag}(v_1, \ldots, v_n)$ is the diagonal matrix of vertex valencies. The eigenvalues of the algebraic Laplacian give us bounds on various graph properties, as discussed further in Sec. II B 4.

Finally, we remark that the algebraic Laplacian should not be confused with the normalized Laplacian $\mathcal{L} = \Delta^{-1/2} L \Delta^{-1/2}$, which is frequently seen in the network theory literature. The algebraic properties discussed in the next section (such as associativity of the hierarchical product) apply to the adjacency matrix as well as the algebraic Laplacian, but not to the normalized Laplacian.

B. Hierarchical product

Here we will define the hierarchical product and illustrate it with simple examples. For a fuller exposition, see Ref. [11], where the hierarchical product of graphs was introduced. Note that, in some contexts, the hierarchical product is also known as the rooted product [12].

Given a graph G, let $\mathbb{1}_G$ denote the identity matrix on $n = |V|$ vertices. We will denote by D_G an $n \times n$ diagonal matrix with 1 as the first entry and zero everywhere else. Note that there is no natural notion of order to graph vertices, so the choice of "first" vertex must be specified explicitly. Graphs with such a specified first vertex are called *rooted graphs* [32]. We write these matrices as

$$\mathbb{1} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix}, \quad D = \begin{pmatrix} 1 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix}. \tag{3}$$

Definition II.1: Given graphs G and H, the *hierarchical product* $P = G \,\Pi\, H$ is the graph on vertices $V_P = V_G \times V_H$ and edges $E_P \subseteq V_P \times V_P$ specified by the adjacency matrix

$$A_P = A_G \otimes D_H + \mathbb{1}_G \otimes A_H, \tag{4}$$

or, equivalently, by the algebraic Laplacian

$$L_P = L_G \otimes D_H + \mathbb{1}_G \otimes L_H. \tag{5}$$

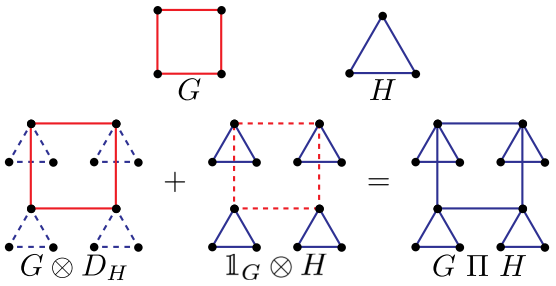

FIG. 1. A simple example of the hierarchical product $G \,\Pi\, H$ between the cycle graphs $G = C_4$ and $H = C_3$. The first term in Eq. (4), $A_G \otimes D_H$, creates one copy of G on the vertex set formed by the first vertices of each H copy, while the second term $\mathbb{1}_G \otimes A_H$ creates the four copies of H.

We will often use the shorthand $A_P = A_G \,\Pi\, A_H$ and $L_P = L_G \,\Pi\, L_H$.

If G and H are graphs, then $G \,\Pi\, H$ may be thought of as one copy of G with $|G|$ copies of H, each attached to a different vertex of G (see Fig. 1). Thus, $G \,\Pi\, H$ is a graph which has $|G|$ *modules* of $|H|$ nodes each. The modules' internal connectivity is described by H, and the modules are connected to one another in a manner described by G. The hierarchical product formalism therefore naturally produces modular graphs. Its main advantage comes from the convenience of working with the algebra at the level of adjacency matrices and Laplacians, which in turn makes the computation of important properties of such graphs straightforward.

We now present some properties of the hierarchical product which make it an attractive formalism for practical applications in quantum networking.

1. *Structural properties*

At the level of adjacency matrices, the hierarchical product is *associative*. Let A, B, C be three adjacency matrices. Then,

$$(A \,\Pi\, B) \,\Pi\, C = A \,\Pi\, (B \,\Pi\, C). \tag{6}$$

For a proof, we refer the reader to Ref. [11].

Associativity implies that a product of multiple graphs does not depend on the order of evaluation. Therefore, we can unambiguously take the hierarchical product over many graphs to produce a graph of the form $G_k \,\Pi\, G_{k-1} \,\Pi \cdots \Pi\, G_1$. We will refer to such graphs as *hierarchies*, and the ith graph in the product G_i as the ith level of the hierarchy, enumerated from the bottom level upwards (symbolically, from right to left). In particular, if all G_i are equal to some graph G, then we write

$$G^{\Pi k} := \underbrace{G \,\Pi \cdots G \,\Pi}_{k-1 \text{ times}} G. \tag{7}$$

and refer to $G^{\Pi k}$ as a depth-k (or k-level) hierarchy.

Note that the hierarchical product *does not* satisfy many properties which are commonly assumed for operations on matrices. In particular,

(1) Bilinearity: $(A_1 + A_2) \,\Pi\, B = A_1 \otimes D_B + A_2 \otimes D_B + \mathbb{1}_{(A_1+A_2)} \otimes B \neq A_1 \,\Pi\, B + A_2 \,\Pi\, B$. Similarly, $A \,\Pi\, (B_1 + B_2) \neq A \,\Pi\, B_1 + A \,\Pi\, B_2$.

(2) Scalar multiplication: For any scalar α, $(\alpha A) \Pi B = \alpha A \otimes D_B + \mathbb{1}_A \otimes B \neq \alpha(A \Pi B) \neq A \Pi (\alpha B)$. Note however that scalar multiplication is distributive in the following way: $\alpha(A \Pi B) = (\alpha A) \Pi (\alpha B)$.

Hierarchical graphs are also instances of hyperbolic graphs. The Gromov hyperbolicity [33], which measures curvature and is small for a graph with large negative curvature, is only a constant for hierarchical graphs. Since the hyperbolicity in general is at most half the graph diameter, whereas in this case it is independent of the diameter, it is termed *constantly hyperbolic* in the parlance of Ref. [34]. Hyperbolic graphs are seen in several real-world complex networks [35,36], most notably the internet [37,38]. Hyperbolic lattices have also been realized recently in superconducting circuits [39].

Finally, hierarchies have low tree, clique, and rank widths, which are each measures of the decomposibility of a graph [40]. These structural properties imply efficient algorithms for optimization problems expressible in monadic second-order (MSO) logic—a class which, for arbitrary graphs, includes several NP-hard problems. This feature could potentially be used to solve circuit layout and optimization problems on modular architectures without resorting to heuristics. We refer the reader to Ref. [41] for details on these structural results.

2. Scalability

So far we have discussed hierarchies in which the edges in different levels of the hierarchy are equally weighted. However, one useful generalization would be to allow the weight of edges at each layer of the hierarchy to vary. The meaning of this weight could vary depending on the context. In some cases, weights can be used to quantify the costs of an edge (*cost weight*). In others, we may wish to use weighted edges to quantify the power or performance of a network, interpreting edge weights as the strength of terms in a Hamiltonian or, inversely, the time required to communicate between nodes (*time weight*).

In this work we prefer to remain agnostic to the meaning of the weights as much as is possible. When we calculate graph properties in Sec. III, we will do so without reference to the meaning of the weights. In general, we will allow a graph to assign multiple kinds of weights to its edges, and each type of weight might scale differently. For now, we define a generalization of the hierarchical product which will allow us to construct hierarchies that incorporate different weights at different levels of the hierarchy.

Definition II.2: Given graphs G and H, and $\alpha \in \mathbb{R}_+$, the *α-weighted hierarchical product* $P = G \Pi_\alpha H$ is a graph on vertices $V_P = V_G \times V_H$ and edges $E_P \subseteq V_P \times V_P$ specified by the adjacency matrix

$$A_P = \alpha A_G \otimes D_H + \mathbb{1}_G \otimes A_H, \tag{8}$$

or, equivalently, by the algebraic Laplacian

$$L_P = \alpha L_G \otimes D_H + \mathbb{1}_G \otimes L_H. \tag{9}$$

We will often use the shorthand $A_P = A_G \Pi_\alpha A_H$, and $L_P = L_G \Pi_\alpha L_H$.

As before, we may construct a k-level, *weighted* hierarchy out of k base graphs $G_1, \ldots, G_k$, and k weights $\alpha_i, \ldots, \alpha_k \equiv \vec{\alpha}$, so that the edges of the ith level graph G_i are weighted by the ith component of $\vec{\alpha}$, α_i. The adjacency matrix of such a hierarchy may be written as

$$A^{\Pi_{\vec{\alpha}} k} := \sum_{i=1}^{k} \alpha_i \mathbb{1}_{[i+1..k]} \otimes A_i \otimes D_{[1..i-1]}, \tag{10}$$

where the subscripts $[a\,..\,b]$ on $\mathbb{1}$ and D are shorthand for the Kronecker product of matrices over all descending indices in the integer interval $[a\,..\,b]$. For instance, $D_{[1..i-1]} := D_{G_{i-1}} \otimes D_{G_{i-2}} \otimes \cdots \otimes D_{G_1}$.

Defined as above, a weighted hierarchy $G^{\Pi_{\vec{\alpha}} k}$ is uniquely and efficiently specified by a real vector of weights $\vec{\alpha} \in \mathbb{R}_+^k$ and an ordered tuple of graphs $(G_1, \ldots, G_k)$. It will be the case that our analyses are unaffected by an overall scaling of the weight vector, so that one may identify $\vec{\alpha} \equiv c\vec{\alpha}$ for any real scalar c. As convention, we will always normalize by setting $\alpha_1 = 1$, which corresponds to assigning a unit-weight multiplicative factor to the lowest-level graphs in the hierarchy.

We can construct the adjacency matrix of the graph $G^{\Pi_{\vec{\alpha}} k}$ by repeated application of the twofold product (Def. II.2) in some well-defined way, analogous to Eq. (7). However, unlike before, the weighted product is nonassociative, so we must first define an order of operations for manifold weighted products. Unless otherwise specified, we will always evaluate a manifold product from *right to left*, which corresponds to building the hierarchies from the bottom up, and is required in order to ensure that this definition matches Eq. (10). For example, in the threefold product $A_3 \Pi_{\alpha_3} A_2 \Pi_{\alpha_2} (\alpha_1 A_1)$, we will first evaluate the product $A_2 \Pi_{\alpha_2} (\alpha_1 A_1)$, and then take the product of A_3, weighted by α_3, with the resulting graph. The final result is

$$\alpha_3 A_3 \otimes D_2 \otimes D_1 + \alpha_2 \mathbb{1}_3 \otimes A_2 \otimes D_1 + \alpha_1 \mathbb{1}_3 \otimes \mathbb{1}_2 \otimes A_1. \tag{11}$$

In fact, a k-fold product, when evaluated this way, matches the right-hand side of Eq. (10). Therefore, the k-level weighted hierarchy can also be written unambiguously as

$$A^{\Pi_{\vec{\alpha}} k} = A_k \Pi_{\alpha_k} A_{k-1} \Pi_{\alpha_{k-1}} \cdots \Pi_{\alpha_2} (\alpha_1 A_1). \tag{12}$$

Henceforth, the weight α_1, which scales the lowest-level adjacency matrix A_1, will be dropped due to our normalization choice of $\alpha_1 = 1$.

An important class of hierarchy graphs is one where the level weights follow a geometric progression of weights, i.e., $\alpha_i = \alpha^{i-1}$. We will denote such hierarchies by $G^{\Pi_\alpha k}$, where the scalar subscript α will be understood to mean the mutual weighting between successive hierarchies. For $\alpha > 1$, this leads to a "fat tree" structure, while for $\alpha < 1$, we instead get a "skinny tree" for which the edge weights decrease between consecutive levels from the leaves to the root. These constructions are illustrated in Fig. 2, and mentioned because fat trees are known to be a commonly used architecture in classical networks [42].

Allowing a clear separation of the modular system into hierarchical levels, each of which can be assigned unique edge weight, enables straightforward discussion of computation that occurs both within and between modules in a unified

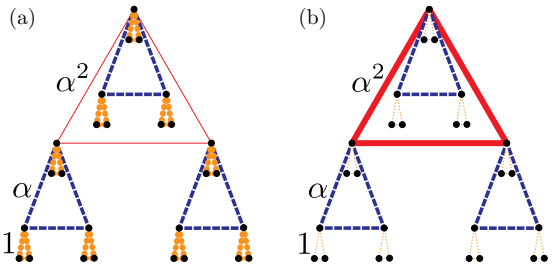

FIG. 2. An illustration of the use of the hierarchical product to produce (a) skinny and (b) fat trees. In each case, the hierarchy $K_3^{\Pi_\alpha 3}$ is drawn, with the thickness of edges illustrating the weight of those edges. Depending on whether $\alpha < 1$ or $\alpha > 1$, this can lead to either lower-weighted high-level edges as in (a) or higher-weighted ones as in (b). Note that, for ease of visualization, here we break the usual convention of taking the lowest-level edges as unit weight.

framework. When two nodes interact, we can assign this a cost that depends on the edges between them.

3. Node addressal

A hierarchy on N nodes gives a natural labeling of the nodes. Suppose the hierarchy H contains k levels and each level is described by a graph G with $|G| = n$ nodes, where $n^k = N$. Label the vertices of G by indices $j = 0, 1, \ldots, n-1$. Then, the adjacency matrix $\mathbb{1}_G \otimes G$ (which corresponds to n disjoint copies of G) has vertices which may be labeled as (jk), where $j, k = 0, 1, \ldots, n-1$. The first label identifies which copy of G the node occurs in, while the second identifies where in G it appears. The same vertex labeling can then be used for the two-level hierarchy $G \,\Pi\, G$. In this manner, the k-level hierarchy has n^k vertices with labels of the form $(b_1 b_2 \cdots b_k)$, where $b_i \in \{0, 1, \ldots, n-1\}$ for all i. This is essentially a k-digit, base-n representation of numbers from 0 to $N = n^k - 1$, as illustrated in Fig. 3.

This node addressal scheme allows for each node to be uniquely identified in a way that simultaneously describes its connectivity to other nodes and allows for easy counting of how many nodes lie in either the entire graph or in particular subgraphs. This addressal scheme will be important for describing a variant of hierarchies in Sec. II B 5 and for implementing the graphs in software, e.g., as used to generate the numerical results in Sec. IV C.

4. Spectral properties

One of the tools frequently used in analyzing large networks is the spectral decomposition of the Laplacian. The behavior of the largest eigenvalue, the first eigenvalue gap, and the distribution of eigenvalues as a function of the network parameters are some of the diagnostics that can provide key information about dynamical processes on the network, and can also be used as points of comparison between competing network topologies [43].

The smallest eigenvalue of a Laplacian is always $\lambda_1 = 0$, which corresponds to the uniform eigenvector $\vec{e}_1 = (1, 1, \ldots, 1)$. In ascending order, the eigenvalues of L may be denoted by $0 = \lambda_1 \leqslant \lambda_2 \leqslant \cdots \leqslant \lambda_N$. We now state some

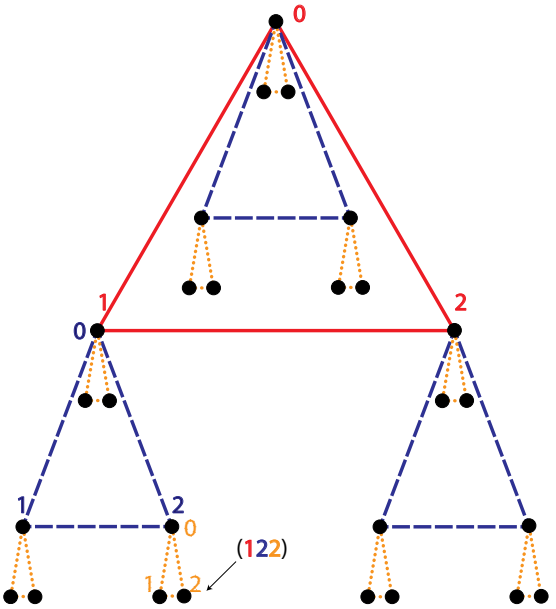

FIG. 3. Addressing nodes in the hierarchy, layer by layer. Shown is a three-level hierarchy with the triangle graph K_3 as its base. Each vertex is represented as a three-digit number in base 3. The first digit points to a node at the top level (red solid triangle), the second to a location in the second level (blue dashed triangle), and finally, the last digit (yellow dotted triangle) specifies the node location completely.

graph properties that can be related to the spectrum of L [43,44].

The second eigenvalue λ_2 is known as the *algebraic connectivity* of the graph and is closely related to the expansion and connectivity properties of the graph. Broadly, the larger the value of λ_2, the better the connectivity of the network. To illustrate this point, consider the graph diameter $\delta(H)$ which can be bounded using λ_2 as follows:

$$\frac{4}{N\lambda_2} \leqslant \delta(H) \leqslant 2\left\lceil \frac{\Delta + \lambda_2}{4\lambda_2} \ln(N-1) \right\rceil, \tag{13}$$

where Δ is the maximum degree of H. It can be seen that a larger value for λ_2 will lead to a smaller graph diameter. We also have the following asymptotic bound on the mean distance between nodes $\bar{\rho}(H)$:

$$\frac{2}{(N-1)\lambda_2(H)} + \frac{1}{2} \lesssim \bar{\rho}(H) \lesssim \left\lceil \frac{\Delta + \lambda_2}{4\lambda_2} \ln(N-1) \right\rceil. \tag{14}$$

Another important diagnostic of a network is given by the *Cheeger constant* $h(H)$ [45], also called the isoperimetric number or the graph conductance. This graph invariant is a measure of how difficult the graph is to disconnect by cutting edges. For a connected graph, this number is always positive. As benchmark values, the complete graph K_N has Cheeger constant $N/2$ while a cycle graph C_N has Cheeger constant $4/N$. The relationship between λ_2 and $h(H)$ can be seen through the following bounds:

$$\frac{\lambda_2}{2} \leqslant h(H) \leqslant \sqrt{\lambda_2(2\Delta - \lambda_2)}. \tag{15}$$

Many other graph properties may be derived from the Laplacian spectrum as well (see, e.g., Refs. [43,44]).

For a large network, finding the eigenvalues can be numerically expensive. However, hierarchies have a special structure which can be exploited for the evaluation of graph spectra. Here we show (in Theorem II.1) that if the spectra of the base graphs L_i are known, then one can derive the spectrum of the k-level hierarchy efficiently using a recursive procedure. We first present two lemmas. The first lemma generalizes Theorem 3.10 from Ref. [11], which states that the characteristic polynomial $\phi_P(x)$ $(= \det[x\mathbb{1} - P])$ of an unweighted hierarchical product of adjacency matrices A, B is given by

$$\phi_P(x) = \phi_{B'}(x)^{n_A}\phi_A\left(\frac{\phi_B(x)}{\phi_{B'}(x)}\right), \tag{16}$$

where A' (B') is the matrix A (B) with the first row and first column removed, and $n_A = |G_A|$ is the order of the graph A. In fact, Eq. (16) applies to Laplacians as well as adjacency matrices. The lemma below further generalizes this statement to a weighted product of Laplacians.

Lemma II.1: Let K and L be two graph Laplacians with characteristic polynomials given by $\phi_K(x)$ and $\phi_L(x)$, respectively. Then, the characteristic polynomial $\phi_\Pi(x)$ of the hierarchical product $K\ \Pi_\alpha\ L$ is given by

$$\phi_\Pi(x) = [\alpha\phi_{L'}(x)]^{n_K}\phi_K\left(\frac{1}{\alpha}\frac{\phi_L(x)}{\phi_{L'}(x)}\right), \tag{17}$$

where $n_k = \dim\{K\}$, and L' is defined similar to A' and B' above.

Proof. Denote the spectra of K and L by $\{\kappa_j\}$ and $\{\lambda_j\}$, respectively. Recall that the α-weighted hierarchical product may be written as

$$K\ \Pi_\alpha\ L = \alpha K \otimes D_L + \mathbb{1}_K \otimes L. \tag{18}$$

If U_K is a unitary that diagonalizes K, we conjugate the above equation with the unitary $U_K \otimes \mathbb{1}_L$, and look at the resulting block matrix. Each block corresponds to an eigenvalue of K, and thus the jth block is given by $\alpha\kappa_j D_L + L$. The full spectrum may then be expressed as a disjoint union of the block spectra,

$$\mathrm{spec}(K\ \Pi_\alpha\ L) = \bigsqcup_{j=1}^{|K|}\mathrm{spec}(\alpha\kappa_j D_L + L). \tag{19}$$

Now we apply Eq. (16) to $K\ \Pi_\alpha\ L \equiv (\alpha K)\ \Pi\ L$ and use the fact that $\phi_{\alpha K}(x) = \det[x\mathbb{1} - \alpha K] = \alpha^{n_K}\det[\frac{x}{\alpha}\mathbb{1} - K] \equiv \alpha^{n_K}\phi_K(\frac{x}{\alpha})$. This yields Eq. (17), as desired.

Now we show that if the eigenvalues of K and the polynomials ϕ_L and $\phi_{L'}$ are known, then there is a straightforward procedure to compute the eigenvalues of $K\ \Pi_\alpha\ L$. ■

Lemma II.2: Let K and L be graph Laplacians, as before. Each eigenvalue of the product characteristic polynomial ϕ_Π can be found as a solution of the equation

$$\alpha\kappa_i = \frac{\phi_L(x)}{\phi_{L'}(x)} \tag{20}$$

for some K-eigenvalue κ_i.

Proof. Any eigenvalue of the product graph must be a zero of the left-hand side of Eq. (17) and, by equality, a zero of the right-hand side. Now the degree of polynomial ϕ_K is n_K, which implies that the term of degree n_K must be nonzero. Thus, in the product $\phi_{L'}(x)^{n_K}\phi_K(\frac{1}{\alpha}\frac{\phi_L(x)}{\phi_{L'}(x)})$, there must be a term which is *indivisible* by the polynomial $\phi_{L'}(x)$. Therefore, the zero of the right-hand side cannot be a root of the polynomial $\phi_{L'}$.

We are seeking values of x such that the polynomial $\phi_K(\frac{1}{\alpha}\frac{\phi_L(x)}{\phi_{L'}(x)})$ evaluates to zero. In other words, we are looking for x such that the term $\frac{1}{\alpha}\frac{\phi_L(x)}{\phi_{L'}(x)}$ is a root of ϕ_K. Therefore, we solve Eq. (20) for x, for all roots κ_i of K. ■

If the forms of ϕ_L and $\phi_{L'}$ are known (and if each have sufficiently low degree), then computing the roots of ϕ_Π becomes tractable, even if K is a large matrix. This suggests a recursive procedure for computing the spectrum of a k-level hierarchy, by writing it as a product of the $(k-1)$-level hierarchy with the kth base graph. We now frame this as our main result of this section:

Theorem II.1: Suppose we have a k-level hierarchy $L^{\Pi_{\vec{\alpha}}k}$ described by base graph Laplacians $L_1, L_2, \ldots, L_k$ and weights $\vec{\alpha} = (1, \alpha_2, \ldots, \alpha_k)$ as follows:

$$L^{\Pi_{\vec{\alpha}}k} = L_k\ \Pi_{\alpha_k}\ L_{k-1}\ \Pi_{\alpha_{k-1}} \cdots \Pi_{\alpha_3}\ L_2\ \Pi_{\alpha_2}\ L_1. \tag{21}$$

Define a new set of weights $\vec{\beta} = (1, \beta_2, \ldots, \beta_k)$ with $\beta_i = \alpha_i/\alpha_{i-1}$, and a new set of Laplacians $M_k, M_{k-1}, \ldots, M_1$ recursively as

$$M_k = L_k,$$
$$M_i = M_{i+1}\ \Pi_{\beta_{i+1}}\ L_i.$$

Then, the following hold:

(1) $M_1 = L^{\Pi_{\vec{\alpha}}k}$.

(2) Any eigenvalue of M_i (for $i < k$) may be found as a solution to the equation

$$\beta_{i+1}\mu^{(i+1)} = \frac{\phi_{L_i}(x)}{\phi_{L_i'}(x)} \tag{22}$$

for some $\mu^{(i+1)} \in \mathrm{spec}\{M_{i+1}\}$.

Proof. First, we prove statement 1. It can be seen that

$$\begin{aligned} M_{k-1} &= M_k\ \Pi_{\beta_k}\ L_{k-1} = L_k\ \Pi_{\beta_k}\ L_{k-1} \\ &= \frac{1}{\alpha_{k-1}}(\alpha_k L_k \otimes D_{k-1} + \alpha_{k-1}\mathbb{1}_k \otimes L_{k-1}), \end{aligned} \tag{23}$$

$$\begin{aligned} M_{k-2} &= M_{k-1}\ \Pi_{\beta_{k-1}}\ L_{k-2} \\ &= \frac{1}{\alpha_{k-2}}(\alpha_k L_k \otimes D_{k-1} \otimes D_{k-2} + \alpha_{k-1}\mathbb{1}_k \otimes L_{k-1} \\ &\quad \otimes D_{k-2} + \alpha_{k-2}\mathbb{1}_{k-1} \otimes \mathbb{1}_{k-2} \otimes L_{k-2}), \end{aligned} \tag{24}$$

and so on, until we have an $\vec{\alpha}$-weighted sum over all k of the base graphs (with an overall denominator of $\alpha_1 = 1$), which is precisely $L^{\Pi_{\vec{\alpha}}k}$.

The proof of statement 2 follows as a direct consequence of Lemma II.2, with $K = M_{i+1}$, $L = L_i$, and $\alpha = \beta_{i+1}$. ■

Theorem II.1 provides an algorithm to compute the spectrum of $L^{\Pi_{\vec{\alpha}}k}$, namely:

(1) Compute the relative weight vector $\vec{\beta}$ from $\vec{\alpha}$.

(2) Start with $i = k$, where the spectrum of $M_k = L_k$ is known. Decrease i by one.

(3) Compute the spectrum of M_i from the known spectrum of M_{i+1} and Eq. (22). Decrease i by one.

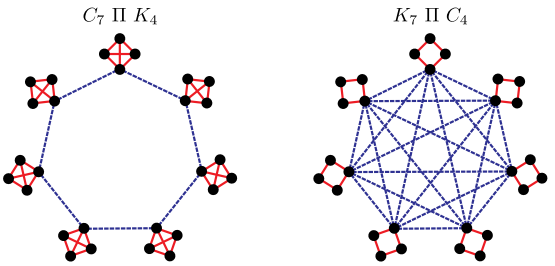

FIG. 4. Two topologies with the same number of nodes (28) and edges (49). While the diameters for the two graphs are the same, are they equally well connected? A comparison of the Cheeger constants (see Table I) suggests that the left graph is less interconnected. This is consistent with the spectral gap, which is smaller for the left graph, indicating poorer connectivity.

(4) Perform step 3 repeatedly, halting at $i = 0$. Return the spectrum of $M_1 = L^{\Pi_{\vec{\alpha}} k}$.

Therefore, given a large hierarchy, one can efficiently compute the Laplacian eigenvalues and use them to find bounds on important graph properties. This is a scalable technique for obtaining figures of merit efficiently for hierarchies. Later, in Sec. III we will present analytic results for some of these figures of merit for simple hierarchies, but the results of the current section can be used even in more complicated cases, such as hierarchies that do not use the same G at every layer or that have heterogeneous scaling parameters.

Due to the structural richness and heterogeneity of graphs, it is not always easy to decide whether one graph is, for instance, more connected than another graph. One aspect of connectivity is how close the nodes are to one another, which is captured by quantities like the diameter and mean distance. In Fig. 4 we compare two graphs, $C_7 \Pi K_4$ and $K_7 \Pi C_4$, which have an identical number of nodes (28) and edges (49). The two graphs also have identical diameters (5 each), but the mean distance for the left graph is smaller (see Table I). Under these measures, the left graph appears better connected.

Better connectivity also corresponds to having fewer bottlenecks in the graph, which corresponds to a larger Cheeger constant. In Fig. 4, the graph on the right has a larger Cheeger constant, as one would expect given that it has complete connectivity between the seven modules. Note that this metric of connectivity need not agree with the mean distance, as seen in this example.

Similarly, a parameter-by-parameter comparison of the two hierarchy graphs $C_{13} \Pi K_5$ and $K_{13} \Pi C_5$ (Table I) reveals

TABLE I. Comparison of topologies by connectivity measure. In each case, the graphs being compared have an identical number of nodes and edges. The better value for each comparison is underlined.

Graph invariant	$C_7 \Pi K_4$	vs	$K_7 \Pi C_4$	$C_{13} \Pi K_5$	vs	$K_{13} \Pi C_5$
Number of edges	49		49	143		143
Number of nodes	28		28	65		65
Diameter	<u>5</u>		<u>5</u>	8		<u>5</u>
Mean distance	<u>2.68</u>		2.71	4.77		<u>3.23</u>
Cheeger constant	0.17		<u>1.0</u>	0.07		<u>1.4</u>
Spectral gap λ_2	0.16		<u>0.46</u>	0.04		<u>0.34</u>

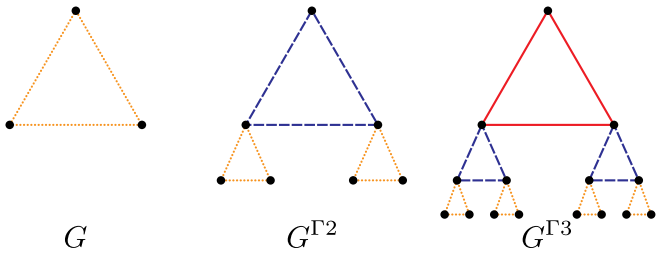

FIG. 5. A demonstration of how our hierarchical product can be truncated to avoid requiring many interconnections at one node. As the hierarchy grows, the graph is duplicated and then attached to a subset of nodes in a larger version of the base graph G.

that, while both graphs are two-level hierarchies with the same number of nodes and edges, $K_{13} \Pi C_5$ has the smaller diameter, smaller mean distance, larger Cheeger constant, and a larger spectral gap, all of which indicate better connectivity. While structural comparisons for the above examples can be carried out simply by inspection or a quick calculation of graph quantities, general hierarchies may be far too complex to compare this way. In practice, when choosing a modular topology with the best connectivity, one might hope for a single, balanced measure of connectivity that relates to aspects such as node distance and bottleneckedness and is easy to compute. The spectral gap λ_2 meets these requirements. It is asymptotically related to the other invariants discussed here via upper and lower bounds in Eqs. (13)–(15). Furthermore, λ_2 can be efficiently computed using the recursive procedure described earlier in this section.

5. *Truncated hierarchical product*

In some scenarios, there may be physical or technological limitations on the total number of interconnections allowed at a single node of a quantum computer. In our framework, this manifests as a restriction on the maximum degree of a node. We believe that hierarchical structures can still prove useful in this context, but (as we will see in Sec. III) the hierarchy we have described thus far has a maximum degree which grows linearly with the number of levels of the hierarchy.

We now introduce an architecture which maintains the hierarchical properties but also has a bounded maximum node degree (i.e., maximum node degree that does not go to infinity as the number of levels goes to infinity). To model such an architecture, we modify the hierarchical product $G_1 \Pi G_2$. Whereas previously, $|G_1|$ copies of G_2 were connected according to G_1, we now bring together $|G_1| - 1$ copies, which we connect according to G_1, and add the root node of G_1 without an associated subhierarchy (see Fig. 5). When extended to a many-level hierarchy, this means that every node will be connected to, at most, two levels, and so its degree will not grow as the hierarchy grows. We will denote this *truncated* hierarchical product by $G_1 \Gamma G_2$, and its weighted version as $G_1 \Gamma_\alpha G_2$. It can be written algebraically in terms of adjacency matrices by adopting a more general definition of the hierarchical product.

Definition II.3: Given rooted graphs G and H, the *weighted truncated hierarchical product* $P = G \Gamma_\alpha H$ is a graph on vertices $V_P = V_G \times V_H$ and edges $E_P \subseteq V_P \times V_P$

specified by the adjacency matrix

$$A_P = \alpha A_G \otimes D_H + P_G \otimes A_H, \tag{25}$$

or, equivalently, the algebraic Laplacian

$$L_P = \alpha L_G \otimes D_H + P_G \otimes L_H. \tag{26}$$

Here P_G is a projector onto all nodes in G except the root node. At the level of adjacency matrices, we may also write $A_P = A_G \, \Gamma_\alpha \, A_H$. An unweighted version $G \, \Gamma \, H$ can be obtained by setting $\alpha = 1$.

An illustration of this architecture can be found in Fig. 5. From this definition, we naturally derive both unweighted and weighted truncated hierarchies $G^{\Gamma k}$ and $G^{\Gamma_\alpha k}$. We note that a generalization of this definition to allow an arbitrary projector (rather than one that only excludes the root node) is possible, but we do not consider such a case in this paper.

The addressing scheme outlined in Sec. II B 3 can also be used for truncated hierarchies. However, since many nodes do not sit atop subhierarchies in this case, not all node addresses are valid. We will assume that the node in the ith level which connects to the level above it has a zero in the ith digit of its address. In a truncated hierarchy, each node whose address contains a zero (representing the "root" of a hierarchy) must have only zeros in all following positions, as it does not contain any further subhierarchies. The base-n addressal scheme can thus be used to specify which nodes are present in a truncated hierarchy.

Note that the truncated hierarchical product adds nodes more slowly than (although with the same scaling as) the hierarchical product structure specified at the beginning of Sec. II B. When we perform graph comparisons in Sec. III, we will consider all cost functions and optimizations in terms of the total number of nodes so that the two architectures can be compared fairly.

III. GRAPH COMPARISONS

Having developed the machinery to construct hierarchies, we will now evaluate them against other potential architectures. Any evaluation is impossible to do in an absolute sense, since what properties are desirable in a graph and how serious the cost of improving them is will depend on both the application as well as the physical system under consideration. In general, we assume that the most desirable quality of a graph is some measure of connectivity or the ease with which the graph can transport information between nodes. Note that it is always possible to translate between quantum circuit architectures with some overhead. A detailed atlas summarizing these overheads can be found in Ref. [46].

We will look at the scenario of state transfer, which is an important subroutine that may need to be carried out if an algorithm requires gates to be performed between two qubits that are not directly connected. We consider the worst-case state transfer time on a given graph, which allows us to evaluate graphs without reference to any particular quantum algorithm. If we are interested in the time taken for state transfer in the graph, an appropriate metric can be the diameter of the graph $\delta(H)$ under the assumption that information transfer takes unit time along any edge in the graph. The diameter then captures the maximum distance, and hence the maximum time required for information to travel between any two nodes in the system.

For graphs produced by the weighted hierarchical product, we will also consider a diameter which takes into account edge weight. This "weighted diameter" $\delta_w(H)$ can be found by considering all pairs of nodes j, k and identifying the two whose least-weighted connecting path has the highest sum weight of edges. If we consider a path between two nodes j and k to be a set of nodes $P = \{j, v_1, v_2, \ldots, v_n, k\}$ with a weight $W(P)$ given by the sum $w_{j,v_1} + w_{v_1,v_2} + \cdots + w_{v_n,k}$, then the weighted diameter can be written as

$$\delta_w(H) = \max_{j,k} \min_P W(P). \tag{27}$$

One way to grasp why the weighted diameter is a useful quantity is to consider the time weights of edges, where the weight signifies the time required to perform a gate between two connected qubits. In this case, the weighted diameter is the maximum time it will take us to perform a chain of two-qubit gates that connects two different qubits (for instance, using SWAP operations to bring the two qubits to adjacent positions and then performing the final desired operation).

However, optimizing *only* with respect to connectivity yields a trivial result, because a fully connected graph is obviously most capable of communicating information between any two points. Therefore, we will consider a number of different possible "costs" associated with physical implementations of graphs. One potential input to the cost function is the maximum degree of a graph $\Delta(H)$. As discussed in the previous section, we want to avoid needing to connect too many different communication channels to a single node. Another is total edge weight $w(H)$—if it costs time, energy, money, coherence, or effort to produce communication between two nodes, we should try to use as few communication channels as possible.

We now walk through the calculations for several important graph quantities for several graphs: an all-to-all connected graph, a cycle graph, a star graph, a square grid, a hierarchy graph with scaling parameter α, and a truncated version of that same hierarchy graph. We calculate how quantities scale with the total number of nodes N. For ease of calculation, we assume that N nodes fit in the architecture of the current graph; for instance, we assume $N = \ell^d$ for some integer ℓ for a d-dimensional square graph. All results of this section are compiled in Table II, and examples of the graphs for small N are illustrated in Fig. 6.

A. Graph calculations

1. Complete graph K_N

Since all nodes in a complete graph [Fig. 6(a)] have edges between them, the diameter is simply 1. This comes at the cost of very high maximum degree $N - 1$, as every node is connected to all $N - 1$ other nodes. The total weight of every edge is the same, and there are $N(N-1)/2$ edges because every pair of nodes has a corresponding edge. Therefore, the total edge weight scales as $\Theta(N^2)$.

TABLE II. Summary of scalings of important graph properties with total node number N. All entries describe only the scaling of the leading coefficient with d, n, and N.

Graph H	Diameter δ	Weighted diameter δ_w	Maximum degree Δ	Total edge weight $w(H)$
K_N	const.	const.	N	N^2
S_N	const.	const.	N	N
C_N	N	N	const.	N
Square grid, d-dim	$dN^{1/d}$	$dN^{1/d}$	d	dN
$K_n^{\Pi_\alpha k}, \alpha \neq n$	$\log_n N$	$\max\left(\frac{2}{1-\alpha}, N^{\log_n \alpha}\right)$	$n \log_n N$	$nN^{\max(1,\log_n \alpha)}$
$K_n^{\Pi_\alpha k}, \alpha = n$	$\log_n N$	$\max\left(\frac{2}{1-\alpha}, N^{\log_n \alpha}\right)$	$n \log_n N$	$nN \log_n N$
$K_{n+1}^{\Gamma_\alpha k}, \alpha \neq n$	$\log_n N$	$\max\left(\frac{2}{1-\alpha}, N^{\log_n \alpha}\right)$	n	$nN^{\max(1,\log_n \alpha)}$
$K_{n+1}^{\Gamma_\alpha k}, \alpha = n$	$\log_n N$	$\max\left(\frac{2}{1-\alpha}, N^{\log_n \alpha}\right)$	n	$nN \log_n N$

2. Cycle graph C_N

In a cycle graph [Fig. 6(b)], the diameter is $\lfloor N/2 \rfloor$, the distance to the opposite side of the circle. The maximum degree is only 2, and the total weight of the edges is likewise only N. This graph is thus able to reduce the cost factors associated with the complete graph, but at the cost of a much higher asymptotic diameter.

3. Star graph S_N

The star graph is the graph which has a single central node connected to all others [Fig. 6(c)]. Like the complete graph, it also has a constant diameter, although this diameter is two rather than one. The maximum degree of the star graph is $N - 1$, the same as the complete graph. However, the star graph improves over the complete graph, as it has a lower total edge weight of $N - 1$ rather than $\binom{N}{2}$. Thus, we have improved the cost asymptotically without affecting the overall scaling of the diameter of the graph.

The example of S_N raises a complication which we do not attempt to quantify in this paper. In a realistic distributed quantum computer, we expect that a significant amount of operations need to be performed at the same time and need to be scheduled on the graph. But in the star graph, all operations between nodes must pass through the single central hub. This is likely to lead to a scheduling bottleneck when performing general quantum algorithms. While we do not attempt to treat scheduling of such algorithms on the network in this paper, in future work we hope to consider these complications, which will at times make the star graph unsuitable for real-world use. An experimental comparison of the star graph and the complete graph in existing five-qubit quantum computers can be found in Ref. [47]. In those experiments, the requirement that all information be shuttled through a central node for the S_N connectivity made high-fidelity execution of quantum algorithms more difficult.

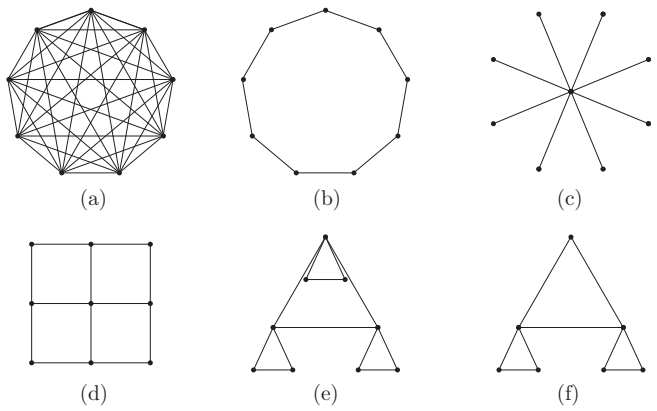

FIG. 6. Illustration of the graph structures considered in this section, each with nine nodes except (f). (a) The complete graph K_9. (b) The cycle graph C_9. (c) The star graph S_9. (d) The nearest-neighbor grid in two dimensions. (e) The hierarchical product $K_3^{\Pi 2}$. (f) The truncated hierarchical product of Sec. II B 5, $K_3^{\Gamma 2}$.

4. Square grid graph

We consider now a square grid (i.e., a hypercubic lattice) in d dimensions [Fig. 6(d)]. Here the diameter is $d(N^{1/d} - 1)$, since this is the distance from the point in one corner labeled $(1, 1, 1, \dots)$ to the opposite corner at $(N^{1/d}, N^{1/d}, \dots)$ (note that diagonal moves are not allowed). The maximum degree depends on the dimension, as each interior node is connected to $2d$ other nodes. The total edge weight can be found by considering that each node on the interior of the graph corresponds with exactly d edges, and it is these edges that dominate as $N \to \infty$. Therefore, the total edge weight scales as $\Theta(dN)$.

5. Hierarchy graph $G^{\Pi_{\tilde{a}} k}$

As the hierarchy graph [Fig. 6(e)] is built recursively, it is easiest to calculate its properties using recursion relations. We consider a graph that has k levels to it, so that given a base graph G and $n = |G|$, then the overall graph has n^k nodes.

First, we calculate the unweighted diameter of a k-level hierarchy, which we denote by $\delta(G^{\Pi_{\tilde{a}} k})$. Since all subhierarchies are rooted at their first vertex, we will need to keep track of the *eccentricity* of the root node, which we denote by $\varepsilon(F)$ for any subhierarchy F. The eccentricity of any graph node is defined as the maximum distance from that node to any other node in the graph F. Here we fix $\varepsilon(F)$ to be the root eccentricity for the graph in question.

Now we write recursion relations for two quantities, the unweighted diameter $\delta(G^{\Pi_{\tilde{a}} i})$ of an i-level hierarchy for some

intermediate i, and the eccentricity $\varepsilon(G^{\Pi_{\vec{\alpha}} i})$ of the top-level root node of the current i-level hierarchy.

Consider a diametric path in an i-level hierarchy. This path must ascend and descend the entire hierarchy. That is, using the notation of Sec. II B 3, two maximally separated qubits have addresses that are different in their first digit. Such a path can always be partitioned into three disjoint pieces, the terminal two of which each lie in some $(i-1)$-level subhierarchy, while the middle piece lies in the current top (i.e., ith) level. These three pieces must be independently maximal, since the path is diametric. The middle piece maximizes to the diameter of the top-level graph, which is simply $\delta(G)$. The two sublevel pieces each maximize to the root eccentricity of the $(i-1)$th level subhierarchy, which is precisely the quantity $\varepsilon(G^{\Pi_{\vec{\alpha}}(i-1)})$. Therefore, our first recursion reads

$$\delta(G^{\Pi_{\vec{\alpha}} i}) = 2\varepsilon(G^{\Pi_{\vec{\alpha}}(i-1)}) + \delta(G). \tag{28}$$

The ith level root eccentricity may be found by a similar argument. Partition the most eccentric path (starting at the top level root node) into two pieces, one which lies at the top level, and the other which lies exclusively in the lower levels. Maximizing both pieces, one gets

$$\varepsilon(G^{\Pi_{\vec{\alpha}} i}) = \varepsilon(G^{\Pi_{\vec{\alpha}}(i-1)}) + \varepsilon(G). \tag{29}$$

Solving the second relation, we get $\varepsilon(G^{\Pi_{\vec{\alpha}} i}) = i\varepsilon(G)$. By substitution, the first recursion has the solution

$$\delta(G^{\Pi_{\vec{\alpha}} k}) = 2(k-1)\varepsilon(G) + \delta(G). \tag{30}$$

Since the total number of levels is given by $k = \log_n N$, and the graph diameter is no greater than twice the eccentricity of any node, we conclude that the diameter scales as $\Theta[\varepsilon(G)\log_n N]$ for a general graph G. If we specifically examine the case when G is a complete graph of order n, $\delta(G) = 1$, and $\varepsilon(G) = 1$, and the exact expression is $\delta(G^{\Pi_{\vec{\alpha}} k}) = 2\log_n(N) - 1$.

Next we calculate the maximum degree. Again, we proceed by recursion. Iterating the hierarchical product to some level i can be viewed as attaching a copy of the graph $G^{\Pi_{\vec{\alpha}}(i-1)}$ to every point in the graph G. Therefore, the degree of every root node in the $(i-1)$ level subhierarchies increases by the degree of the corresponding node in graph G. The maximal increase achievable thus is the maximum degree $\Delta(G)$ of graph G. Since the root node for an i-level subhierarchy has i distinct copies of G attached to it, its degree is given by $i\deg(g_1)$, where g_1 is the root node of G. Then the i-level maximum degree can be expressed as

$$\Delta(G^{\Pi_{\vec{\alpha}} i}) = \max\{(i-1)\deg(g_1) + \Delta(G), \Delta(G^{\Pi_{\vec{\alpha}}(i-1)})\} \tag{31}$$

$$\cdots = \max_{0\leqslant j\leqslant i-1} \{j\deg(g_1) + \Delta(G)\} \tag{32}$$

$$= (i-1)\deg(g_1) + \Delta(G), \tag{33}$$

where the second step was obtained by recursion. For a general G, this gives the maximum degree scaling as $\Delta(G^{\Pi_{\vec{\alpha}} k}) = \Theta(\log_n N)$. For $K_n^{\Pi_{\vec{\alpha}} k}$, the root degree and the maximum degree of the base graph K_n are both $n-1$, so $\Delta(K_n^{\Pi_{\vec{\alpha}} k}) = (n-1)\log_n N$.

Now we consider the total edge weight of the hierarchy. We compute this by a recursion relation, first by duplicating the existing edge weight at $i-1$ levels by n (the number of smaller hierarchies we must bring together) and then adding new edges. If the edges at level i have weight α_i, we can write this as

$$w(G^{\Pi_{\vec{\alpha}} i}) = nw(G^{\Pi_{\vec{\alpha}}(i-1)}) + \alpha_i w(G). \tag{34}$$

By counting the number of subhierarchies with different weights, we find the following form for the total edge weight of the weighted hierarchy:

$$w(G^{\Pi_{\vec{\alpha}} k}) = w(G)\sum_{i=1}^{k} \alpha_i |G|^{k-i}. \tag{35}$$

This can be verified by checking that it satisfies the recursion relation Eq. (34). If we now specialize to the case where $G = K_n$ and $\alpha_i = \alpha^{i-1}$, we find

$$w\left(K_n^{\Pi_\alpha k}\right) = \frac{n(n-1)}{2}\sum_{i=1}^{k} \alpha^{i-1} n^{k-i}. \tag{36}$$

This behavior can be broken into three regimes. For $\alpha = n$, every term in the sum is identical, and the overall scaling is $\Theta(nN\log_n N)$. Otherwise, we can perform the geometric sum to obtain

$$w\left(K_n^{\Pi_\alpha k}\right) = \frac{n(n-1)}{2}\frac{n^k - \alpha^k}{n-\alpha}. \tag{37}$$

Here the scaling will depend on the relative size of n and α. For $n > \alpha$, the first term in the numerator dominates, and $w(K_n^{\Pi_\alpha k}) = \Theta(nN)$. Otherwise, we can write $\alpha^k = N^{\log_n \alpha}$ and find $w(K_n^{\Pi_\alpha k}) = \Theta(nN^{\log_n \alpha})$.

Finally, we calculate the weighted diameter of a k-level hierarchy $\delta_w(G^{\Pi_{\vec{\alpha}} k})$, just as for the unweighted diameter, by solving recursion relations for the quantities $\delta_w(G^{\Pi_{\vec{\alpha}} i})$ and $\varepsilon_w(G^{\Pi_{\vec{\alpha}} i})$, which are, respectively, the weighted diameter and weighted root eccentricity for an i-level weighted hierarchy. Here note that the top level (at any intermediate stage i) is weighted by α_i. Therefore, the recursion for the weighted diameter is modified to

$$\delta_w(G^{\Pi_{\vec{\alpha}} i}) = 2\varepsilon_w(G^{\Pi_{\vec{\alpha}}(i-1)}) + \alpha_i \delta_w(G). \tag{38}$$

Similarly, the recursion for the weighted eccentricity becomes

$$\varepsilon_w(G^{\Pi_{\vec{\alpha}} i}) = \varepsilon_w(G^{\Pi_{\vec{\alpha}}(i-1)}) + \alpha_i \varepsilon_w(G), \tag{39}$$

which has the solution $\varepsilon_w(G^{\Pi_{\vec{\alpha}} i}) = \varepsilon_w(G)\sum_{j=1}^{i} \alpha_j$. Finally, we have

$$\delta_w(G^{\Pi_{\vec{\alpha}} k}) = 2\varepsilon_w(G)\sum_{j=1}^{k-1} \alpha_j + \delta_w(G)\alpha_k. \tag{40}$$

For $G = K_n$ and $\alpha_i = \alpha^{i-1}$, this becomes

$$\delta_w(K_n^{\Pi_\alpha k}) = 2\sum_{i=1}^{k-1} \alpha^{i-1} + \alpha^{k-1} \tag{41}$$

$$= \frac{\alpha^k + \alpha^{k-1} - 2}{\alpha - 1}. \tag{42}$$

Therefore, the scaling of the weighted diameter with N has two regimes, depending on α. For $\alpha < 1$ the geometric sum converges as $i \to \infty$ to $\frac{2}{1-\alpha}$. This means that for $\alpha < 1$, a constant time suffices to traverse the entire hierarchy no matter

how large it is. For $\alpha = 1$ the weighted diameter is equal to the (unweighted) diameter, which we have already computed. For $\alpha > 1$, δ_w scales as $\alpha^{k-1} = N^{\log_n \alpha}/\alpha \sim N^{\log_n \alpha}$. Note that the last scaling only applies if α does not scale with n. Since $n > 1$ and $\alpha > 1$, this exponent $\log_n \alpha$ is always positive. Therefore, the total edge weight is asymptotically always either constant (for $\alpha < 1$) or growing (for $\alpha \geqslant 1$), as expected.

6. *Truncated hierarchy $G^{\Gamma_{\bar{a}} k}$*

Finally, we look at how the results above are modified if we use the truncated hierarchical product discussed in Sec. II B 5 [Fig. 6(f)]. Although many of the calculations in terms of the number of levels k are similar to those for the nontruncated hierarchy, it is no longer the case that $k = \log_n N$. In order to compare graphs fairly, we will need to recalculate the order of $G^{\Gamma_{\bar{a}} k}$ so that results in this section can be written in terms of the total number of nodes N.

Under the node addressal scheme of Sec. II B 3, the nodes of a truncated hierarchy are in one-to-one correspondence with base-n strings of length k that only have trailing zeros. As before, a 0 label points to a root node, but since root nodes do not bear subhierarchies due to truncation, all subsequent labels are forced to be 0. In other words, we only label nodes using strings of the form $(l_1 l_2 \cdots l_i 00 \cdots 0)$ for some $i \leqslant k$, and $l_j \neq 0$ for all $j \leqslant i$. The number of such strings with i nonzero labels followed by $(k - i)$ zero labels is $(n-1)^i$. Therefore, the total number of nodes is

$$N = \sum_{i=0}^{k} (n-1)^i. \tag{43}$$

Since $N = \Theta[(n-1)^k]$, many quantities of a truncated hierarchy with a base graph of order $n+1$ have the same scaling with the number of nodes N as those for a nontruncated hierarchy with a base graph of order n.

In terms of the number of levels k, the maximum diameter will be proportional to k, just as it was in Sec. III A 5. It follows that the diameter scales with the total number of nodes as $\delta = \Theta(\log_{n-1} N)$ for a truncated hierarchy.

On the other hand, truncation offers a large improvement in the maximum degree of the hierarchy. As discussed in Sec. II B 5, the maximum degree of the truncated hierarchy is $\Delta(G^{\Gamma_{\bar{a}} k}) = 2\Delta(G)$, which is constant in N.

The edge weight recursion relation is simply $n-1$ copies of the current graph and then new, additional edges:

$$w(G^{\Gamma_{\bar{a}} i}) = (n-1) w(G^{\Gamma_{\bar{a}} (i-1)}) + \alpha_i w(G). \tag{44}$$

This is identical to the recursion relation for the standard hierarchy, Eq. (34), except that there are now only $n-1$ copies, and also, for a given number of qubits N, the number of levels k may be different by constant factors and terms. Thus, the only modification to the recursion relation is to replace n with $n-1$, and the solution of the relation is otherwise identical. This means that none of the asymptotic scaling with k is affected, and the scaling with N is only affected by changing the total number of levels required to construct a graph of N nodes.

The recursion relation for weighted diameter is similar to Eq. (38). Due to truncation, one needs to make a careful comparison of paths that do or do not terminate at the root node of the top level, but in any case the weighted diameter's scaling with k is the same as the nontruncated weighted diameter's scaling. The weighted diameter scaling with N can thus be found from Eq. (42), using the appropriate value of k for truncated hierarchies with N nodes.

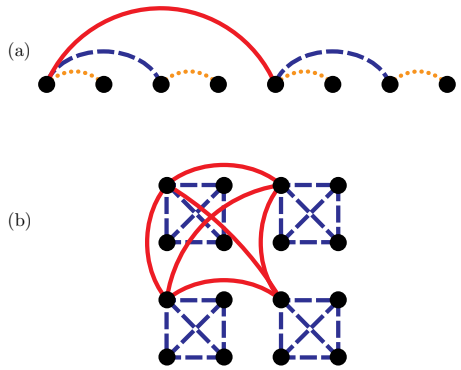

FIG. 7. An illustration of the embedding of a hierarchy on a (a) one- or (b) two-dimensional lattice of qubits. In both cases, the length of an edge doubles at every level of the hierarchy, but the scaling in total edge length used changes from $\Theta(N \log_2 N)$ to $\Theta(N)$ when going from one to two dimensions. In $d = 3$, a similar hierarchy with doubling length scales connects modules of eight qubits.

B. Choosing among graphs

1. *Graph embeddings*

The long list of comparisons summarized in Table II can make it difficult to see exactly when different graphs are preferable. To make our calculations more concrete, we would like to compare concrete scenarios for the connection of qubits arranged on a grid in d dimensions. Specifically, in each dimension ($d = 1$, 2, and 3), we examine a hierarchy that is embedded into the grid, comparing its properties to the same grid but with nearest-neighbor connections. We consider building modules where each small module is a complete graph of size n, laid out in cubes on the grid so that the side length of the cube is $n^{1/d}$. The $d = 1$ and $d = 2$ cases with $n = 2^d$ are illustrated in Fig. 7.

As shown, the length of an edge must increase by a factor of $n^{1/d}$ (2 in Fig. 7) at every level of the hierarchy in order to make these hierarchies possible. Therefore, to determine the total length of wire used, we can use a cost weight with $\alpha = n^{1/d}$. Keeping factors of N only, Table II shows that for $d = 1$, we expect a total cost weight $\Theta(N \log_n N)$, while for the higher-dimensional cases we expect a total cost weight $\Theta(N)$ [48]. For the d-dimensional grid, this total cost weight is always $\Theta(N)$.

Now, to consider the performance of the two graphs, we must fix a separate scaling factor for the time weight β. There are several options which might be reasonable for different physical applications. If $\beta = 1$, i.e., all links act identically in terms of time required to traverse them, then the weighted diameter of the hierarchy is simply $\Theta(\log_n N)$. Another option would be to take $\beta = \alpha$, i.e., to assume that links take as long to move through as they are long. In this case, we find that the hierarchy's weighted diameter scales as $\Theta(N^{1/d})$,

meaning that the hierarchy and nearest-neighbor graphs match in performance.

We may also want to allow hierarchies to make use of the "fat tree" concept to produce a better-performing graph [42]. Suppose that we allow ourselves to "spend more" on higher-level links, causing their cost weight to increase with a factor α, but improving their performance so that the time weight scales with the factor $\beta = 1/\alpha$. In this case, the question is what range of α allows for the hierarchy to perform better than the nearest-neighbor grid (lower time-weighted diameter) for less cost (lower total edge cost weight)? (Note that this cost weight includes any contribution from "lengthening" wires at higher levels of the hierarchy.)

To answer the first, we compare the two asymptotic diameter scalings $N^{\max(0,\log_n 1/\alpha)}$ and $N^{1/d}$. This suggests that if $\alpha \geqslant n^{-1/d}$, the hierarchy will allow for faster traversal than the nearest-neighbor grid. However, we wish to avoid causing the hierarchy to have a total cost weight that scales worse than $\Omega(N)$, which requires $\log_n \alpha < 1$. We find that a winning hierarchy can be constructed if α lies in the range $\alpha \in [n^{-1/d}, n)$. The optimal α is as large as possible but less than n; at that point an additional logarithmic factor is introduced to the total cost weight scaling.

In these cases, we have not allowed the nearest-neighbor grid to modify the weight (either kind) of its links. This is because any modification in its cost or time weight enters simply as a constant factor; if the individual links have weight c instead of 1, the overall weighted diameter is just $cN^{1/d}$ while the total cost weight is just cN. Of course, one can apply different constants to each figure of merit, or apply c to one and $1/c$ to the other. In order to make the nearest-neighbor grid match the performance of the hierarchy, the unit-length time weight would have to be $N^{\log_n(\alpha)-1/d}$ while the unit-length cost weight must not scale with N.

2. *Pareto efficiency*

Our calculation of various graph parameters suggests that the hierarchy architecture offers significant advantages over others. One way to make this comparison more exact is to appeal to the economics concept of Pareto efficiency, which is used to designate an acceptable set of choices in multiparameter optimization [49]. A choice is Pareto efficient if switching to a different choice will cause at least one parameter to become worse. Suppose we eliminate all constants to focus only on the scaling with N for three parameters: weighted diameter, maximum degree, and total edge weight. By removing these constants, we assume that the small multiplicative factors they provide will not influence decision making. For simplicity, we will assume that both cost and time weights scale with the same factor α.

For comparison one could ask: what minimum number of edges is required for a graph on N nodes to have maximum degree Δ and diameter δ? Reference [50] answers this optimization question partially, and constructs what are known as *porcupine graphs* which achieve the optimum, illustrated in Fig. 8. We observe here that qualitatively, porcupines are modular, since they may be described by attaching trees to the nodes of a complete graph. In particular, the graph $K_{\sqrt{N}} \,\Pi\, S_{\sqrt{N}}$ is a porcupine graph that achieves a diameter

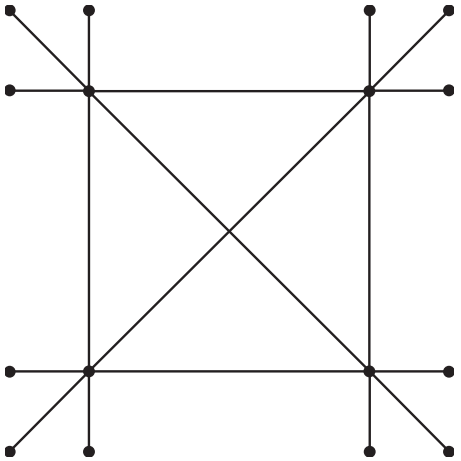

FIG. 8. An example of a porcupine graph as defined in Ref. [50], in this case $K_4 \,\Pi\, S_4$.

$\delta = 3$ and a maximum degree of $\Delta = 2(\sqrt{N} - 1)$ with the minimal number of edges.

We summarize the scalings of these graphs in Table III. Assume that $n^{1/d} \geqslant \alpha \geqslant 1$. In this case, we can find the Pareto-efficient solutions by noting which options can be eliminated. We see that K_N is strictly worse than S_N and can be eliminated; S_N is then dominated by the porcupine. C_N is dominated by the square grid, which has identical scaling of total weight and degree but lower diameter. The square grid, in turn, is dominated by the hierarchy due to the assumptions we have made on α. This means that the two Pareto-efficient choices in this case are the truncated hierarchy and the porcupine graph. If we chose any option besides these two, we could improve the scaling with respect to N without any trade-off by switching to one of them. While this framework does not offer a decision rule to choose between the porcupine and $K_n^{\Gamma_\alpha k}$, the latter is clearly preferable if our aim is to create a modular quantum system that does not rely on a few centralized nodes. We stress that this optimization procedure is only intended to evaluate the quantities and graphs introduced, and the Pareto-

TABLE III. An illustration of the scaling with N of three key parameters to be used in Pareto optimization. Here δ_w is the weighted diameter, Δ is the maximum degree, and w is the total edge weight of the graph. A star ($\star$) has been placed next to the two graphs we find to be Pareto efficient. We have also included the $\alpha = 1$ (unweighted) hierarchy in the final row, as it has a different scaling for the weighted diameter. Our Pareto efficiency judgment is made assuming $n^{1/d} \geqslant \alpha \geqslant 1$.

Graph	δ_w	Δ	w
K_N	const.	N	N^2
S_N	const.	N	N
C_N	N	const.	N
Square grid	$N^{1/d}$	const.	N
$\star\ K_{\sqrt{N}} \,\Pi\, S_{\sqrt{N}}$	const.	$\sqrt{N}$	N
$\star\ K_{n+1}^{\Gamma_\alpha k}$ $\alpha \neq 1$	$N^{\log_n \alpha}$	const.	N
$\star\ K_{n+1}^{\Gamma_\alpha k}$ $\alpha = 1$	$\log_n N$	const.	N

efficient choices will change if other figures of merit or other graphs are included in the optimization.

3. Optimality of diameter for hierarchical graphs

The use of $K_n^{\Gamma_\alpha k}$ may be further motivated via the degree-diameter problem [51] (for a survey, see Ref. [52]). Given a graph with a maximum allowed degree Δ on each node and diameter no greater than δ, the degree-diameter problem asks for the maximum number of nodes $N(\Delta, \delta)$ that such a network could hold. This problem is practically well motivated in the design of networks, and may be answered for special classes of graphs. The Moore bound, which is a bound for general graphs, states that the number of nodes N is at most $\frac{\Delta(\Delta-1)^\delta-2}{\Delta-2}$. This means that for a constant maximum degree $\Delta \gtrsim 3$, the diameter satisfies $\delta = \Omega(\log N)$, meaning that hierarchical graphs have an optimal diameter up to a constant factor. Tighter bounds on the number of nodes may be shown, for instance, when the *tree width* of the graph is bounded.

Reference [53] shows that graphs with small tree widths t and an odd diameter δ satisfy

$$N(\Delta, \delta; t) \sim t(\Delta - 1)^{\frac{\delta-1}{2}}. \tag{45}$$

As discussed towards the end of Sec. II B 1, hierarchies have low tree widths. In particular, the tree width of the truncated hierarchy $K_n^{\Gamma_\alpha k}$ is at most $n-1$. Next, the diameter of the truncated hierarchy $K_n^{\Gamma_\alpha k}$ is $\delta(k) = 2k-1$ (which is odd), and the maximum degree is $\Delta(k) = 2(n-1)$. Comparing the number of nodes in this hierarchy $N(k)$ to the node capacity $N[\Delta(k), \delta(k); n-1]$ as in Eq. (45), we get

$$\frac{N(k)}{N[\Delta(k), \delta(k); n-1]} \gtrsim \frac{n^k}{(n-1)(2n-3)^{k-1}}. \tag{46}$$

Keeping the total number of nodes N fixed, consider two limits: one, a shallow hierarchy in which the number of levels k is $O(1)$, and two, a deep hierarchy, in which the size n of the base graph is $O(1)$ [i.e., $k = O(\log N)$]. We see that when the hierarchy is shallow, the right side of Eq. (46) is $\Theta(1)$, which indicates optimality. For a deep hierarchy, the above ratio scales as $2^{-\log_n N} = N^{\frac{-1}{\log(n)}}$, which is polynomially suboptimal. However, when $n = 3$, the ratio in Eq. (46) is again $\Theta(1)$, and the truncated hierarchy $K_3^{\Gamma_\alpha k}$ is degree-diameter optimal in this case.

IV. ENTANGLED STATE CONSTRUCTION

A. Setup

Although some of the graph properties calculated in the previous section give a heuristic sense for the capabilities of the hierarchical graph versus the nearest neighbor or all-to-all graphs, we would like to examine their performance directly in terms of a quantum information processing task. The task we have chosen as a benchmark is the creation of a many-qubit GHZ state. Since this entangled state is difficult to create across long distances when using nearest-neighbor interactions, we hope that it can serve as a useful yet basic benchmark for processing quantum information with unitary evolution [15]. As shown in Ref. [15], preparation of a GHZ state also provides a means of transferring a state across the graph. Thus, the results of this section also bound state transfer time. However, in this work, unlike Ref. [15], we focus on the use of discrete unitary operations (gates) rather than Hamiltonian interactions. This means that we cannot take advantage of the many-body interference which provided a speed-up in Ref. [15].

Using GHZ state creation as a benchmark for potential quantum architectures allows us to use physical limitations (represented by the Lieb-Robinson bound) to place computational limits on information processing. The GHZ state is directly useful on its own [13–15], but even in systems which do not directly produce the GHZ state, it is likely that quantum operations will require the creation of long-range correlations between distant sites. For example, the same physical bounds which govern the creation of the GHZ state also restrict the speed at which topological order can be produced [16]. We focus on the GHZ state as an easy-to-analyze example for the problem of creating these nonlocal correlations, but we stress that our results generalize to any state which possesses nonlocal correlations of the kind whose creation is limited by the Lieb-Robinson bound.

We adopt a framework in which every vertex of the graph represents one logical qubit, while an edge of the graph represents the ability to perform a two-qubit gate between nodes. For the purposes of this work, we assume that we can ignore single-qubit operations, instead focusing on the cost imposed by the required two-qubit gates between nodes.

B. Analytical results for deterministic entanglement generation

In order to create the GHZ state, we assume that we begin with all qubits in the state $|0\rangle$ except for one qubit that we place in the initial state $|+\rangle$. By performing controlled-NOT operations between this qubit and its neighbors, a GHZ state of those qubits is created. The state can be expanded by continuing to use further CNOT operations to expand the "bubble" of nodes contained in the GHZ state until it eventually spans the entire graph. For state transfer, we instead assume the initial state $|\psi\rangle$ to be transferred sits on one qubit, which is then transferred through the graph using SWAP operations until it reaches its destination.

We first consider a graph which has been assigned time weights, so that a gate between two linked edges can be performed deterministically in a time given by the weight of the edge between them. We assume that one node can act as the control qubit for several CNOT operations at once. Therefore, according to our protocol above, the time t_{GHZ} required to construct the GHZ state is found by identifying the qubit that will take the longest to reach from the initial qubit by hopping on the graph. A similar argument holds for the state transfer time.

This implies that a GHZ state can be created, or a state transferred, in time that scales like the (time-)weighted eccentricity of the node we choose as the initial $|0\rangle + |1\rangle$ state. However, if we take the further step in our analysis of maximizing over weighted eccentricities (identifying the worst-case starting node), then the time will simply be the weighted diameter of the graph as calculated in the previous section. Note that the difference between the best-case

weighted eccentricity (the weighted graph radius) and the worst-case weighted eccentricity (the weighted graph diameter) over all nodes is at most a factor of 2—if we look at the midpoint of the path that realizes the graph diameter, its distance to the endpoints of the path is bounded by the radius—so from the perspective of how this time scales asymptotically with N, the two are interchangeable.

C. Numerical results for probabilistic entanglement generation

As shown in the previous subsection, in a deterministic setting of entanglement generation where a gate between two nodes of our graph H can be performed in fixed time, the time required to create a GHZ state is equal to the weighted diameter $\delta_w(H)$. However, in many situations in long-distance quantum information processing, probabilistic or heralded methods might be used instead. We might suppose that, in a small time step, the network succeeds in performing a desired two-qubit gate with probability p (and that we know whether the gate succeeded or not). Upon failure, one can try performing the gate again in the next time step without having to rebuild the state from the beginning. In this setting, we expect that the scaling will likely be similar to the deterministic case but more difficult to calculate exactly. Fortunately, it is easy to re-interpret the meanings of the edge weights to account for this.

The main complication arising from the inclusion of unitaries that do not get completed in a fixed amount of time is that multiple paths between two nodes can all contribute to the total probability that entanglement has been produced, making it a harder problem to solve exactly. However, we can turn to numerical simulation to get an idea of the behavior. In the following, we define a new edge weight called the probability weight p_{ij} which is the probability of success of edge (i, j) in one time step.

The algorithm for simulating the creation of a GHZ state is as follows:

(i) At each time step t, identify the subgraph F of nodes that have already joined the GHZ state.

(ii) For each edge between a GHZ node $i \in F$ and a non-GHZ node $j \notin F$, identify the probability edge weight p_{ij}. With probability p_{ij}, allow node j to join the GHZ state in the current time step t.

(iii) Once all edges have been tested, repeat the procedure for the next time step on the new, possibly larger, set of GHZ nodes.

A single number p_0 is chosen as the base probability, so that the probability weights on the lowest level are p_0, and edges on the ith level of the hierarchy succeed with probability $p_0\alpha^{i-1}$. Note that we must fix $\alpha < 1$. As a first step toward evaluating the performance of a graph, we estimate its time weights as $w_{ij} = 1/p_{ij}$, the time required to perform a two-qubit unitary on average. The overall estimate of the expected time taken is then δ_w/p_0, where δ_w is the time taken for the deterministic case with time weights scaling by a factor $\beta = 1/\alpha$ at each level. We find that this predicts very well the rate at which the GHZ state can be constructed over a wide range of α values (Fig. 9). The expected time remains $\Theta(N^{\log_n(1/\alpha)})$.

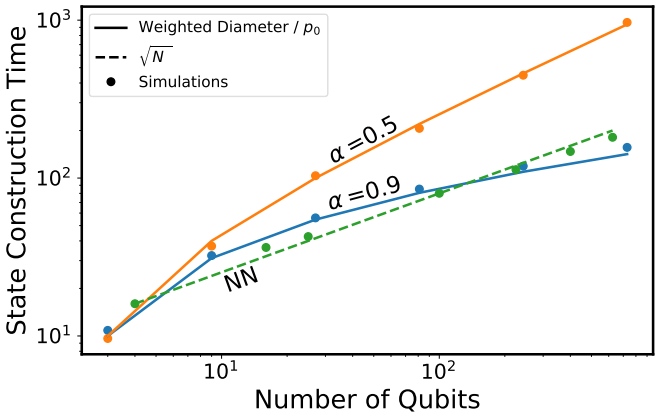

FIG. 9. Graph-theoretic predictions and simulation of t_{GHZ} for the hierarchy $K_3^{\Pi_\alpha k}$ at various α, and a two-dimensional nearest-neighbor (NN) grid; $p_0 = 0.1$. The $\sqrt{N}$ fit shows the scaling of t_{GHZ} for the nearest-neighbor case, with a prefactor in the range suggested by the text's argument. Note that since $n = 3$, the crossover for the hierarchy to asymptotically outperform the nearest-neighbor grid is at $\alpha \geqslant 1/\sqrt{3} \approx 0.58$, which is seen in the numerical results. Code for generating this figure can be found at [54].

For graphs with multiple potential paths between two nodes, such as a two-dimensional grid, the expected time is not simply the deterministic time scaled by the extra time factor the probabilistic setup requires in each step. We can however still bound the expected time to build the GHZ state $\mathbb{E}[t_{\mathrm{GHZ}}]$ above and below for a graph H. We will bound it above by considering a modified graph in which the only path between the initial qubit and the qubit farthest from the starting point has distance $\delta_w(H)$. Such a path completes in time $\delta_w(H)/p_0$ on average. Since H has strictly more paths than this, the expected time will be lower. However, the shortest path between the initial and final qubits has total distance $\delta_w(H)$, which would take time $\delta_w(H)$ to complete even if $p_0 = 1$ and all gates were deterministic. Therefore, no path can finish faster than this, and the expected outcome over all possible paths cannot improve over $\delta_w(H)$. We can therefore write the following restriction on the expected time:

$$\delta_w(H) \leqslant \mathbb{E}[t_{\mathrm{GHZ}}] \leqslant \frac{\delta_w(H)}{p_0}, \tag{47}$$

where $\mathbb{E}[\cdot]$ denotes the expected value. This implies $\mathbb{E}[t_{\mathrm{GHZ}}] = \Theta[\delta_w(H)]$. Therefore, although the prefactor is difficult to calculate, we can tell that the time required to complete the creation of a GHZ state on the nearest-neighbor graph with $d = 2$ is $\Theta(\sqrt{N})$. This scaling implies that the condition for the hierarchy to outperform the nearest-neighbor grid in 2D is $\alpha \geqslant n^{-1/2}$, which is reflected in Fig. 9.

Using the GHZ-creation time and state transfer as examples, we can see many of the advantages of hierarchical graphs as network topologies. Such architectures are able to rapidly incorporate a very large number of qubits (exponential in the number of hierarchy levels), while the time-weighted diameter (and thus communication time) grows linearly with the number of levels. Since the weighted diameter is not substantially changed even if we use the truncated hierarchical

product of Sec. II B 5, these benefits can also be realized in that setup.

V. CIRCUIT PLACEMENT ON HIERARCHIES

A final reason we believe hierarchies could be a useful way to organize modular quantum systems is that they may be able to take advantage of straightforward methods for circuit placement. Circuit placement is a problem that arises when a quantum circuit or algorithm must be translated onto a physical system [55]. Suppose we are given a specification for a quantum algorithm in the form of a circuit diagram, and we wish to run that algorithm on a given quantum computer (which presumably has enough quantum memory to perform that algorithm). In order to translate the circuit into instructions for our machine, we must identify each algorithm qubit with a machine qubit and then determine how the individual quantum gates can be realized in our machine [56].

Circuit placement is an important part of the quantum software stack, just as the compilation to machine code is in classical computers. By placing qubits which must operate on each other often close together in the real-world machine, we can minimize the amount of time spent performing long-range quantum gates. However, this problem is generally quite difficult for arbitrary instances and in fact has been shown to be NP complete [55].

However, since we are interested in the subproblem of circuit placement on hierarchies, it is possible that the hardness results of Ref. [55] do not apply and the exact solution can be found in polynomial time, just as the problem can be solved tractably in linear qubit chains [57]. Whether or not an exact algorithm exists, we can appeal to heuristics to efficiently place circuits as well as possible. Such an approach is promising because hierarchies are extremely structured with clear prioritization of clustering between small groups of qubits, which can be recognized in the algorithm and matched to the physical architecture.

To explain further, we consider the following model. We suppose that we begin with a weighted circuit graph C with a vertex set V_C and an edge set E_C, in which an edge exists between two vertices if there is at least one two-qubit gate between them in the circuit, with the weight of the edge corresponding to the number of gates. We then specify a machine graph M with vertex set V_M and edge set E_M, in which each edge (u, v) indicates that the machine can perform two-qubit gates between u and v.

We now seek a mapping $f : V_C \to V_M$ that assigns algorithm qubits to machine qubits. A mapping f has a total cost found by considering, for every edge in E_C between vertices c_i and c_j, the shortest-path distance between $f(c_i)$ and $f(c_j)$ in M, multiplying that distance by the weight of the edge in C and summing over all edges. Thus, it captures the total distance that must be traversed by all gates in order to execute the circuit when the current mapping is used. Reducing this is expected to reduce the amount of time spent performing SWAP gates in order to connect two distant qubits. Performing this mapping is an important subroutine in any quantum programming framework, and at least one existing quantum compiler has a "mapper" phase that takes into account the actual graph that a program must be compiled onto [58].

Our cost function is a choice made from convenience, and others are possible. Using this cost function ignores several important aspects of quantum circuits. First, our cost function does not account for the fact that a different mapping might allow for more parallelism, since it evaluates the cost of each gate individually. In addition, we take the circuit graph C as a given, when in fact many different circuits exist for any given quantum operation. In fact, it is likely that optimization of C could be performed, possibly by using the structure of M itself. A more realistic model for circuit placement may require a back-and-forth in which a circuit is first placed, then optimized, then re-placed, and so forth. A more advanced placement algorithm may even permit the swapping of qubits throughout the circuit, thus optimizing the placement of the quantum algorithm without constructing a circuit connectivity graph as an intermediate step.

For this paper we will ignore these concerns and proceed with a heuristic approach to circuit placement for hierarchies. We describe our algorithm as "partition and rotate," as it requires these two basic subroutines. First, qubits are partitioned into subhierarchies by examining whether they are connected by many gates in C. This process continues recursively, with each partition being subdivided and so on until every qubit is identified with its point in the hierarchy. This top-down process is then followed by a bottom-up process in which each small cluster is rotated so that its most-communicative qubit is at the root of the subhierarchy, and then the partitions themselves are rotated, and then clusters of clusters, etc. Ideally, this results in a mapping in which every qubit is (a) placed close to qubits it needs to communicate with and (b) placed in easy access to other modules if that qubit requires such access. We will now explore in detail these subroutines and the circuit speed-ups that result. We will place algorithms on a machine graph M which we take to be defined by $K_n^{\Pi k}$ for some integer k. Note that we examine unweighted hierarchies, but these methods can be applied to weighted hierachies as well.

A. Partitioning

For the first step of our algorithm, we wish to divide the computational graph C into n subgraphs which are as disconnected as possible. In addition, since we wish to assign each node in C to physically separate and limited qubit registers, it is important that each of the subsets has precisely $|C|/n$ nodes. This problem is known as *balanced graph partitioning*, and the problem of finding the optimal solution is NP complete for $n \geqslant 3$ [59]. However, heuristic methods exist which approximate the solution, and are widely used in the field of parallel computing and circuit design [60]. We have illustrated this process in Fig. 10.

Our method for performing circuit placement on hierarchies relies on a subroutine that performs balanced graph partitioning. There are many algorithms and software packages from which to choose. Here we have used a software package called Metis, which implements an algorithm called recursive bipartitioning [60,61].

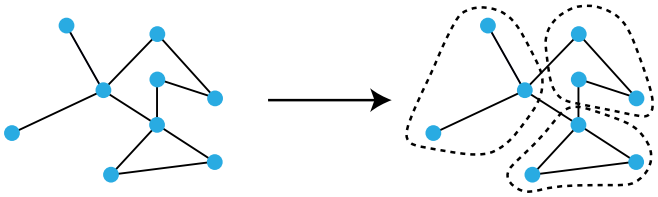

FIG. 10. Illustration of how we might divide a hypothetical graph into smaller clusters. This process is repeated many times, recursively.

We begin by supposing that we have the circuit graph C and we wish to identify groups of $|C|/n$ nodes which have high connection to each other but low connection outside of the group. This is accomplished by finding a balanced graph partition in which the weight of the edges connecting each group is minimized. If we call the initial set of all nodes S, then we wish to identify subsets $S_0, S_1, \ldots, S_n$. In terms of the addressal scheme of Sec. II B 3, all the nodes in set S_i will have have digit i in their base-n representation. In the next section we will discuss the choice of which digit to assign to each set.

Once the subsets S_i are found, partitioning can be run again on that relevant subgraph, creating n new subsets of this subset. Eventually every node in the graph will be identified with a lowest-level module of size n, a next-level module of size n^2, and so forth.

Here we have used a generalized, preexisting algorithm for graph partitioning. It is possible that the specifics of this problem, and the possibility of co-designing the precise quantum circuit implementing the algorithm (and thus C) with the architecture, enable more specific, better-performing approaches.

B. Rotation

Drawing partitions between qubits is not enough to fully specify their placement into a hierarchy. If we consider using the i-digit representation, we can imagine that partitioning essentially describes the process of deciding, from a set of qubits, which ones will share a digit in the next level. However, these digits are more than arbitrary markers, because there is one node in any subhierarchy which connects to the hierarchy above. This node (which we say has digit 0) has privileged access to communication with other subhierarchies. Therefore, in order for our circuit placement to succeed, we should ensure that the qubit on top of each subhierarchy is the one which requires the most access.

In order to do this, we implement a second subroutine, the "rotate" part of the algorithm. This is called rotation because, once we know which qubits will be together in a module, we must choose how to orient them relative to the larger modular structure. Whereas partitioning is top-down (the full graph is broken into small subgraphs which are then themselves partitioned), rotation is bottom-up. Suppose the modular structure is $K_n^{\Pi k}$. We begin with sets of n qubits and must choose which will be the top of each smallest instance of K_n. We then take each partition of n instances of K_n and decide which instance of K_n will connect to the next level up, and so on. This process is illustrated in Fig. 11.

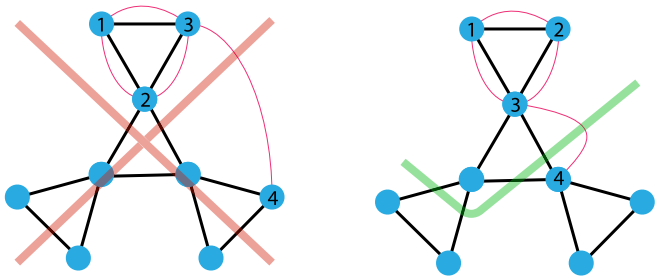

FIG. 11. An illustration of how and why the process of rotation works in our circuit placement algorithm. In this diagram, red links represent gates to be performed (edges in C) and black ones are available communicative links (edges in M). In the graph C, the qubits 1, 2, and 3 are all connected, and 3 is connected with 4. These qubits have been correctly placed into clusters (1, 2, 3) and (4). However, if they are not rotated correctly (see left), the link between 3 and 4 can become quite long, necessitating a long-range quantum gate. By properly rotating (right), the gate between links 3 and 4 becomes much shorter, improving the placement.

Note that the general structure of our algorithm is to first go down the hierarchy, partitioning nodes, and then to go up, rearranging subhierarchies in the proper order. We perform this procedure only once to obtain our circuit mapping.

C. Results

Now that the placement algorithm is specified, we turn toward examining its performance on quantum circuits. We consider two separate questions. First, we investigate whether the algorithm is effective—does it actually reduce, relative to a random assignment, the amount of distance that must be traversed in a circuit to execute all the requested gates? Second, we will examine whether the algorithm executes efficiently on a classical computer. This second point is important because in general the problem can be solved by brute-force search, but such a search requires a time $O(N!)$ to perform (although, as we stated earlier, it is possible that an exact algorithm exists with a lower time cost for the special case of hierarchies).

To investigate the above concerns, we examine the algorithm's performance on random circuits. For each trial, we first generate a random circuit of N_g two-qubit gates on N total qubits. The precise type of two-qubit gate is irrelevant in this framework. Likewise, single-qubit gates require no communication overhead, so we do not consider them. The random circuit then implies a computational graph C, where, as described above, the vertices represent the algorithm qubits and the edge weights represent the number of gates that must be applied between each pair of qubits. Once this computational graph has been generated, we first attempt to map it blindly to the hierarchy graph, using the addressing scheme of Sec. II B 3 and an arbitrary order of the graph C. Then, we apply partition-and-rotate and calculate the new cost function. By comparing the cost function between these two, we develop an idea of how much long-range quantum information processing is eliminated by partition-and-rotate. We perform this several times to build up statistics on average

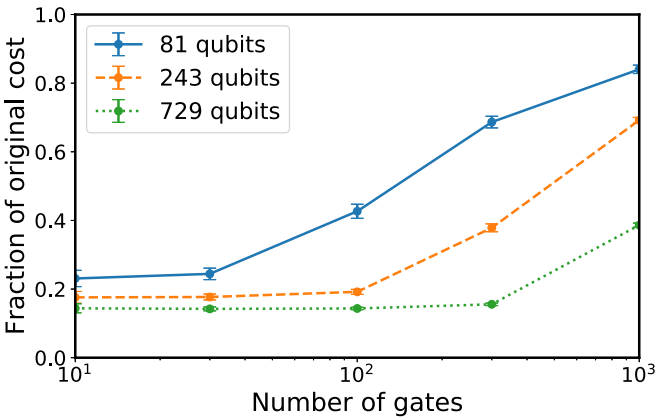

FIG. 12. Plot of the average ratio (total gate distance after partition-and-rotate)/(total gate distance before) given 100 trials each for different numbers of random gates and random qubits. Error bars represent one standard deviation. As the number of gates begins to saturate the number of qubits, the possible improvement from optimization begins to decrease.

time costs and average improvement. Code which performs circuit placement and generates the profiling figures included in this section can be found at [54].

In our simulations, we test hierarchies $K_3^{\Pi k}$ up to 729 qubits ($k = 6$). We find that as gates are added, the improvement over the initial cost is decreased. This is sensible, because as more randomly placed gates are present, different node mappings become more similar. Such an effect will likely not be present for quantum algorithms which do not have their gates placed randomly. For cases in which the number of gates is significantly fewer than the number of qubits, partition-and-rotate is able to significantly reduce the cost function. We find that 100 gates can be placed on a 729 qubit hierarchy with a total cost less than 20% of the original on average. When 1000 gates are placed on a 729 qubit hierarchy, the final cost is still only 40% of the initial one. Results for $K_3^{\Pi 4}$, $K_3^{\Pi 5}$, and $K_3^{\Pi 6}$ can be seen in Fig. 12.

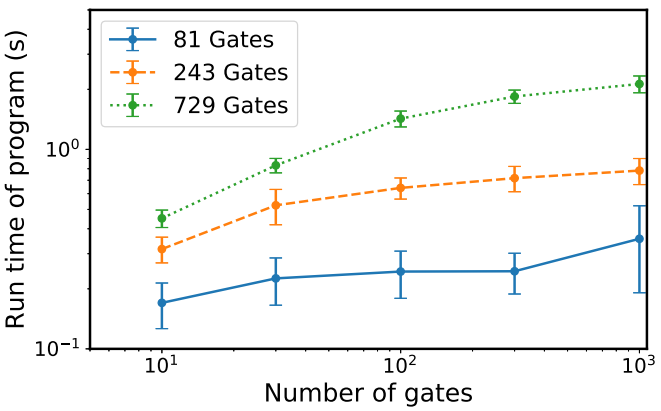

FIG. 13. Average run times over 100 trials for partition-and-rotate on a 2015 MacBook Pro with a 2.6 GHz processor [61]. Each line represents an increasing number of gates for a constant circuit size as measured by the number of qubits. Error bars represent one standard deviation.

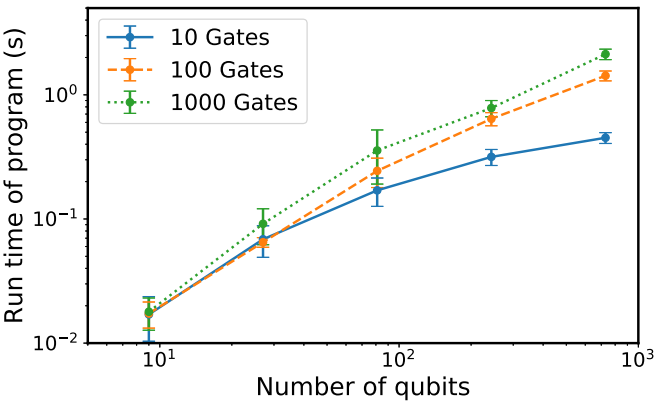

FIG. 14. Average run times over 100 trials for partition-and-rotate on a 2015 MacBook Pro with a 2.6 GHz processor [61]. Each line represents an increasing number of qubits for a constant number of gates. Error bars represent one standard deviation.

Next, we examine the time required to place such a circuit. Our code, most of which is written in Python3 but which uses a C implementation of Metis for graph partitioning, can place 1000 gates on a 729-qubit hierarchy in roughly 2 s when running on a 2015 MacBook Pro [61]. Although the algorithm seems naturally suited to parallelization, our implementation uses only a single core. Our current implementation appears to scale with the number of qubits as $O(N)$ and not to depend on the number of gates included at all once there are a sizable number of gates. We illustrate these two relationships in Figs. 13 and 14. These times compare favorably to the times reported in Ref. [55], with much optimization still possible in our implementation.

Note that using random graphs as described above means that our results may not be valid for more general quantum algorithms. It is possible that practical quantum algorithms have structure that makes them either particularly amenable or particularly difficult for partition-and-rotate algorithms to place, depending on the actual algorithm being examined.

VI. CONCLUSIONS AND OUTLOOK

In this paper we have developed the theory of hierarchies using the existing binary operation of the graph hierarchical product. We have shown that hierarchies may be a promising architecture for large quantum information processing systems. To demonstrate this, we analyzed both properties of the underlying graph (such as diameter, maximum degree, total edge weight) as well as the time it would require to perform a representative quantum information process (constructing the GHZ state/state transfer) in both deterministic and probabilistic settings. We have also computed and tabulated these properties for many other graphs which appear as potential architectures, for comparison. We have shown that, for much of parameter space, hierarchies have favorable scalings in cost and performance with the total number of qubits N compared to these competitors. Also, since hierarchical graphs are hyperbolic, they share many of the advantages of hyperbolic graphs such as efficient routing schemes [62], network security [63], and node addressal [64].

We have also presented a conceptually simple circuit placement algorithm which allows for simple optimization using existing graph-partitioning software packages. Our partition-and-rotate algorithm scales well with the number of qubits and gates in the circuit and reliably reduces the total distance that needs to be traversed by random quantum circuits, which we verified by simulation.

One significant limitation of our analysis in this paper has been that we remained confined to unitary operations. Nonunitary operations (for instance, measurements which are then fed forward to choose future unitary operations) are capable of establishing long-range correlations like those present in the GHZ state much more quickly than unitary ones if measurements and classical communication are fast. In the future we hope to extend our results into nonunitary domains [65].

In addition, we have made the assumption that the primary way in which quantum architectures will differ is the speed with which two qubits can communicate (as represented by our time weights on edges). Another important case might be one in which the primary way edges are enhanced is by improving bandwidth or duplicating nodes to provide parallel routes rather than affecting gate speed directly. For some schemes, our abstract notion equating the time of a two-qubit gate with the edge weight may still be a useful tool of analysis, but in other cases bandwidth and speed may not be interchangeable. We intend to undertake the analysis appropriate for this case in a future manuscript [65].

In this paper we limited ourselves to consideration of a few quantum processes (generation of a large entangled state or transfer of a state across the graph), which might not be representative of other, more general distributed quantum information tasks. Some algorithms, such as Shor's algorithm, are known to be able to run with little additional overhead even on one-dimensional, nearest-neighbor graphs [66]. Therefore, when selecting an architecture for a practical quantum computer, care will need to taken to select the proper benchmarking task.

In future work we hope to look at a wider variety of quantum circuits and use those to better benchmark different modular architectures. In addition, we hope to gain a better understanding of the treatment of probabilistic links for general graphs. For instance, as we discussed briefly when assessing the star graph S_N, one real concern in a networked setting is whether some parts of the network will form bottlenecks. To analyze the impact of this in a general way will require a better understanding of realistic quantum algorithms and the demands they place on a network. Analyzing more complex quantum algorithms could also shed light on the performance of partition-and-rotate placement algorithms in realistic settings when sequencing and scheduling also enter into consideration.

Finally, in addition to asking ourselves how current circuits and algorithms can be executed on highly modular systems, we also hope to explore the possibility that highly modular architectures open up new possibilities for parallelized quantum algorithms. For instance, Ref. [67] shows that quantum fan-out gates can be used to parallelize gate sequences, decreasing the time to perform an algorithm at the cost of requiring additional memory qubits. Hierarchies could implement such schemes by using high-level connections to perform the initial fan-out gates and then performing the various parallelized operations in each individual module.

ACKNOWLEDGMENTS

We thank A. Childs, L. Jiang, C. Monroe, A. Houck, K. Hyatt, I. Kim, A. Kollár, M. Lichtman, Y.-K. Liu, D. Maslov, E. Schoute, A. Wallraff, and D. Wentzlaff for discussions. This work was supported by ARO MURI, ARL CDQI, NSF Ideas Lab, ARO, NSF PFC at JQI, and the Department of Energy ASCR Quantum Testbed Pathfinder program. A.B. is supported by the QuICS Lanczos Fellowship. Z.E. is supported in part by the ARCS Foundation. J.R.G. is supported by the the NIST NRC Research Postdoctoral Associateship Award. F.C. is supported by NSF Expeditions in Computing Grant CCF-1730449, Los Alamos National Laboratory and the U.S. Department of Defense under subcontract 431682, by NSF PHY Grant 1660686, and by a research gift from Intel Corporation. This work was performed in part at the Aspen Center for Physics, which is supported by National Science Foundation Grant PHY-1607611.

[1] R. Van Meter and K. M. Itoh, Phys. Rev. A **71**, 052320 (2005).
[2] R. V. Meter, W. J. Munro, K. Nemoto, and K. M. Itoh, ACM J. Emerg. Technol. Comput. Syst. **3**, 1 (2008).
[3] M. Ahsan and J. Kim, in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)* (IEEE Conference Publications, Piscataway, NJ, 2015), pp. 1108–1113.
[4] T. Metodi, D. Thaker, and A. Cross, in *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)* (IEEE, Piscataway, NJ, 2005), pp. 305–318.
[5] L.-M. Duan and C. Monroe, Rev. Mod. Phys. **82**, 1209 (2010).
[6] C. Monroe and J. Kim, Science **339**, 1164 (2013).
[7] M. H. Devoret and R. J. Schoelkopf, Science **339**, 1169 (2013).
[8] T. Brecht, W. Pfaff, C. Wang, Y. Chu, L. Frunzio, M. H. Devoret, and R. J. Schoelkopf, Npj Quantum Inf. **2**, 16002 (2016).
[9] P. Kurpiers, P. Magnard, T. Walter, B. Royer, M. Pechal, J. Heinsoo, Y. Salathé, A. Akin, S. Storz, J.-C. Besse, S. Gasparinetti, A. Blais, and A. Wallraff, Nature (London) **558**, 264 (2018).
[10] A. A. Hagberg, D. A. Schult, and P. J. Swart, in *Proceedings of the 7th Python in Science Conference*, edited by G. Varoquaux, T. Vaught, and J. Millman (Pasadena, CA, 2008), pp. 11–15.
[11] L. Barrière, F. Comellas, C. Dalfó, and M. Fiol, Discrete Appl. Math. **157**, 36 (2009).
[12] C. Godsil and B. McKay, Bull. Austral. Math. Soc. **18**, 21 (1978).
[13] J. J. Bollinger, W. M. Itano, D. J. Wineland, and D. J. Heinzen, Phys. Rev. A **54**, R4649 (1996).
[14] Z. Eldredge, M. Foss-Feig, J. A. Gross, S. L. Rolston, and A. V. Gorshkov, Phys. Rev. A **97**, 042337 (2018).

[15] Z. Eldredge, Z.-X. Gong, J. T. Young, A. H. Moosavian, M. Foss-Feig, and A. V. Gorshkov, Phys. Rev. Lett. **119**, 170503 (2017).

[16] S. Bravyi, M. B. Hastings, and F. Verstraete, Phys. Rev. Lett. **97**, 050401 (2006).

[17] G. Bentsen, Y. Gu, and A. Lucas, arXiv:1805.08215.

[18] M. Cuquet and J. Calsamiglia, Phys. Rev. A **86**, 042304 (2012).

[19] A. Ać́in, J. I. Cirac, and M. Lewenstein, Nat. Phys. **3**, 256 (2007).

[20] S. Perseguers, G. J. Lapeyre, D. Cavalcanti, M. Lewenstein, and A. Acín, Rep. Prog. Phys. **76**, 096001 (2013).

[21] K. Kieling, T. Rudolph, and J. Eisert, Phys. Rev. Lett. **99**, 130501 (2007).

[22] M. Cuquet and J. Calsamiglia, Phys. Rev. Lett. **103**, 240503 (2009).

[23] M. Cuquet and J. Calsamiglia, Phys. Rev. A **83**, 032319 (2011).

[24] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, in *22nd International Symposium on Reliable Distributed Systems* (IEEE Computer Society, Washington, DC, 2003), pp. 25–34.

[25] D. J. Watts and S. H. Strogatz, Nature (London) **393**, 440 (1998).

[26] A.-L. Barabási and R. Albert, Science **286**, 509 (1999).

[27] R. Albert and A.-L. Barabási, Rev. Mod. Phys. **74**, 47 (2002).

[28] S. Perseguers, M. Lewenstein, A. Ać́in, and J. I. Cirac, Nat. Phys. **6**, 539 (2010).

[29] C. Di Franco and D. Ballester, Phys. Rev. A **85**, 010303 (2012).

[30] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim, Phys. Rev. A **89**, 022317 (2014).

[31] H.-J. Briegel, W. Dür, J. I. Cirac, and P. Zoller, Phys. Rev. Lett. **81**, 5932 (1998).

[32] F. Harary, Trans. Am. Math. Soc. **78**, 445 (1955).

[33] M. Gromov, in *Metric Structures for Riemannian and Non-Riemannian Spaces*, edited by J. LaFontaine and P. Pansu, Modern Birkhäuser Classics (Birkhäuser, Basel, Switzerland, 2007).

[34] W. Chen, W. Fang, G. Hu, and M. W. Mahoney, Internet Math. **9**, 434 (2013).

[35] P. Lohsoonthorn, Ph.D. Thesis, University of Southern California, Los Angeles, CA, 2003.

[36] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguñá, Phys. Rev. E **82**, 036106 (2010).

[37] F. de Montgolfier, M. Soto, and L. Viennot, in *Proceedings of the 2011 IEEE 10th International Symposium on Network Computing and Applications*, NCA '11 (IEEE Computer Society, Washington, DC, 2011) pp. 25–32.

[38] M. Boguñá, F. Papadopoulos, and D. Krioukov, Nat. Commun. **1**, 62 (2010).

[39] A. J. Kollár, M. Fitzpatrick, and A. A. Houck, arXiv:1802.09549.

[40] S.-i. Oum, J. Comb. Theory Ser. B **95**, 79 (2005).

[41] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width* (Springer, Berlin, 1998), pp. 1–16.

[42] C. E. Leiserson, IEEE Trans. Comput. **C-34**, 892 (1985).

[43] M. W. Newman, Master's Thesis, University of Manitoba, Winnipeg, Canada, 2000.

[44] B. Mohar, *Graph Theory, Combinatorics, and Applications* (Wiley, New York, 1991), pp. 871–898.

[45] F. R. K. Chung, *Spectral Graph Theory* (American Mathematical Society, Providence, RI, 1997), Chap. 2.

[46] D. Cheung, D. Maslov, and S. Severini, in *Workshop on Quantum Information Processing* (Brisbane, Australia, 2007).

[47] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, Proc. Natl. Acad. Sci. **114**, 3305 (2017).

[48] If, for the application at hand, a planar graph is required, cycles such as C_n can yield the same scaling.

[49] R. S. Pindyck and D. L. Rubinfeld, *Microeconomics* (Pearson, London, 2013).

[50] Z. Füredi, Graphs Comb. **6**, 333 (1990).

[51] A. J. Hoffman and R. R. Singleton, IBM J. Res. Dev. **4**, 497 (1960).

[52] M. Miller and J. Sirán, Electron. J. Comb. **1000**, DS14 (2005).

[53] G. Pineda-Villavicencio and D. R. Wood, Electron. J. Comb. **22**, 2.46 (2015).

[54] Z. Eldredge *et al.*, unitary-modular, https://github.com/zeldredge/unitary-modular, GitHub repository.

[55] D. Maslov, S. M. Falconer, and M. Mosca, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **27**, 752 (2008).

[56] We studiously avoid referring to the machine qubits as "physical" in this paper, as we do not want to confuse this conceptual distinction with the physical/logical qubit divide in error correction. All of the qubits referenced in this section are logical qubits in the error-correcting sense.

[57] M. Pedram and A. Shafaei, IEEE Circuits Syst. Mag. **16**, 62 (2016).

[58] D. S. Steiger, T. Häner, and M. Troyer, Quantum **2**, 49 (2018).

[59] K. Andreev and H. Racke, Theory Comput. Syst. **39**, 929 (2006).

[60] G. Karypis and V. Kumar, SIAM J. Sci. Comput. **20**, 359 (1998).

[61] Specific product citations are for the purpose of clarification only, and are not an endorsement by NIST.

[62] R. Kleinberg, in *IEEE INFOCOM 2007* (IEEE, Piscataway, NJ, 2007), pp. 1902–1909.

[63] E. Jonckheere and P. Lohsoonthorn, in *Proceedings of the 2004 American Control Conference* (IEEE, Boston, MA, USA, 2004), Vol. 2, pp. 976–981.

[64] V. Chepoi, F. F. Dragan, B. Estellon, M. Habib, Y. Vaxès, and Y. Xiang, Algorithmica **62**, 713 (2012).

[65] Z. Eldredge *et al.* (unpublished).

[66] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg, Quantum Inf. Comput. **4**, 237 (2004).

[67] P. Hoyer and R. Spalek, Theory Comput. **1**, 81 (2005).