A network configuration algorithm based on optimization of Kirchhoff index

Adam Hackett*, Deepak Ajwani†, Shoukat Ali‡, Steve Kirkland*, and John P. Morrison†
*Hamilton Institute, National University of Ireland Maynooth, Ireland
{adam.hackett, stephen.kirkland}@nuim.ie
†The Centre for Unified Computing, University College Cork, Ireland
{d.ajwani, j.morrison}@cs.ucc.ie
‡Exascale Systems Group, IBM Dublin Research and Development Lab, Ireland
shoukat.ali@ie.ibm.com

***Abstract*—Traditionally, a parallel application is partitioned, mapped and then routed on a network of compute nodes where the topology of the interconnection network is fixed and known beforehand. Such a topology often comes with redundant links to accommodate the communication patterns of a wide range of applications. With recent advances in technology for optical circuit switches, it is now possible to construct a network with much fewer links, and to make the link endpoints configurable to suit the communication pattern of a given application. While this is economical (saving both links and the power to run them), it raises the difficult problem of how to configure the network and how to reconfigure it quickly when the application's communication pattern changes.**

In this paper, we propose the Kirchhoff index (KI) of a certain weighted graph related to the interconnection network as a proxy for its communication throughput. Our usage of this metric is based on a theoretical analogy between resistances in an electrical network and communication loads in the interconnection network. We show how mathematical techniques for reducing KI can be used to configure a network in a dramatically shorter time as compared to the current state-of-the-art scheme.

***Keywords*-Graph partitioning algorithm; Reconfigurable topology; Optical circuit switch; Kirchhoff index**

I. Introduction

Recently, there has been a new opportunity for increasing the performance of parallel applications. This opportunity has been afforded by optical switches that can quickly configure an interconnect topology to suit a particular application. The freedom to configure the interconnect to suit the requirements of an application is a potentially rich vein of theoretical investigation. Whereas the traditional parallel computing task has been to assign computational clusters and a communication pattern to a static interconnect in order to elicit, for example, the best possible throughput, one can now explore the possibility of improving performance even further by selecting a topology whose characteristics are advantageous to a given task graph. While the first optimization is well known and well studied in the literature (e.g., [1], [2], [3]), the latter optimization of the interconnect topology has not been well studied. This is mostly because configurable optical switches have only recently become feasible in terms of price and switching time, and have therefore recently afforded configuration of the interconnect topology on an application-by-application basis. Accordingly, there has been considerable recent research (e.g., [4], [5], [6]) in tailoring the interconnect to suit the requirements of a given application, in particular a stream computing application.

Stream computing is an important computing model that captures applications like real-time analysis of financial and medical data [7], audio/video systems, continuous database queries, intelligent transportation systems [8] and analysis of dynamic social networks [9], [10]. These applications are termed as stream computing applications, in that they consist of processing continuous, high-volume data-streams in real time. Such applications are particularly suited for utilizing a configurable parallel architecture, like the one enabled by a configurable optical switch ([11], [12], [13], [14]), because these applications have long duration flows (in seconds) that can easily amortize the cost of configuring the switch (usually in a few 10s of millisecs).

Recently, an iterative co-optimization technique [5] has been proposed to solve this problem. It takes turns in optimizing the application partitioning and routing for a given interconnect topology, and optimizing the interconnect topology to alleviate the bottlenecks identified in the partitioning and routing phase. A major problem with this approach is that it is computationally expensive to repeatedly compute good solutions for partitioning, mapping and routing. Although this iterative co-optimization framework has been an important first step in tackling this problem in its entirety, the running time of this approach makes it difficult to use it in real systems.

In this paper, we propose a different framework for tackling this problem. Rather than repeatedly computing partitioning, mapping and routing solutions to estimate the throughput obtained from an interconnect topology, we compute such a solution only once and then use a metric (the Kirchhoff index of a certain weighted matrix related to the topology) dependent on the topology that correlates well with the throughput. By optimizing the interconnect topology for this metric, we can directly obtain a topology that elicits good throughput for the given application. The central component of this algorithm is a novel edge rewiring scheme that greedily minimizes the Kirchhoff index based metric.

We show that the topologies obtained from this algorithm provide a performance comparable to (and in many cases, much larger than) that obtained from the iterative co-optimization approach, but crucially, the time to search for a good topology is *dramatically* reduced. This makes the proposed scheme much better suited for deployment in real systems. As one motivating example, our algorithm configures a 16-node network so that it suits the characteristics of a popular graph benchmark,as-22july06, in a little over 6 *seconds* while the iterative co-optimization technique [5] takes more than 6 *hours*. At the same time, our algorithm achieves about *3 times higher throughput*.

Another problem with the iterative co-optimization framework is its dependence on a large number of heuristics for partitioning, mapping and routing. It is difficult to determine what combination of heuristics will provide the best throughput for a given application graph and to tune the heuristic parameters. It is even more difficult to achieve a good throughput-time trade-off. We may be willing to sacrifice some throughput if it leads to a considerable reduction in the time to compute it.

The usage of heuristics seems inevitable as most variants of partitioning, mapping and routing problems are NP-complete. But by relating our problem to that of optimizing the Kirchhoff index of a related electrical network and by using a simple, greedy, polynomial time algorithm for it, our proposed scheme reduces the number of heuristics and associated parameters considerably.

II. Preliminaries

In this section we provide an outline of the problem to be addressed, and describe the notations and definitions used in the remainder of the paper.

A. Space of Configurable Topologies

We consider the problem of configuring the interconnect topology of a computing system built around a reconfigurable switch (e.g., an optical circuit switch). Such a switch allows one to configure any topology $H(V_H, E_H)$ satisfying the constraints that (i) the maximum degree of a node in H is at most $d_{\max}$ (referred to as the *maximum degree constraint*) and (ii) the total number of links in H is at most $E_{\max}$ (referred to as the *maximum edge constraint*). These constraints derive respectively from the fact that each compute node has a limited number of ports to connect to the configurable switch, and that the switch itself has a limit on the number of links that it can maintain simultaneously. Furthermore, we assume that each compute node in H has the same processing speed, denoted by $\underline{S_{\text{C}}}$. We also assume that each link in H has the same bandwidth, denoted by $\underline{S_{\text{L}}}$.

From among the large space of topologies that fit this basic description, we are interested in computing a topology that elicits a high throughput for a given instance of a stream-computing application.

B. Stream-Computing Application Task Graphs

The computational task graph of a stream-computing application is represented as an undirected weighted graph $G(V_G, E_G)$. Each vertex $u \in V_G$ denotes a computational kernel (e.g., a split, join, or filter) and its weight $w_G(u)$ quantifies the average amount of computation that must be performed in the corresponding kernel to produce an element of output. Similarly, each edge $\{u, v\} \in E_G$ represents a data-stream (capturing the data dependency) between the kernels u and v and its weight $w_E(\{u, v\})$ represents the average amount of data transfer between these two kernels.

C. Suitability of a Topology for a Given Application

To evaluate the suitability of a topology H for a given application task graph G, we simulate a run of G on H. In this run, each vertex $v_i \in V_G$ is mapped to a compute node $u_i \in V_H$ and each edge $\{v_i, v_j\} \in E_G$ is routed along a sequence of links (or *path*) in H. Let $\mu(v_i)$ determine the node to which a particular vertex v_i is mapped, and let $\rho(v_i, v_j)$ determine the sequence of links that are used to route an edge $\{v_i, v_j\} \in E_G$. Given such mapping and routing schemes, the computational load on each node $u_i \in V_H$ is

$$w_V(u_i) = \sum_{\substack{u_i=\mu(v_i) \\ v_i \in V_G}} w_V(v_i), \tag{1}$$

and the communication load on each link $\{u_i, u_j\} \in E_H$ is

$$w_E(\{u_i, u_j\}) = \sum_{\substack{\{u_i,u_j\} \in \rho(v_i,v_j) \\ \{v_i,v_j\} \in E_G}} w_E(\{v_i, v_j\}). \quad (2)$$

Based on these formulations, we define the throughput of a compute node u_i as $S_{\mathrm{C}}/w_V(u_i)$ and the throughput of a link $\{u_i, u_j\}$ as $S_{\mathrm{L}}/w_E(u_i, u_j)$. The *computation throughput* of the system is then the minimum throughput over all compute nodes, and the *communication throughput* of the system is the minimum throughput over all links. The *system throughput* is the smaller of the computation and communication throughputs. This throughput is used as a performance metric to determine the suitability of a topology for the given application. Note that this definition of throughput arises out of stream computing applications, where we view the compute nodes and communication links as processing units running concurrently so that the overall throughput is equal to the throughput of the slowest processing unit.

D. Optimization of Communication Throughput

The throughput obtained in the preceding evaluation depends crucially on the algorithm used to map and route the application to the topology. Since most variants of topology-aware partitioning, mapping and routing algorithms are NP-hard, one often relies on heuristics or approximation algorithms to obtain efficient solutions. This makes it difficult to optimize for the throughput directly. Therefore, to achieve this optimization we propose an alternative metric that correlates well with throughput (and in particular, communication throughput for reasonable solutions of partitioning and routing algorithms) and for which it is possible to optimize.

Our chosen metric is the Kirchhoff index [15] of a certain weighted graph where weights on this graph are derived from the mapping of the application graph to the interconnect topology. We start with an initial topology, obtained, for example, from the algorithm of [5], and a mapping of the application graph to it. We employ a network configuration algorithm designed to iteratively reduce the Kirchhoff index of the initial topology, and thereby alleviate potential communication bottlenecks. The theoretical underpinning of this procedure is discussed below in Sec. III-B. First, let us provide the motivation for our use of this metric.

III. Our Methodology

A. Kirchhoff Index and Communication Throughput

In graph theory, the *resistance distance* [15] between two vertices of a connected graph, H, is equal to the effective resistance between two corresponding points on an electrical network that has been constructed so as to match the structure of H, and in which each edge of H has been replaced by a 1 ohm resistor. The resistance distance is a metric on the vertices of any graph. By a similar thought experiment, we may view the interconnect topology with communication loads on its links as an electrical network with resistances. This analogy between the resistances in the electrical network and communication loads on an interconnect topology has been frequently used in the networking literature (e.g., [16], [17], [18], [19], [20]). In our case, the analogy assumes that just as the amount of current (number of charge units per second) in an electrical circuit depends on the conductance of a wire, the data throughput (number of data packets processed per second) over a communication link in an interconnect topology depends on the remaining bandwidth on the link. If a link is heavily congested, the corresponding wire in the electrical network can be considered to be of low conductance. In other words, such a link offers high resistance to any additional flow over it. On the other hand, the wire corresponding to a link with light communication load has a high conductance (low resistance).

We assume that the throughput (our performance metric in this paper) between two vertices in the topology graph will increase if the number of routes between these two vertices increases. This notion is in fact captured in the "multiple-route distance diminishment" [15] property of the resistance distance between a pair of vertices. In our specific context where we have a graph with weighted edges, the resistance distance between vertices u and v is defined as the effective resistance between the two vertices when each graph edge is replaced by a weight that is a function of how G was mapped on H. In particular, we focus on using the *Kirchhoff Index*, which is defined as the sum of resistance distances between all pairs of vertices in H, as an indicator of the resistance of the entire network. This justifies the usage of the Kirchhoff index as a proxy for communication throughput when mapping an application to the interconnect topology because a better value of Kirchhoff index indicates a network that allows a larger value of throughput.

We now give a formal definition of the Kirchhoff index. Let H be an undirected weighted graph on n vertices, with vertex indices $i = 1, \ldots, n$. Given distinct indices i, j we write $i \sim j$ if i and j are adjacent in H, and $i \not\sim j$ if they are not adjacent. The value $w_{i,j}$ is known as the weight of the edge between vertices i and j, and is defined such that whenever $i \sim j$,

$w_{i,j} = w_{j,i} > 0$, with $w_{i,j} = w_{j,i} = 0$ if $i \not\sim j$. The Laplacian matrix for H, denoted L, is given by $L = [l_{i,j}]_{i,j=1,\ldots,n}$, where

$$l_{i,j} = \begin{cases} -w_{i,j} & \text{if } i \sim j, \\ 0 & \text{if } i \neq j, i \not\sim j, \\ d_i & \text{if } i = j. \end{cases}$$

The Kirchhoff index of H can be written as $\mathrm{KI}(H) = n \cdot \mathrm{tr}(L^{\dagger})$, where $L^{\dagger}$ denotes the Moore-Penrose pseudo-inverse [21] of L, and $\mathrm{tr}(\cdot)$ denotes the trace function, e.g., $\mathrm{tr}(H) = \sum_{i=1}^{n} h_{ii}$.

B. Algorithm for Configuring the Initial Topology

We now present a greedy algorithm that iteratively modifies a seed topology in an effort to reduce the Kirchhoff index. To obtain an initial topology we first compute a partitioning of the application graph (using any partitioning library like METIS [22] or SCOTCH [23]) into $|H|$ clusters and then contract each cluster to a single node forming a *condensed graph* (also referred to as a quotient graph in the literature). The weight of an edge in a condensed graph is the sum of weights of edges between the two corresponding partitions in the application graph. We then remove some low-weight edges, if necessary, to satisfy the maximum degree and maximum edge constraints and treat the resultant graph (ignoring its weights) as the initial topology. Since this topology mimics the communication pattern of the application, it is argued (see, e.g., [5]) that the application should map well to this topology, thus making it a good seed topology for our algorithm.

The formation of the condensed graph gives an implicit partitioning and mapping from the application graph to the condensed graph topology. In this implicit mapping, all vertices that were condensed to form a node in the condensed graph are mapped to that node in the topology. We first use the shortest path routing scheme to assign initial routes to all those application edges whose incident vertices are mapped to different nodes in the topology. Later, we consider the application edges in decreasing order of their weight and re-route them using the min-congestion path (using a minimum spanning tree algorithm) in the topology.

The shortest path routing minimizes the total flow summed over all links and the min-congestion routing distributes the flow more evenly at the cost of increasing the total flow by a hopefully small amount.

Once the routing is completed, we can compute the communication load on each link using Eq. 2. We then view the initial topology as an electrical network. Among all links in the topology, let w_E^{max} be the maximum communication load. The conductance w_C of a wire $\{u_i, u_j\}$ in this electrical network is determined by the following equation:

$$w_C(\{u_i, u_j\}) = w_E^{max} + 1 - w_E(\{u_i, u_j\}) \qquad (3)$$

The above metric always assigns a positive value to the conductance and conforms to the intuitive notion that the congested links (with a high existing load) should have low conductance and links with relatively low communication load should have high conductance. By computing the Laplacian matrix of the initial topology with the weights defined by the conductance values, we can now compute the Kirchhoff index of the corresponding electrical network, as described in Sec. III-A.

Once an initial topology has been constructed, we *rewire* repeatedly until the Kirchhoff index of the topology has been reduced below a certain threshold (see Algorithm 1 below). In a given rewiring iteration, an edge is deleted from the topology and a new edge is added to the topology. An edge deletion always worsens the Kirchhoff index (i.e., increases it) because it reduces the number of routes between some pair of vertices. Similarly every edge addition always improves the Kirchhoff index because it increases the number of routes between some pair of vertices. A simple argument (see the last paragraph of the current section) shows that every rewiring iteration carried out in Algorithm 1 either improves the Kirchhoff index or does not change it. The latter scenario occurs when the added edge happens to be the same as the deleted edge.

Algorithm 1 Network configuration algorithm

Input: A partitioning, mapping and routing of the application graph G on the topology graph H, such that the max-degree and max-edge constraints have been satisfied. A value for the tolerance ϵ.

Output: H_2, a topology graph optimized for communication throughput. A routing of G on H_2.

```
1: $H_0 = H$, done = false
2: while not done do
3:     $H_2 =$ rewire($H_0$)
4:     if $\mathrm{KI}(H_0) - \mathrm{KI}(H_2) < \epsilon \mathrm{KI}(H_0)$ then
5:         done = true,
6:     else
7:         $H_0 = H_2$
8:     end if
9: end while
10: Reroute $G$ onto $H_2$ (from scratch).
```

The edge rewiring algorithm outlined in Step 3 of Algorithm 1 and explained in detail in Algorithm 2 below, is the key component of our approach to the task of configuring the topology. However, before we discuss the mathematical details of Algorithm 2, in the interest of clarity, we offer the following general summary of the operations this algorithm performs.

Let H_0, H_1, and H_2 denote the three different topologies that are involved in every iteration of the **while** loop of Algorithm 1. To begin rewiring, we first identify the set, S_{del}, of edges in H_0 whose deletion will not cause H_0 to become disconnected.[1] We then select from S_{del} the edge, $i_0 \sim j_0$, say of weight $w_C(\{i_0, j_0\})$, whose deletion will cause the least increase in the Kirchhoff index of the topology, and delete $i_0 \sim j_0$. Next, from the newly formed topology graph $H_1 = H_0 \setminus \{i_0 \sim j_0\}$ we identify the pairs of vertices that are not already adjacent in H_1, and whose respective degrees are less than d_{max}, so that the graph after inserting an edge between this pair still satisfies the maximum degree constraint. From this set, S_{add}, of pairs of vertices, we then select the pair, i_1, j_1, which if connected by the edge $i_1 \sim j_1$ would cause the largest decrease in the Kirchhoff index of all pairs in S_{add}. We then add the edge $i_1 \sim j_1$ to H_1 to create the graph H_2. Finally, the edge $i_1 \sim j_1$ is assigned the weight, $w_C(\{i_0, j_0\})$, of the deleted edge $i_0 \sim j_0$. Thus, an edge has been removed from the topology and a new edge with the same weight as the removed edge has been added. This procedure ensures that if the input graph satisfied the maximum degree constraint and the total edge constraint, the output graph will also satisfy these constraints.

By repeating this rewiring procedure, we attempt to improve (i.e., decrease) the Kirchhoff index. As indicated earlier, rewiring does not increase the KI. We stop iterating (see Algorithm 1, Step 4) when the decrease in KI obtained from the current iteration is less than a prespecified percentage, ϵ, of the value of KI obtained from the previous iteration. In the final step of Algorithm 1, we re-route the application graph, G, onto H_2.

The purpose of the next four equations (Eqs. (4)-(7)) is to describe the effect of edge deletion or addition on the Kirchhoff index of the topology. These results follow as direct corollaries of Theorem 1 and Lemma 2, shown in the Appendix. Equations (4)-(7) allow us to avoid invoking the computationally expensive procedure of computing the Kirchhoff index from scratch each time an edge is considered for removal or insertion. This, in turn, makes the rewiring procedure extremely fast.

We require the following definitions. For each pair of distinct indices i, j between 1 and n, let $u(i, j)$ denote the vector in $\mathbb{R}^n$ given by $e_i - e_j$, where e_i and e_j are the i–th and j–th standard unit basis vectors in $\mathbb{R}^n$. The proof of Eqs. 4 and 5 follows immediately by applying Theorem 1 (Appendix) with $k = 1$, $U = u(i_0, j_0)$ and $\alpha_1 = w_C(\{i_0, j_0\})$. Observe that since $i_0 \sim j_0$ is not a cut edge, $1 - u(i_0, j_0)^t (L_0^\dagger) u(i_0, j_0) > 0$, by Lemma 2 (Appendix). The proof of Eqs. (6)-(7) is analogous to that of Eqs. 4 and 5, and is omitted.

First, consider edge deletion. Let H_0 be a connected weighted graph on n vertices with Laplacian matrix L_0. Suppose that $i_0 \sim j_0$ is an edge of H_0 with weight $w_C(\{i_0, j_0\})$, and suppose further that $i_0 \sim j_0$ is not a cut edge. Let H_1 denote the weighted graph obtained from H_0 by deleting the edge $i_0 \sim j_0$. Denote the Laplacian matrices of H_0 and H_1 by L_0 and L_1, respectively. Then

$$\begin{aligned} L_1^\dagger =& L_0^\dagger + w_C(\{i_0, j_0\}) \\ & \times \frac{L_0^\dagger u(i_0, j_0) u(i_0, j_0)^t L_0^\dagger}{1 - w_C(\{i_0, j_0\}) u(i_0, j_0)^t L_0^\dagger u(i_0, j_0)} \end{aligned} \tag{4}$$

and

$$\begin{aligned} \mathrm{KI}(H_1) =& \mathrm{KI}(H_0) + w_C(\{i_0, j_0\}) n \\ & \times \frac{u(i_0, j_0)^t (L_0^\dagger)^2 u(i_0, j_0)}{1 - w_C(\{i_0, j_0\}) u(i_0, j_0)^t L_0^\dagger u(i_0, j_0)}. \end{aligned} \tag{5}$$

Thus, Eq. 5 provides a computationally cheap way to calculate the Kirchhoff index, $\mathrm{KI}(H_1)$, of the newly formed graph H_1 after the edge $i_0 \sim j_0$ has been removed from H_0. Similarly, Eq. 4 tells us how to cheaply calculate the new Moore-Penrose pseudo-inverse, $L_1^\dagger$, of L_1 from the old value, $L_0^\dagger$. The new value $L_1^\dagger$ is required to further modify the edge set of H_1.

Next, consider edge addition. Let H_1 be a connected weighted graph on n vertices with Laplacian matrix L_1. Let H_2 denote the weighted graph obtained from H_1 by adding the edge $i_1 \sim j_1$ with weight $w_C(\{i_1, j_1\})$. Denote the Laplacian matrices of H_1 and H_2 by L_1 and L_2, respectively. Then

$$\begin{aligned} L_2^\dagger =& L_1^\dagger - w_C(\{i_1, j_1\}) \\ & \times \frac{L_1^\dagger u(i_1, j_1) u(i_1, j_1)^t L_1^\dagger}{1 + w_C(\{i_1, j_1\}) u(i_1, j_1)^t L_1^\dagger u(i_1, j_1)} \end{aligned} \tag{6}$$

and

$$\begin{aligned} \mathrm{KI}(H_2) =& \mathrm{KI}(H_1) - w_C(\{i_1, j_1\}) n \\ & \times \frac{u(i_1, j_1)^t (L_1^\dagger)^2 u(i_1, j_1)}{1 + w_C(\{i_1, j_1\}) u(i_1, j_1)^t L_1^\dagger u(i_1, j_1)}. \end{aligned} \tag{7}$$

[1]We refer to these edges later as *non-cut* edges; *cut-edges* are those edges whose deletion will cause a graph to become disconnected.

Thus, Eq. 7 provides a computationally cheap way to calculate the Kirchhoff index, $\mathrm{KI}(H_2)$, of the newly formed graph H_2 after the edge $i_1 \sim j_1$ has been added to H_1. And, Eq. 6 tells us how to cheaply calculate the new Moore-Penrose pseudo-inverse, $L_2^\dagger$, of L_2 from the old value, $L_1^\dagger$.

Based on Eqs. (4)-(7), we suggest the 4-step strategy shown below in Algorithm 2 for modifying the topology of a connected weighted graph so as to reduce the corresponding Kirchhoff index, while keeping the number of edges the same, and ensuring that the maximum degree is bounded above by d_{max}.

Algorithm 2 `rewire`(H_0)

```
Output: Returns a graph $H_2$ such that $\mathrm{KI}(H_2) \leq \mathrm{KI}(H_0)$
1: For each edge $i \sim j$ of $H_0$, let $w_C(\{i,j\})$ denote the corresponding edge weight. From the set of all non–cut edges of $H_0$, select an edge $i_0 \sim j_0$ that, upon setting $i = i_0$ and $j = j_0$ (see Eq.5), minimizes the quantity
       $$\frac{w_C(\{i,j\})u(i,j)^t(L_0^\dagger)^2u(i,j)}{1 - w_C(\{i,j\})u(i,j)^t(L_0^\dagger)u(i,j)}.$$
2: Denote the Laplacian matrix of $H_0$ by $L_0$. Form the weighted graph $H_1$ from $H_0$ by deleting the edge $i_0 \sim j_0$, and update the Moore–Penrose inverse of the corresponding Laplacian matrix as $L_1^\dagger$, in accordance with Eq. 4.
3: From the set of all vertices of $H_1$ with degree at most $d_{max} - 1$ select a non–adjacent pair $i_1, j_1$ that, upon setting $i = i_1$ and $j = j_1$ (see Eq. 7), maximizes the quantity
       $$\frac{u(i,j)^t(L_1^\dagger)^2u(i,j)}{1 + w_C(\{i_0,j_0\})u(i,j)^t(L_1^\dagger)u(i,j)}.$$
4: Form $H_2$ from $H_1$ by adding to $H_1$ the edge $i_1 \sim j_1$ with weight $w_C(\{i_0,j_0\})$
5: return $H_2$
```

In this algorithm, the graph H_2 can be thought of as having been obtained from H_0 by rewiring a weighted edge; that is, by moving an edge of weight $w_C(\{i_0,j_0\})$ so that it no longer connects vertices i_0 and j_0, and instead connects vertices i_1 and j_1. We offer the following observations concerning this rewiring scheme.

First, we observe that if H_0 has s non–cut edges, then in Step 1 the edge $i_0 \sim j_0$ is identified by selecting the minimum of at most s quantities. In Step 2, it follows from Eq. 5 that among the connected weighted graphs formed by deleting a single edge from H_0, the weighed graph H_1 has the smallest Kirchhoff index. Next, we observe that if H_0 has r vertices of degree at most $d_{max} - 1$, then H_1 has at most $r + 2$ vertices of degree at most $d_{max} - 1$, and hence in Step 3 the pair i_1, j_1 is identified by selecting the maximum of at most $(r+2)(r+1)/2$ quantities. Finally in Step 4, it follows from Eq. 7 that among the weighted graphs formed by adding an edge of weight $w_C(\{i_0,j_0\})$ to H_1 so as to produce a graph of maximum degree at most d_{max}, the weighted graph H_2 has the smallest Kirchhoff index. Thus, if H_0 has s non–cut edges and r vertices of degree at most $d_{max} - 1$, then in generating H_2 from H_0 one will consider at most $s+(r+2)(r+1)/2$ different graphs (i.e., at most s at Step 1 to produce H_1, then at most $(r+2)(r+1)/2$ at Step 3 to produce H_2).

With H_0, H_1, and H_2 as in Steps 1–4 above, we have $\mathrm{KI}(H_2) \leq \mathrm{KI}(H_0)$. The reason for this inequality is straightforward. Our original graph H_0 is a member of the collection of weighted graphs formed by adding an edge of weight $w_C(\{i_0,j_0\})$ to H_1 so as to produce a graph of maximum degree at most d_{max}. Since $\mathrm{KI}(H_2)$ is smallest over all weighted graphs in that collection, it must be the case that $\mathrm{KI}(H_2) \leq \mathrm{KI}(H_0)$.

IV. Experiments and Results

A. Implementation Details

Our implementation for the rewiring procedure is based on C. For the various matrix operations required in the algorithm, we use the Linear Algebra Package (LAPACK) [24] ver. 3.4.2 through its C interface LAPACKE [25]. LAPACK uses the highly optimized Basic Linear Algebra Subprograms (BLAS) routines [26] internally.

For the sake of numerical stability of our implementation, we computed the Moore Penrose pseudo-inverse of a Laplacian matrix as $L^\dagger = (L + (1/n)J)^{-1} - (1/n)J$, where J is the all ones matrix. We found that the computation of the pseudo-inverse using SVD decomposition made our implementation susceptible to numerical errors.

In the following sections, we will refer to the Kirchhoff index based rewiring scheme as *KIR*, while the iterative co-optimization framework from Ajwani et al. [5] will be refereed to as *ICR*.

B. Experimental Framework

Now we present the experiments we conducted to compare the performance of our algorithm, KIR, with the iterative co-optimization algorithm, ICR [5]. We characterize our experiments with the following parameters: the application graph, G; the size of the topology graph, $|H|$; the maximum degree constraint, $d_{\max}$; the maximum edges constraint, $E_{\max}$; the processing speed, S_{C}, of the compute nodes in H; and the bandwidth, S_{L}, of the links in H.

For our application graphs, we took a sampling of graphs from the (a) Graph Partitioning Archive [27] maintained by Chris Walshaw and (b) 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering [28]. The particular graphs we chose are shown in Table I. These graphs represent applications domains of VLSI design, fluid dynamics, Internet structure, and social networks.

For the experiments presented in this paper, we restrict the value of $d_{\max}$ to 4. However, we change the value of $E_{\max}$ between 22 and 28 to present two different levels of communication difficulty to the application graphs. In similar vein, while we fix the value of S_{C} to an arbitrary number of 500, we vary the value of S_{L} to alter the computation to communication ratio. A high ratio implies that computation is less likely to be the bottleneck as the compute nodes can process the computation load faster than the links can move the data around.

For each graph in Table I, we conducted several trials. For each trial i we measured the time, T_i, taken to configure the network, and the goodness of the configuration, as measured by the achieved throughput, θ_i. Two trials differ from each other only in the seed for the random number generator. In each trial, we tested both KIR and ICR, and computed (a) the ratio of throughput achieved by KIR to that of ICR, denoted by

$$\Theta_i = \theta_i^{\mathrm{KIR}} / \theta_i^{\mathrm{ICR}}$$

and (b) the ratio of time for network configuration taken by ICR to that of KIR, denoted by

$$\tau_i = T_i^{\mathrm{ICR}} / T_i^{\mathrm{KIR}}.$$

We report in Table II one such experiment for $|H|$ = 16 and $E_{\max}$ = 28. We show the average values for Θ_i and τ_i, denoted as Θ and τ, respectively. We also report the average of the absolute times, i.e., T^{ICR} and T^{KIR}. It is immediately noticeable that KIR is very fast and certainly much faster than ICR. In most cases, its solution is no worse than 1% of the ICR solution but it is always faster by one to three orders of magnitude. In particular, we note four graphs for which the KIR solution is actually better; add20, memplus, as-22july06, and coAuthorsCiteseer. While add20 and memplus are derived from logic design, as-22july06 is a "symmetrized snapshot of the structure of the Internet at the level of autonomous systems" [28], and coAuthorsCiteseer is a citation network where each node is a paper and two nodes have an edge between them if the corresponding papers have a common author. Our method configures the network for as-22july06 in a little over 6 seconds while ICR takes more than 6 hours. At the same time, KIR achieves about 3 times higher throughput than ICR. We would also like to give an estimate of the absolute quality of KIR's solution for as-22july06. Although not shown in Table II, the throughput value for as-22july06 using KIR is 0.33. This value is pretty high, as it is 94 % of 0.35, a value that would be possible in a hypothetical scenario with perfect load balance (calculated by $|V_H| * S_{\mathrm{C}} / |V_G|$).

All four of these graphs stand out from (almost all of) the others in their degree distribution; they all have high max degrees, being 123, 573, 2390, and 1372, respectively. Given that the average degree for all of these graphs is much smaller than the max degree, the degree distribution likely follows a power law, with a few vertices with a very high degree and and a very high number of vertices with a small degree. Partitioning such a graph gives either small cuts with high imbalance or high cuts with low imbalance. In the KIR approach, we handle such graphs by first partitioning them with a good balance at the expense of a high cut, and then attempting to rewire the network to a topology where the high cuts can be re-routed in a way that most reduces the communication cost. At given values of link bandwidths, this approach is intuitively likely to reduce communication cost more than ICR.

Figures 1 and 2 show, for three different graphs, the effect on Θ and τ when $E_{\max}$ and S_{L} are changed independently of each other. Relaxing the $E_{\max}$ constraint understandably increases the time advantage of KIR because a larger number of topology edges considerably increases the search time for ICR (which is inherently slower than KIR). As can be expected, and seen in Figure 2, the effect on τ of increasing the S_{L} depends on the input application graph.

Table III shows, for a subset of graphs in Figure I, the values of Θ, τ, T^{ICR}, and T^{KIR} for a larger $|H|$. One can see that the two smaller input graphs of add20 and data did not have much use for a larger parallel machine. However, the larger input graph, wing_nodal, increased its throughput advantage. As seen before in Figure 2, a larger number of topology edges makes ICR relatively slower than KIR; the τ values for wing_nodal increases, from 69 to 133. At the same time, note that KIR is scalable; increasing the topology size by 2 increases the T^{KIR} by about the same factor. However, the time increase for ICR is more than a factor of 4.

TABLE I: Application graphs used in our experiments

	V	E	node degree			avg. clust. coeff.
			min	max	avg	
add20	2,395	7,462	1	123	6.23	0.63
data	2,851	15,093	3	17	10.59	0.48
3elt	4,720	13,722	3	9	5.81	0.41
whitaker3	9,800	28,989	3	8	5.92	0.41
crack	10,240	30,380	3	9	5.93	0.47
wing_nodal	10,937	75,488	5	28	13.80	0.42
fe_4elt2	11,143	32,818	3	12	5.89	0.42
memplus	17,758	54,196	1	573	6.10	0.77
as-22july06	22,963	48,436	1	2390	4.22	0.23
preferential Attachment	100,000	499,985	5	983	10.00	0.001
coAuthors Citeseer	227,320	814,134	1	1372	7.16	0.68

TABLE II: Comparison of KIR and ICR

	Θ	τ	T^{ICR} (sec)	T^{KIR} (sec)
add20	1.6	223	397	1.8
data	1	68	52	0.8
3elt	1	87	111	1.3
whitaker3	1	91	131	1.5
crack	0.99	107	253	2.4
wing_nodal	0.99	69	352	5.2
fe_4elt2	0.99	76	208	2.7
memplus	1.9	134	1749	13
as-22july06	3.1	3456	21723	6.3
preferentialAttachment	0.93	283	32197	114
coAuthorsCiteseer	2.4	104	9948	96

TABLE III: Comparison of KIR and ICR for a larger topology, $|V_H|$ = 32

	Θ	τ	T^{ICR} (sec)	T^{KIR} (sec)
add20	1.6	1400	2697	2.0
data	1	139	180	1.3
wing_nodal	1.06	133	1619	13

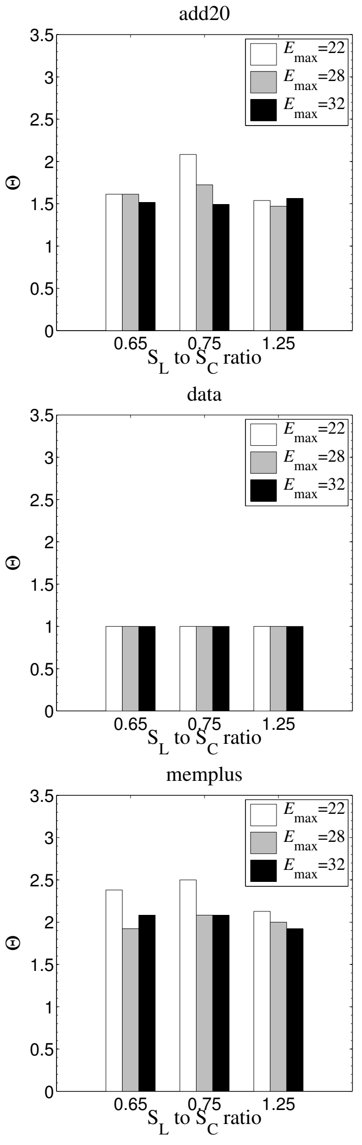

Fig. 1: The effect of increasing $E_{\max}$ and S_{L} on Θ.

V. Discussion

We consider the problem of computing a topology that optimizes the throughput of a given HPC application executing in a stream computing model, and propose a technique based on optimizing the Kirchhoff index as a proxy for maximizing the throughput.

We consider many different graph classes and show that, in most cases, the solution from our algorithm KIR is no worse than 1% of the ICR solution but that KIR is always faster by one to three orders of magnitude. For some important and widely used graph benchmarks from citation networks and Internet autonomous systems, our algorithm produces a topology that elicits a throughput that is larger by a factor of 2 to 3. For the case of as-22july06, our method configures a 16-node network about 3400 times faster ($\sim$6 seconds versus $\sim$6 hours). At the same time, KIR achieves about 3 times higher throughput than ICR (reaching 94% of a theoretical upper bound). For the coAuthorsCiteseer benchmark, our approach gives a throughput 2.4 times as large, in a time that is a 100 times faster when compared to ICR.

A major advantage of our algorithm is that it is based on matrix operations that can be easily parallelized.

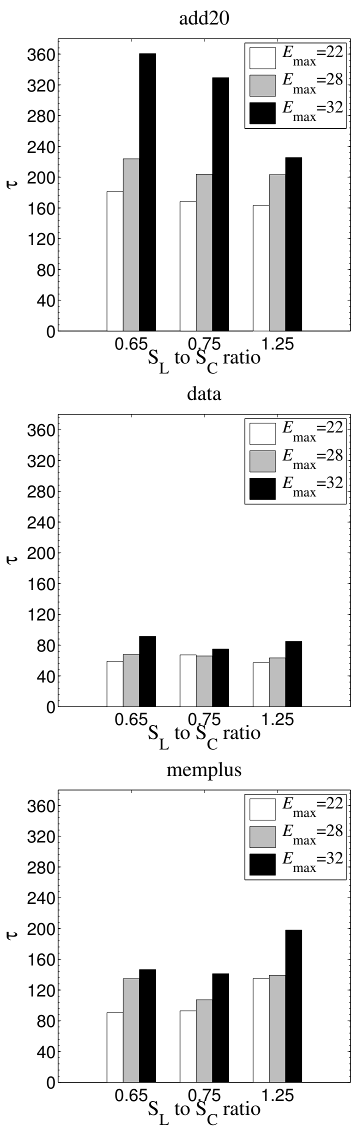

Fig. 2: The effect of increasing $E_{\max}$ and S_{L} on τ.

Furthermore, once the application graph is implicitly or explicitly mapped to an initial topology, the remaining computation does not depend on the application graph at all. This is in sharp contrast to the previous schemes based on sequential local updates (e.g., MST based routing) that are very difficult to parallelize.

We would eventually like to deploy these solutions into a real system, where the workload changes dynamically. For a technique to be deployable in such a system, it has to be extremely fast (taking a few seconds to compute the new topology). Our proposed technique is a crucial step in making this goal realizable.

In this paper, we have always chosen the initial topology as the condensed graph of the application task graph. However, our scheme can be initialized with other topologies as well. It will be instructive to find out how well our scheme works with other initial topologies such as chordal rings, 2-d tori and hypercubes. From the point of view of its deployment in a system with dynamic workload, a more tantalizing question is to use the existing topology and the existing loads on the links and compute nodes as a base case for the rewiring scheme. Similar to the graph repartitioning problems, we would then like to modify the optimization function such that the new topology is similar to the existing topology and the cost of data migration is low.

References

[1] T. Agarwal, A. Sharma, and L. V. Kalé, “Topology-aware task mapping for reducing communication contention on large parallel machines,” in *20th IEEE Int’l Conf. on Parallel and Distributed Processing (IPDPS ’06)*, 2006.

[2] I. Moulitsas and G. Karypis, “Architecture aware partitioning algorithms,” in *8th Int’l Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP ’08)*, 2008.

[3] A. Bhatele and L. V. Kalé, “Heuristic-based techniques for mapping irregular communication graphs to mesh topologies,” in *13th IEEE Int’l Conf. on High Performance Computing & Communication (HPCC ’11)*, 2011.

[4] D. Lugones, K. Katrinis, and M. Collier, “A reconfigurable optical/electrical interconnect architecture for large-scale clusters and datacenters,” in *9th ACM Conf. on Computing Frontiers*, 2012.

[5] D. Ajwani, S. Ali, and J. P. Morrison, “Graph partitioning for reconfigurable topology,” in *Proc. of the 26th IEEE International Symposium on Parallel and Distributed Processing (IPDPS ’12)*. IEEE Computer Society, 2012.

[6] K. Christodoulopoulos, K. Katrinis, M. Ruffini, and D. OMahony, “Topology configuration in hybrid EPS/OCS interconnects,” in *Int’l European Conf. on Parallel and Distributed Computing (EuroPar ’12)*, 2012.

[7] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo, “Spade: the System S declarative stream processing engine,” in *2008 ACM SIGMOD Int’l Conf. on Management of Data (SIGMOD ’08)*, 2008, pp. 1123–1134.

[8] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, “Linear road: a stream data management benchmark,” in *30th Int’l Conf. on Very Large Data Bases (VLDB ’04)*, 2004, pp. 480–491.

[9] J. Riedy, H. Meyerhenke, D. A. Bader, D. Ediger, and T. G. Mattson, “Analysis of streaming social networks and graphs on multicore architectures,” in *37th IEEE Int’l Conf. on Acoustics, Speech, and Signal Processing (ICASSP ’12)*, 2012.

[10] D. Ediger, E. J. Riedy, D. A. Bader, and H. Meyerhenke, “Tracking structure of streaming social networks,” in *5th Workshop on Multithreaded Architectures and Applications (MTAAP ’11)*.

[11] S. Kamil, A. Pinar, D. Gunter, M. Lijewski, L. Oliker, and J. Shalf, “Reconfigurable hybrid interconnection for static and dynamic scientific applications,” in *4th Int’l Conf. on Computing Frontiers (CF ’07)*, 2007.

[12] J. Shalf, S. A. Kamil, L. Oliker, and D. Skinner, “Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect,” in *ACM/IEEE Conf. on Supercomputing (SC ’05)*, 2005.

[13] K. J. Barker, A. F. Benner, R. R. Hoare, A. Hoisie, A. K. Jones, D. J. Kerbyson, D. Li, R. G. Melhem, R. Rajamony, E. Schenfeld, S. Shao, C. B. Stunkel, and P. Walker, "On the feasibility of optical circuit switching for high performance computing systems," in *Proc. of the ACM/IEEE Conference on High Performance Networking and Computing (SC).* IEEE Computer Society, 2005, p. 16.

[14] L. Schares, X. J. Zhang, R. Wagle, D. Rajan, P. Selo, S. P. Chang, J. R. Giles, K. Hildrum, D. M. Kuchta, J. L. Wolf, and E. Schenfeld, "A reconfigurable interconnect fabric with optical circuit switch and software optimizer for stream computing systems," in *Proc. of Optical Fiber Communication Conference (OFC).* San Diego, CA, 2009.

[15] D. J. Klein and M. Randić, "Resistance distance," *J. Math. Chem.*, vol. 12, pp. 81–95, 1993.

[16] A. Tizghadam and A. Leon-Garcia, "Betweenness centrality and resistance distance in communication networks," *IEEE Network*, vol. 24, no. 6, pp. 10–16, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1109/MNET.2010.5634437

[17] G. Ranjan and Z.-L. Zhang, "A geometric approach to robustness in complex networks," in *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2011.

[18] G. Georgiadis and M. Papatriantafilou, "A least-resistance path in reasoning about unstructured overlay networks," in *15th International Euro-Par Conference on Parallel Processing*, 2009, pp. 483–497. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03869-3_47

[19] A. Ghosh, S. Boyd, and A. Saberi, "Minimizing effective resistance of a graph," *SIAM Review*, vol. 50, no. 1, pp. 37–66, feb 2008. [Online]. Available: http://dx.doi.org/10.1137/050645452

[20] A. K. Chandra, P. Raghavan, W. L. Ruzzo, and R. Smolensky, "The electrical resistance of a graph captures its commute and cover times," in *21st Annual ACM Symposium on Theory of Computing (STOC '89)*, 1989, pp. 574–586. [Online]. Available: http://doi.acm.org/10.1145/73007.73062

[21] R. A. Horn and C. R. Johnson, *Matrix Analysis.* Cambridge: Cambridge University Press, 1985.

[22] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[23] F. Pellegrini, "Contributions au partitionnement de graphes parallle multi-niveaux / contributions to parallel multilevel graph partitioning," LaBRI, Université Bordeaux, Habilitation, 2009.

[24] LAPACK – Linear Algebra PACKage. [Online]. Available: http://www.netlib.org/lapack/

[25] The LAPACKE C Interface to LAPACK. [Online]. Available: http://www.netlib.org/lapack/lapacke.html

[26] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, pp. 135–151, 2001. [Online]. Available: http://www.netlib.org/blas/

[27] C. Walshaw. The Graph Partitioning Archive. [Online]. Available: http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/

[28] The 10th DIMACS Implementation Challenge – Graph Partitioning and Clustering. [Online]. Available: http://www.cc.gatech.edu/dimacs10/

[29] C. D. Meyer, "The condition of a finite markov chain and perturbation bounds for the limiting probabilities," *SIAM Journal on Algebraic and Discrete Methods*, vol. 1, pp. 273–283, 1980.

[30] A. Ben-Israel and T. N. E. Greville, *Generalized Inverses: Theory and Applications*, 2nd ed. New York: Springer – Canadian Mathematical Society Books in Mathematics, 2003.

[31] S. Kirkland, M. Neumann, and B. L. Shader, "Distances in weighted trees and group inverse of Laplacian matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, pp. 827–841, 1997.

Appendix

The following result gives a perturbation formula for the Moore–Penrose inverse of the Laplacian matrix for a weighted graph. We require this theorem and the lemma which follows it in order to justify Eqs. (4)-(7) of Sec. III-B, which are fundamental to Algorithm 2.

Theorem 1: Suppose that H and $\bar{H}$ are two connected weighted graphs on n vertices. Denote their Laplacian matrices by L and $\bar{L}$, respectively, and set $E = \bar{L} - L$. Then $I + EL^\dagger$ is invertible, and

$$\bar{L}^\dagger = L^\dagger (I + EL^\dagger)^{-1}. \tag{8}$$

In particular, if there are indices $i_1, \dots, i_k, j_1, \dots, j_k$, and scalars $\alpha_1, \dots, \alpha_k$ such that $E = \sum_{l=1}^{k} \alpha_l u(i_l, j_l) u(i_l, j_l)^t$, then letting U be the $n \times k$ matrix $U = [\ u(i_1, j_1) \mid \dots \mid u(i_k, j_k)\ ]$, and D be the $k \times k$ diagonal matrix $\mathrm{diag}(\alpha_1, \dots, \alpha_k)$, we have

$$\bar{L}^\dagger = L^\dagger - L^\dagger U D \left(I + U^t L^\dagger U D\right)^{-1} U^t L^\dagger. \tag{9}$$

Further, we also have

$$\mathrm{KI}(\bar{H}) = \mathrm{KI}(H) - n \cdot \mathrm{tr}\big((I + U^t L^\dagger U D)^{-1} U^t (L^\dagger)^2 U D\big). \tag{10}$$

Proof: The proof that $I + EL^\dagger$ is invertible is analogous to that of Theorem 3.1 of [29], and is omitted. Let J be the matrix of all ones. By considering the spectral decomposition of L, it follows from results in Sec. 4.3 of [30] that $L^\dagger = (L + \frac{1}{n}J)^{-1} - \frac{1}{n}J$; similarly, $\bar{L}^\dagger = (\bar{L} + \frac{1}{n}J)^{-1} - \frac{1}{n}J$. Note that $\bar{L} + \frac{1}{n}J = L + \frac{1}{n}J + E = (I + E(L + \frac{1}{n}J)^{-1})(L + \frac{1}{n}J) = (I + E(L^\dagger + \frac{1}{n}J))(L + \frac{1}{n}J)$. Since $EJ = 0$, we thus find that $\bar{L} + \frac{1}{n}J = (I + EL^\dagger)(L + \frac{1}{n}J)$. Consequently, $\bar{L}^\dagger = (L + \frac{1}{n}J)^{-1}(I + EL^\dagger)^{-1} = (L^\dagger + \frac{1}{n}J)(I + EL^\dagger)^{-1}$. Using the fact that $JE = 0$, we find that $J(I + EL^\dagger)^{-1} = J$, and Eq. (8) now follows.

Suppose now that $E = \sum_{l=1}^{k} \alpha_l u(i_l, j_l) u(i_l, j_l)^t$, and observe that E can be written as $E = UDU^t$. In that case, we have $I + EL^\dagger = I + UDU^t L^\dagger$. Since $(UD)(U^t L^\dagger)$ and $(U^t L^\dagger)(UD)$ have the same nonzero eigenvalues, we find that $I + U^t L^\dagger U D$ is invertible. Using the formula for the inverse of a low rank update (see Sec. 0.7.4 of [21]), we find that $(I + EL^\dagger)^{-1} = I - UD(I + UDU^t L^\dagger)^{-1} U^t L^\dagger$. Equation (9) is now readily established.

From Eq. (9) it follows that $\mathrm{KI}(\bar{H}) = \mathrm{KI}(H) - n \cdot \mathrm{tr}\big(L^\dagger U D (I + U^t L^\dagger U D)^{-1} U^t L^\dagger\big)$. For any pair of matrices A, B of orders $n \times k$ and $k \times n$, respectively, $\mathrm{tr}(AB) = \mathrm{tr}(BA)$. Applying that fact with

$A = L^\dagger UD$ and $B = \left(I + U^t L^\dagger UD\right)^{-1} U^t L^\dagger$, leads directly to Eq. (5). $\square$

For a connected graph H, an edge $i \sim j$ is a *cut edge* if $H \setminus \{i \sim j\}$ is disconnected; we call an edge of H a *non–cut* edge if it is not a cut edge. The following result deals in part with cut edges.

Lemma 2: Let H be a connected weighted graph on n vertices with Laplacian matrix L. Fix adjacent vertices i, j, and let α denote the weight of the edge $i \sim j$. We have

$$\alpha(e_i - e_j)^t L^\dagger (e_i - e_j) \leq 1, \tag{11}$$

with equality holding if and only if $i \sim j$ is a cut edge of H.

Proof: From Proposition 2.2 and Theorem 2.3 of [31], it follows that $(e_i - e_j)^t L^\dagger (e_i - e_j)$ can be written as

$$(e_i - e_j)^t L^\dagger (e_i - e_j) = \frac{\sum_{T_1, T_2} w(T_1) w(T_2)}{\sum_T w(T)}, \tag{12}$$

where the sum in the denominator is taken over all spanning trees of H, and where the sum in the numerator is taken over all pairs of vertex disjoint trees T_1, T_2 such that $T_1 \cup T_2$ is a spanning subgraph of H, and where $i \in T_1$ and $j \in T_2$. Here our notation $w(T)$ stands for the product of the weights of the edges in T, and similarly for $w(T_1), w(T_2)$.

Suppose that we have a pair of trees T_1, T_2 as above. Observe then that the graph $T = T_1 \cup T_2 \cup \{i \sim j\}$ is a spanning tree of H, and that $w(T) = w(T_1) w(T_2) \alpha$. We now find readily that $\sum_T w(T) \geq \alpha \sum_{T_1, T_2} w(T_1) w(T_2)$, and Eq. (11) follows.

Next, we consider the case of equality in Eq. (11). We find readily from the argument above that equality holds in Eq. (11) if and only if every spanning tree of H contains the edge $i \sim j$. If $i \sim j$ is a cut edge, then necessarily each spanning tree of H contains $i \sim j$, so the equality holds in Eq. (11). On the other hand, if $i \sim j$ is not a cut edge of H, it follows that H contains a spanning subgraph H' that is unicyclic, and where the edge $i \sim j$ is on the cycle in H'; we find readily that H' (and hence H) contains a spanning tree that does not involve the edge $i \sim j$, from which we deduce that equality does not hold in Eq. (11). $\square$