Toward optimal diffusion matrices *

Robert Elsässer, Burkhard Monien, Stefan Schamberger
Universität Paderborn
Department of Mathematics and Computer Science
Fürstenallee 11, D-33102 Paderborn
{elsa,bm,schaum}@uni-paderborn.de

Günther Rote
Freie Universität Berlin
Institute of Computer Science
Takustr. 9, D-14195 Berlin
rote@inf.fu-berlin.de

Abstract

Efficient load balancing algorithms are the key to many efficient parallel applications. Until now, research in this area has mainly been focusing on homogeneous schemes. However, observations show that the convergence rate of diffusion algorithms can be improved using edge weighted graphs without deteriorating the flows quality. In this paper we consider common interconnection topologies and demonstrate, how optimal edge weights can be calculated for the First and Second Order Diffusion Schemes. Using theoretical analysis and practical experiments we show, what improvements can be archived on selected networks.

Keywords. *load balancing, diffusion, eigenvalues, FOS, SOS, hypercubic networks*

1. Introduction

Load balancing is a very important prerequisite for an efficient use of parallel computers. Many parallel applications produce dynamic work load and its amount per processor often changes dramatically during run time. Therefore, to reduce the overall computation time, the total work load of the network has to be distributed evenly among all nodes while the computation proceeds. Obviously, we can ensure an overall benefit of the computation only if the balancing scheme itself is highly efficient.

One example showing the importance of an efficient load balancing scheme is parallel finite element simulation. Using meshes consisting out of several million elements representing the discretized geometric space, these are split into parts and evenly distributed among all processors. Each processor starts computing independently on its part until the next global communication step is required. Depending on the application, the mesh refines and coarsens in some areas during the computation what causes an imbalance between the processors' load and therefore delays the overall computation. In fluid dynamics for example, simulation of turbulences or shocks often depends on such refinements. In these situations, there is a need to balance the load. The application is interrupted and the at this moment static load balancing problem is solved. For a selection of applications, case studies and references on the problem of parallel finite element simulations the reader is referred to [11].

One of the most common approaches for load balancing is the 2-step model (e. g. [7]). The first step calculates a balancing flow. This flow is used in the second step, in which load elements are migrated accordingly. This paper focuses on the first step and analyzes the questions, how much load has to be migrated and where to. More formally, given a network with n nodes, each containing work load w_i, we calculate a load balancing flow over the edges of the network such that after termination of the second step, each node i has the balanced work load of $\overline{w_i} = \sum_{j=1}^{n} w_j/n$. We further assume that no load is generated or consumed during the balancing process and the structure of the network is fixed, meaning we consider a static load balancing scenario.

If the global imbalance vector $w - \overline{w}$ is known, it is possible to solve the problem by solving a linear system of equations [14]. But assuming that processors of the parallel network may only access information of their direct neighbors, load information has to be exchanged locally in iterations until a balancing flow is computed. Two sub classes of local iterative load balancing algorithms are the *diffusion* schemes [3, 5] and the *dimension exchange* schemes [5, 19]. These two classes reflect different communication abilities of the network. Diffusion algorithms assume that a processor can send and receive messages to/from all of its neighbors simultaneously, while the dimension exchange approach is more restrictive and only allows a processor communicate with one of its neighbors during each iteration. The *alternating direction* iterative scheme [10] represents a mixture of the diffusion and dimension exchange

*This work was partly supported by the German Science Foundation (DFG) project SFB-376 and by the IST Program of the EU under contract number IST-1999-14186 (ALCOM-FT).

methods. It reduces the number of iteration steps needed for networks constructed by Cartesian products of graphs. The drawback of this scheme is that the resulting flow may have load migration loops tending to infinity.

In [5], Cybenco defined the general diffusion scheme. If we denote the load after the kth iteration step on node i of the graph $G = (V, E)$ with w_i^k, then w_i^k satisfies the equation

$$w_i^k = w_i^{k-1} - \sum_{\{i,j\} \in E} \alpha_{i,j}(w_i^{k-1} - w_j^{k-1}).$$

Most of the results in this area concentrate on homogeneous schemes with the entries $\alpha_{i,j}$ being the same for any $\{i, j\} \in E$. Furthermore, there is plenty of work [5, 7, 9, 13, 16, 17] focusing on the relation between convergence rates of diffusion algorithms and the condition number of the unweighted Laplacian matrix defined in the next section. In [7] it is shown that all diffusion schemes calculate the same flow and that this flow is minimal considering the l_2-norm. In the same paper is also shown that the known diffusion schemes can be generalized for weighted graphs. Sending a higher amount of load over heavier weighted edges, the calculated flow is still minimal with respect to a weighted l_2-norm. A formal definition of this is given in the next section.

Inhomogeneous schemes can be described by edge weighted graphs. The goal is to find edge weights such that the condition number of the resulting Laplacian matrix is maximized among all Laplacians having the same communication structure, e. g. having the same zero entries. At this time very little is known about this problem. To our knowledge, [8] was the first and up to now the only paper addressing this topic. There, semi definite programming is used and it is proved that a polynomial time approximation algorithm exists to compute the optimal values. Furthermore, some examples of graph classes with optimal weights are given. Using this approach, however, considerable results can only be obtained for graphs of small cardinality.

The results of this paper are the following. First, we consider edge transitive graphs and show that for these graphs the maximal condition number of the corresponding weighted Laplacian matrix is achieved if all edges have the same weight. This result solves some open problems described in [8] with respect to optimal edge weights of Hypercubes, Cycles and the Star. Second, we consider Cayley graphs and prove, that edges generated by the same generator must be of equal weight in order to achieve the maximal condition number. Another general graph class are Cartesian products of graphs. For this class, we compute edge weights that can be used to improve the known load balancing diffusion algorithms on them. Additionally, we consider the Cube Connected Cycles and compute optimal values for the weights of its edges, maximizing the condition number of the corresponding Laplacian. Moreover, we describe how optimal edge weights can be obtained for other hypercubic networks like Cube Connected Paths, Butterfly, wrapped Butterfly and the de Bruijn. To confirm our theoretical results, we perform several experiments using different edge weight scenarios on the mentioned graph types and show the dependencies between edge weights and convergence rate.

2. Background and Definitions

Let $G = (V, E)$ be a connected, weighted undirected graph with $|V| = n$ nodes and $|E| = N$ edges. Let $c_{i,j} \in \mathbb{R}^N$ be the *weight* of edge $e_{i,j} \in E$, $w_i \in \mathbb{R}$ be the *load* of node $v_i \in V$ and $w \in \mathbb{R}^n$ be the vector of load values. Vector $\overline{w} := \frac{1}{n}(\sum_{i=1}^n w_i)(1, \dots, 1)$ denotes the vector of an average load.

Let $A \in \mathbb{R}^{n \times n}$ be the *weighted adjacency matrix* of G. As G is undirected, A is symmetric. Column/row i of A contains $c_{i,j}$ where v_j and v_i are neighbors in G. For some of our constructions we need the *Laplacian* $L \in \mathbb{Z}^{n \times n}$ of G defined as $L := D - A$, where $D \in \mathbb{N}^{n \times n}$ contains the weighted degrees as diagonal entries, e. g. $D_{i,i} = \sum_{\{v_i, v_j\} \in E} c_{i,j}$, and 0 elsewhere.

We consider the following *local iterative balancing algorithm* that requires communication with adjacent nodes only and performs the iteration

$$\begin{aligned} \forall e = \{v_i, v_j\} \in E: \; y_{i,j}^{k-1} &= \alpha c_{i,j}(w_i^{k-1} - w_j^{k-1}) \\ x_e^k &= x_e^{k-1} + \delta_{i,j} y_{i,j}^{k-1} \\ w_i^k &= w_i^{k-1} - \sum_{e=\{v_i,v_j\} \in E} y_{i,j}^{k-1} \end{aligned} \tag{1}$$

on every node $v_i \in V$. Here, $\delta_{i,j}$ represents the arbitrarily assigned edge direction, $\delta_{i,j} y_{i,j}^k$ describes the amount of load sent via edge $e = \{v_i, v_j\}$ in step k, x_e^k is the load sent via edge e added up until iteration k and w_i^k is the load of the node v_i after the k-th iteration. If a directed edge is pointing from v_i to v_j, then $\delta_{i,j} = 1$ otherwise $\delta_{i,j} = -1$. Note, that $\delta_{i,j} = -\delta_{j,i}$ and therefore $\delta_{i,j} y_{i,j}^k = \delta_{j,i} y_{j,i}^k$ for any pair of $\{v_i, v_j\} \in E$. Computing the flow x_e^k can be skipped in case of a 1-step model since there the load is immediately moved and no monitoring needs to be done. Throughout this paper however, we assume applying the 2-step model in which a balancing flow is calculated first and load is moved in a second step accordingly as already mentioned in section 1. The scheme shown in equation (1) is known as the *First Order Scheme* (FOS) and converges to the average load $\overline{w}$ [5]. It can be written in matrix notation as $w^k = M w^{k-1}$ with $M = I - \alpha L \in \mathbb{R}^{n \times n}$. M contains $\alpha c_{i,j}$ at position (i, j) for every edge $e = \{v_i, v_j\}$, $1 - \sum_{e=\{v_i,v_j\} \in E} \alpha c_{i,j}$ at diagonal entry i, and 0 elsewhere.

Now, let $\lambda_i(L)$, $1 \leq i \leq n$ be the eigenvalues of the Laplacian L in non decreasing order. It is known that $\lambda_1(L) = 0$ with eigenvector $(1, \ldots, 1)$ and $\lambda_n(L) \leq 2 \cdot deg_{max}$ with deg_{max} being the maximum weighted degree of all nodes [4]. M has the eigenvalues $\mu_i = 1 - \alpha\lambda_i$. Here, α has to be chosen such that $1 = \mu_1 \geq \mu_2 \geq \ldots \geq \mu_n > -1$. Since G is connected the first eigenvalue $\mu_1 = 1$ is simple to the eigenvector $(1, 1, \ldots, 1)$. The matrix M is called *diffusion matrix*. We denote by $\gamma = \max\{|\mu_2|, |\mu_n|\} < 1$ the second largest eigenvalue of M according to absolute values and call it the *diffusion norm* of M.

Several modifications to the *First Order Scheme* have been discussed in the past. One of them is the *Second Order Scheme* (SOS) [13] which has the form

$$w^1 = Mw^0, w^k = \beta M w^{k-1} + (1-\beta) w^{k-2},\ k = 2, 3, \ldots$$

with β being a fixed parameter, whereby fastest convergence is archived for $\beta = \frac{2}{1+\sqrt{1-\mu_2^2}}$. The Chebyshev method [7] differs from SOS only by the fact that β depends on k according to

$$\beta_1 = 1,\ \beta_2 = \frac{2}{2-\mu_2^2},\ \beta_k = \frac{4}{4-\mu_2^2\beta_{k-1}},\ k = 3, 4, \ldots$$

Generalized, a *polynomial based* load balancing scheme is any scheme for which the work load w^k in step k can be expressed in the form $w^k = p_k(M)w^0$ where $p_k \in \overline{\Pi}_k$. Here, $\overline{\Pi}_k$ denotes the set of all polynomials p of degree $\deg(p) \leq k$ satisfying the constraint $p(1) = 1$.

The convergence of a polynomial based scheme depends on whether (and how fast) the error $e^k = w^k - \overline{w}$ between the load after iteration k, $w^k = p_k(M)w^0$ and the corresponding average load $\overline{w} = \frac{1}{n}(\sum_{i=1}^n w_i^0)(1, \ldots, 1)$ converges to zero. In this work we consider a system to be ϵ-balanced after the kth iteration step iff the error $\|e^k\|_2 \leq \epsilon \cdot \|e^0\|_2$. Here, $\|e^k\|_2$ and $\|e^0\|_2$ represent the vectors e^0 resp. e^k in l_2-norm. In [13], the number of steps needed to ϵ-balance a system is analyzed and using the results of this work we can state the following lemma.

Lemma 1 *Let G be a graph and L be its Laplacian. Let $M = I - \alpha L$ be the diffusion matrix with $\alpha = \frac{2}{\lambda_2(L)+\lambda_m(L)}$ and set $\beta = \frac{2}{1+\sqrt{1-\mu_2^2}}$. Then FOS and SOS both take $\mathcal{O}(\frac{1}{\rho} \cdot \ln(1/\epsilon))$ resp. $\mathcal{O}(\frac{1}{\sqrt{\rho}} \cdot \ln(1/\epsilon))$ steps to ϵ-balance the system. Here, $\rho = \frac{\lambda_2}{\lambda_m}$ is the condition number of L.*

In this lemma α and β are chosen such that the convergence rate of FOS and SOS is maximized. We can see that the SOS converges faster than FOS by almost a quadratic factor. The Chebyshev method can be regarded to perform asymptotically identical to SOS [7]. Lemma 1 also shows the importance of the condition number of the Laplacian. Both schemes, FOS and SOS converge faster, if the condition number is higher. As mentioned in section 1, by using edge weighted graphs it is possible to increase the condition number of the Laplacian and therefore to reduce the number of steps needed to compute a balancing flow distributing the load in the network.

We concentrate now on the flow obtained by the polynomial based diffusion algorithms. Let the l_2 optimal flow be represented by the minimal flow with respect to the weighted Euclidian norm, i. e. the solution to the problem

$$\text{min!}\ \|x^k\|_2 = \sqrt{\sum_{e=1}^{N} \frac{(x_e^k)^2}{c_e}} \text{ over all balancing flows } x^k.$$

Here, $c_1, \ldots, c_N$ represent the weights assigned to the edges of the graph. Then we can state the following lemma [7].

Lemma 2 *FOS and the SOS compute an l_2-minimal balancing flow.*

In [7], lemma 2 is shown for polynomial based load balancing schemes in a general form.

3. New Results

In this section we deal with general graph classes like edge-transitive graphs, Cayley graphs and Cartesian products of graphs as well as with interconnection topologies like Grid (G), Torus (T), Cube Connected Cycles (CCC), Butterfly (BF) and de Bruijn (DB) networks. These interconnection topologies are designed to have many favorable properties for distributed computing, e.g. small vertex degree and diameter and large connectivity. In the present work, we focus our attention on computing optimal edge weights for these graphs in order to maximize the condition number of the corresponding Laplacian. First, let us concentrate on some simple graphs like Cycles, Hypercubes, complete graphs or the Star. All these graphs are edge transitive. In other words, for any pair of edges $\{u, v\}$ and $\{u', v'\}$ there is an automorphism σ such that $\sigma(u) = u'$ and $\sigma(v) = v'$. An *automorphism* of a graph is a one-to-one mapping of nodes onto nodes such that edges are mapped onto edges. To show that for all edge transitive graphs the maximal condition number is achieved if all edges have the same weight, the following lemma is useful.

Lemma 3 *Let $L_0, L_1, \ldots, L_m \in \mathbb{R}^{n\times n}$ be Laplacian matrices of weighted graphs $G_0, \ldots, G_m$, all with the same adjacency structure, and $L_0 = \frac{L_1+\ldots+L_m}{m}$. Then $\lambda_2(L_o) \geq \min\{\lambda_2(L_1), \ldots, \lambda_2(L_m)\}$ and $\lambda_n(L_0) \leq \max\{\lambda_n(L_1), \ldots, \lambda_n(L_m)\}$.*

Proof 1 *We know that* $(1,\ldots,1)$ *is an eigenvector of* L_i, $0 \le i \le m$ *to the eigenvalue* 0 *and for all Graphs* G_i *we denote the number of vertices with* n *and the number of edges with* N. *Furthermore, we denote by* $c_1^i, \ldots c_N^i$ *the weights of the edges of* G_i. *We assume w.l.o.g. that* $\lambda_2(L_1) \le \lambda_2(L_2) \le \ldots \le \lambda_2(L_m)$. *Let* $(x_1,\ldots,x_n)$ *be the eigenvector of* L_0 *to the eigenvalue* $\lambda_2(L_0)$. *Then, using the Rayleigh coefficient we obtain*

$$\begin{aligned}
\lambda_2(L_0) &= \frac{\sum_{\{u,v\}\in E_{G_i}} \left(\frac{\sum_{j=1}^m c^j_{\{u,v\}}}{m} (x_u - x_v)^2 \right)}{\sum_{u\in V_{G_i}} x_u^2} \\
&= \frac{1}{m} \sum_{j=1}^m \frac{\sum_{\{u,v\}\in E_{G_i}} c^j_{\{u,v\}} (x_u - x_v)^2}{\sum_{u\in V_{G_i}} x_u^2} \\
&\ge \frac{1}{m} \sum_{j=1}^m \left(\min_{y \perp \mathbf{1}} \frac{\sum_{\{u,v\}\in E_{G_i}} c^j_{\{u,v\}} (y_u - y_v)^2}{\sum_{u\in V_{G_i}} y_u^2} \right) \\
&= \frac{1}{m} (\lambda_2(L_1) + \ldots + \lambda_2(L_m)) \ge \lambda_2(L_1)
\end{aligned}$$

where y *and* $\mathbf{1} = (1,\ldots,1)$ *are vectors of size* n.

The second statement of the lemma can be obtained in a similar manner. □

Now, let L_1 be the Laplacian of a weighted edge symmetric graph. Applying the lemma to the family of all matrices that can be obtained from L_1 by permuting rows and columns according to some automorphism of G, we obtain a Laplacian L_0 where all edge weights are equal. Due to lemma 3, the condition number of L_0 will not be smaller than the condition number of L_1 and we can state the following theorem.

Theorem 1 *Let* G *be an unweighted, edge transitive graph. Among all weighted graphs with* G*'s adjacency structure, the condition number of the Laplacian is maximal for the one that has all edge weights set to* 1.

In the present paper we consider several graphs that can be viewed as Cayley graphs. The definition of a Cayley graph is given below.

Definition 1 *Let* G *be any abstract finite group, with identity* 1 *and let* Ω *be a set of generators for* G, *with the properties* $x \in \Omega \Rightarrow x^{-1} \in \Omega$ *and* $1 \notin \Omega$. *The Cayley graph* $\Gamma = \Gamma(G,\Omega)$ *is a simple graph with vertex set* $V_\Gamma = G$ *and edge set* $E_\Gamma = \{\{g,h\} | g^{-1}h \in \Omega\}$.

An edge {h,k} is generated by a generator $\omega \in \Omega$, iff $h^{-1}k = \omega$ or $k^{-1}h = \omega$. We now show that edges of the same generator of Ω must have the same weights in order to achieve a minimal amount of iteration steps in diffusion algorithms.

Theorem 2 *Let* Γ *be a Cayley graph and let* Ω *be the set of its generators. The condition number of the Laplacian is maximized, if for any two edges* $e = \{g,h\}$ *and* $e' = \{g',h'\}$ *generated by the same generator* $\omega \in \Omega$ *the edge weights of* e *and* e' *are equal.*

Proof 2 *For each* $g \in G$ *we may define a permutation* $\overline{g}$ *of* V_Γ *by the rule* $\overline{g}(h) = gh$, $(h \in G)$. *This is an automorphism of* Γ *[2]. If there exists an edge between* h *and* k *generated by* ω, *then there also exists an edge between* gh *and* gk *generated by* ω. *Assume* $\omega^{-1} \neq \omega$ *and let* p *be the smallest integer with the property* $\omega^p = 1$. *Then* ω *generates cycles of length* p *where each vertex has an incident edge generated by* ω *and an other incident edge generated by* ω^{-1}. *Therefore, the number of edges generated by* ω *equals the number of vertices of* Γ. *Next, we have to show that for different* g *and* g' *the edge* $\{h,k\}$ *is mapped to different edges. Assume* $\{gh, gk\} = \{g'h, g'k\}$. *Then* $gk = g'k$ *and* $gh = g'h$ *or* $gk = g'h$ *and* $gh = g'k$. *In the first case, we have a contradiction to the assumption that* $g \neq g'$, *while in the second case there is a contradiction to* $\omega^{-1} \neq \omega$. *Hence, we can use* $|G|$ *permutations to map each edge to every other edge and the theorem follows by lemma 3. If* $\omega^{-1} = \omega$, *using* $|G|$ *permutations causes each edge being mapped twice to every other edge in the graph and the theorem also follows by lemma 3.* □

As a consequence of this theorem, edges belonging to the same dimension of a Torus must have the same weight. On the other hand, a Torus can be viewed as a Cartesian product of Cycles. For a Cartesian product of two graphs G and H however, we can state the following theorem.

Theorem 3 *Let* G *and* H *be two unweighted graphs and let* $G \times H$ *be their Cartesian product. W.l.o.g, assume* $\lambda_2(G) \le \lambda_2(H)$. *The diffusion schemes on* $G \times H$ *can be improved by assigning the weight* $\frac{\lambda_2(H)}{\lambda_2(G)}$ *to the edges contained in* G *and* 1 *to the edges contained in* H.

Proof 3 *The second smallest eigenvalue of* $G \times H$ *is* $\min\{\lambda_2(G), \lambda_2(H)\}$ *and the largest eigenvalue of* $G \times H$ *has the form* $\lambda_n(G) + \lambda_n(H)$. *Let* a *be the weight of the edges of* G. *We have to maximize the function* $\rho = \min\{\frac{a\lambda_2(G)}{a\lambda_n(G)+\lambda_n(H)}, \frac{\lambda_2(H)}{a\lambda_n(G)+\lambda_n(H)}\}$. *The function* $\frac{a\lambda_2(G)}{a\lambda_n(G)+\lambda_n(H)}$ *is increasing, while* $\frac{\lambda_2(H)}{a\lambda_n(G)+\lambda_n(H)}$ *is decreasing in* a. *It follows that* $a = \frac{\lambda_2(H)}{\lambda_2(G)}$ *holds for a maximized* ρ. □

Note, that if both graphs G and H defined in theorem 3 are edge transitive graphs, assigning weight $\frac{\lambda_2(H)}{\lambda_2(G)}$ to the edges of G and 1 to the edges of H maximizes the condition number of the Laplacian matrix.

Since the eigenvalues of a Cycle of length n are $2 - 2\cos(\frac{2\pi j}{n})$, $0 \leq j < n$, we can state the following:

Corollary 1 *Let T be the d-dimensional Torus generated from the Cartesian product of d Cycles of length $n_1 \leq n_2 \leq \ldots \leq n_d$. The polynomial based diffusion algorithms have their fastest convergence rate, if the edge-weights of cycle i, $1 \leq i \leq d$ are set to $(2 - 2\cos(\frac{2\pi}{n_1}))/(2 - 2\cos(\frac{2\pi}{n_i}))$.*

Other graphs with a similar structure are d-dimensional Grids. However, these are not Cayley graphs and it is known that edges of the same dimension do not necessarily need to have the same edge weight [8]. However, considering them as Cartesian products of Paths of length $n_1 \leq n_2 \leq \ldots \leq n_d$, we can also improve the diffusion algorithms on them. Similar to corollary 1, the best results are achieved by setting the edge weight of a dimension i to $(2 - 2\cos(\frac{\pi}{n_1}))/(2 - 2\cos(\frac{\pi}{n_i}))$.

Another example showing the power of this method is the Cartesian product of a Path of length k and a complete graph of cardinality k^2. Using theorem 3 and the result of [13], we see that $\mathcal{O}(n^4 \cdot \ln(1/\epsilon))$ steps are required to ϵ-balance the system using FOS. Assigning a weight of n^2 to the edges of the Path, only $\mathcal{O}(n^2 \cdot \ln(1/\epsilon))$ steps are required.

Next, we consider the Cube Connected Cycles Network of dimension d, which will be denoted by CCC(d). The CCC(d) contains 2^d cycles of length d. We can represent each node by a pair (i, q) where i, $(0 \leq i < d)$ is the position of the node within its Cycle and q is a d-bit binary string, where q is the label of the node that corresponds to the cycle. Two nodes (i, q) and (i', q') are adjacent, iff either $q = q'$ and $i - i' = \pm 1 \bmod d$, or $i = i'$ and q differs from q' in exactly the i-th bit. Edges of the first type are called *cycle edges*, while edges of the second type are referred to as *hypercube edges*. Our objective is to determine the edge weights for which the diffusion algorithms FOS and SOS will have the fastest convergence. We use the fact that the CCC(d) is a Cayley graph [1]. It is known that the cycles in the CCC(d) are generated by one generator of the corresponding Cayley graph, while the hypercube edges are generated by some other generator. As a consequence of theorem 2, the CCC(d)'s optimal value for the condition number is obtained, iff all cycle edges are of one weight and all hypercube edges of some other weight. We normalize the weight of the Cycle edges to 1 while the weight of the Hypercube edges remains variable and is set to a. To compute the optimal value of a we need the following lemmas.

Lemma 4 *Let $C \in I\!R^{p\times p}$ and $C' = C + a \cdot J$ where $J \in I\!R^{p\times p}$ with $J_{1,1} = 1$ and all other entries of J equal 0. Then $\lambda_i(C) \leq \lambda_i(C')$ for all $a \geq 0$ and $1 \leq i \leq p$.*

The proof of this lemma immediately follows from the so called Separation Theorem [18]. In the next lemma we compute the eigenvalues of a modified Laplacian of a Cycle, where one diagonal entry contains the value $2 + 2a$ and all other diagonal entries are set to 2.

Lemma 5 *Let C be the Laplacian of an unweighted Cycle of length n and $C' = C + 2a \cdot J$ where $a > 0$ and J is defined as in lemma 4. Then $\frac{\lambda_1(C')}{2a+4}$ is maximized for $a = 2 \cdot \sqrt{2}\frac{1}{\sqrt{n}} + \mathcal{O}(\frac{1}{n})$.*

The proof of this lemma has to be omitted due to space limitations. Now, we are ready to formulate the following theorem.

Theorem 4 *The optimal value of the condition number of the Laplacian of a weighted CCC(d) is achieved for $a = 2 \cdot \sqrt{2}\frac{1}{\sqrt{d}} + \mathcal{O}(\frac{1}{d})$.*

Proof 4 *The Laplacian of the weighted CCC(d) is of the form $L_{CCC(d)} = \begin{pmatrix} C_d & D_d \\ D_d & C_d \end{pmatrix}$ where*

$$C_1 = L_{C_d} + a \cdot I_d,\ C_k = \begin{pmatrix} C_{k-1} & D_{k-1} \\ D_{k-1} & C_{k-1} \end{pmatrix} \text{ and}$$

$$D_k = I_{2^{k-1}} \otimes \begin{pmatrix} O_{d-k} & O & O \\ O^T & -a & O \\ O^T & O^T & O_{k-1} \end{pmatrix}, 1 < k \leq d.$$

Here, C_d represents the unweighted Cycle of length d, L_{C_d} its Laplacian, O denotes a matrix where all entries equal 0 and the operation "$\otimes$" is the Kronecker Product*: For $A \in I\!R^{m\times n}$, $B \in I\!R^{p\times q}$ the matrix $A \otimes B \in I\!R^{mp\times nq}$ is the matrix obtained from A by replacing every element a_{ij} by the block $a_{ij}B$. The eigenvalues of $L_{CCC(d)}$ are equal to the eigenvalues of the matrices $C_d + D_d$ and $C_d - D_d$. Applying this transformation d times, we obtain some matrices of the form $E_d - A_{C_d}$ where A_{C_d} represents the the adjacency matrix of an unweighted cycle of length d. E_d is a diagonal matrix with all diagonal entries belonging to the set $\{2, 2 + 2a\}$ and all of-diagonal entries set to 0. Lemma 4 states that the second smallest eigenvalue of the Laplacian of the weighted CCC(d) is the smallest eigenvalue of $E_d - A_{C_d}$, where E_d contains exactly one diagonal entry set to $2 + 2a$ and all other diagonal entries equal 2. Furthermore, lemma 4 also implies that the largest eigenvalue of this Laplacian is the largest eigenvalue of $E'_d - A_{C_d}$ where all diagonal entries of E'_d equal $2 + 2a$. Thus, $\rho(x)$ calculated in lemma 5 equals the condition number of the $L_{CCC(d)}$ and we obtain the theorem.* □

Analyzing the improvement of the condition number of the Laplacian by setting the weight of the hypercube edges to $\frac{2\sqrt{2}}{\sqrt{d}}$, it can be calculated that the quotient between the condition number of the weighted Laplacian and the condition number of the unweighted Laplacian converges to $3/2$. Therefore, we can save about $1/3$ of the time needed for FOS to balance the load on large topologies of this kind.

Due to space limitations, the remaining part of this section contains only a brief overview of our proves and cognitions on other network topologies.

The structure of the Cube Connected Path is similar to the one of the CCC. Its definition is identical to the Cube Connected Cycle, except that the edges between $(0, q)$ and $(d-1, q)$ are missing. Similar to the CCC, edges of the first type are called *path edges*, while edges of the second type are *hypercube edges*. In the following we denote the d-dimensional Cube Connected Path of $d \cdot 2^d$ vertices by CCP(d). The CCP is not a Cayley graph and therefore it is quite difficult to determine optimal parameters for its edges. Anyway, a similar approach can be used to improve the convergence rate, assigning weight 1 to the path edges and a to the hypercube edges. Doing this, the calculations in lemma 5 and theorem 4 provide a value of $\frac{\sqrt{2}}{\sqrt{d}} + O(1/d)$ for a. As in the case of the CCC, this value improves the condition number of the Laplacian compared to the unweighted case by a factor of approximately $3/2$ for large d.

Another common interconnection topology is the Butterfly graph, which has a similar structure to the CCP and is defined as follows. The nodes of the d-dimensional Butterfly BF(d) correspond to pairs (i, q) where i is the dimension of the node $(0 \leq i \leq d)$ and q is a d-bit binary number that denotes the row of the node. Two nodes (i, q) and (i', q') are adjacent iff $i' - i = 1$ and either $q = q'$ or q and q' differ in precisely the i'th bit. In the first case the edges are called *path edges*, while in the second case the edges are *cross edges*. These graph is neither a Cayley graph, so that using similar approaches as applied for the CCC we can only derive improvements for the convergence rate of diffusion algorithms, but we can not determine the optimal values for the edge weights. However, using weight 1 for the path edges and weight a for the cross edges it turns out, that the condition number of the Laplacian is maximized for $a = 1$.

Like in the case of CCC and CCP, we can define a Butterfly graph with wrap around edges. Two vertices (i, q) and (i', q') are adjacent, iff $i' - i = 1 \bmod d$ and either $q = q'$ or q and q' differ in precisely the i'th bit. Again, edges of the first kind are *Cycle edges* while edges of the second kind are *cross edges*. This type of graph is a Cayley graph [1], so optimal values for the edge weights can be determined. Similar to lemma 5 and theorem 4, the optimal condition number occurs when the weight of the cycle edges equals 1 and the weight of the cross edges are $a = \frac{1}{\sqrt{2}-1} + O(1/\sqrt{d})$. However, there is no significant improvement of the condition number for large d. To obtain this value for a, we have reduced the eigenvalues of the weighted Laplacian of the wrapped Butterfly to the eigenvalues of some matrices of the form $E_d - B_{C_d}$. B_{C_d} represent the adjacency matrices of some weighted Cycles of length d with edge weights from the set $\{1, 1+a\}$ while E_d is a diagonal matrix with all diagonal entries set to $2 + 2a$ and all others 0.

We finish this section by considering the de Bruijn graph DB(d). The directed de Bruijn graph contains 2^d vertices labeled from 0 to $2^d - 1$ with d-bit binary numbers, such that there is a directed edge from vertex $(i_1, \ldots, i_d)$ to $(j_1, \ldots, j_d)$ whenever $j_l = i_{l+1}$ for all $1 \leq l \leq d-1$. By replacing each directed edge by an undirected edge, we obtain the undirected de Bruijn which is regular graph of degree 4. Note, that this definition allows 2 loops at the vertices $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$ and one double edge between the vertices $(0, 1, 0, 1 \ldots)$ and $(1, 0, 1, 0, \ldots)$. To improve the diffusion algorithms on the de Bruijn network, we assign 1 to the edges formed between $(i_1, \ldots, i_d)$ and $(j_1, \ldots, j_d)$ where $j_d = i_1$, and a to the edges between $(i_1, \ldots, i_d)$ and $(j_1, \ldots, j_d)$ where $j_d = \overline{i_1}$. By using techniques from [6] and lemma 5 together with theorem 4, we get the same value for a as we obtain for the wrapped Butterfly. The reason for this is that the eigenvalues of the de Bruijn graph are reduced to the eigenvalues of matrices also contained in the set of matrices used for calculating the eigenvalues of the wrapped Butterfly.

4. Experiments

To show the effects of the approach introduced in chapter 3, we have implemented a simulation program and run several tests. Network types included are Grid (G), Torus (T), Cube Connected Cycles (CCC), Cube Connected Paths (CCP), Butterfly (BF), wrapped Butterfly and de Bruijn (DB). The program was implemented in C++, using the ARPACK++ library [15] for eigenvalue computations. While it is possible to determine eigenvalues of relatively small networks (e.g. CCC(8)) from the Laplacian itself, we are not able to do this for larger networks (e.g. CCC(16)) in a reasonable amount of time. Therefore, we determine the second smallest and largest eigenvalues of these graphs by either using explicit formulas or by reducing their calculations to the computation of eigenvalues of only parts of the original graph. A detailed description of this approach applied to the CCC can be found in section 3 and we use similar techniques for other hypercubic networks.

Prior to the first iteration of the simulation, the network's load is either distributed randomly (RS) over the network or placed onto a single node (SS), while we normalize the balanced load $(\overline{w} = 1)$. The total amount of load is therefore equal to the total number of nodes n in the graph. We apply the FOS and the SOS and keep iterating until an almost evenly distributing flow is calculated. For our tests, we define this to be archived as soon as after the t^{th} iteration $||\, w^t - \overline{w}\, ||_2$ is less than 0.01. For both diffusion schemes, we have chosen the optimal value of $\alpha = \frac{1}{\lambda_2 + \lambda_n}$, for SOS we used $\beta = \frac{2}{1+\sqrt{1-\mu_2^2}}$. The time spent on computing eigenvalues of large graphs is reduced by applying the approach described in section 3, and most of the computation time is consumed by the flow calculations.

Figures 1 through 6 show some results of our experiments. For each selection of a on the x-axis the resulting convergence rate μ_2 of FOS applied on the specific network type (left) and the number of iterations needed by SOS to compute a balancing flow (right) is shown. Note, that since the results are very similar for any combination of one of the schemes (FOS/SOS) and one of the load patterns (RS/SS), we have only included those for the SOS and SS. The results shown in figures 1 through 6 are also included in tables 1, 2 and 3, where a short overview on the simulation results with other network sizes is given.

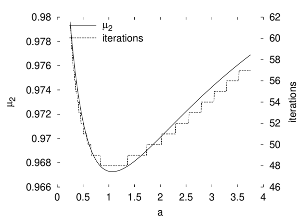

Figure 1. SOS SS CCC(8)

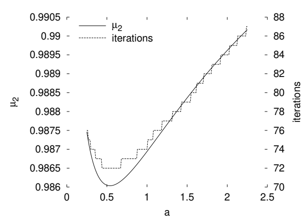

Figure 2. SOS SS CCP(8)

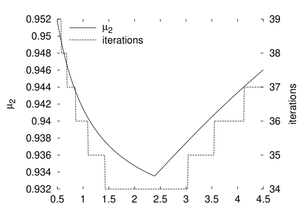

Figure 3. SOS SS wrapped BF(8)

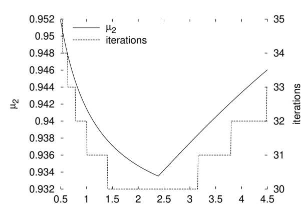

Figure 4. SOS SS DB(8)

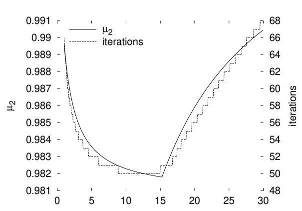

Figure 5. SOS SS G(4×16)

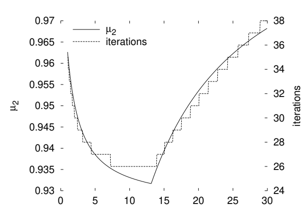

Figure 6. SOS SS T(4×16)

As we can see from figures 1 to 6, the closer a is to the optimal value a_{opt}, the smaller becomes the number of iterations needed to compute a balancing flow on all network types. First, let us study the CCC. In case of the 3 to 8-dimensional CCC we have an optimal a_{opt} greater than 1. We obtain the best improvements for the 4-dimensional CCC and the savings decrease when increasing the dimension. Considering CCC of higher dimensions than 9, we observe that the improvements increase again with larger dimension. As described in section 3, we can win using FOS at most $1/3$ for the flow computation when d tends to infinity. Similar savings can be archived for the CCP, but we obtain an a_{opt} value smaller than 1 for the 3-dimensional CCP and the improvements become higher with higher dimensions. Note however, that for the CCC a_{opt} converges to 0 for large n in contrast to wrapped BF and DB, where a_{opt} will stay about the same. A special case is the BF with its optimal value $a_{opt} = 1$. Here of course, no savings are possible at all, so we omit the corresponding graph. In the case of the wrapped BF and DB, the maximum savings are also modest, ranging from 3% to 14% and as pointed out in section 3, we cannot expect higher improvements for larger dimensions.

Second Order Scheme (SOS), Single Source (SS)

n	CCC(n) Iterations ($a =$) 1	CCC(n) Iterations ($a =$) a_{opt}	CCC(n) a_{opt}	CCC(n) Savings	CCP(n) Iterations ($a =$) 1	CCP(n) Iterations ($a =$) a_{opt}	CCP(n) a_{opt}	CCP(n) Savings
3	16	16	1.50	0%	19	19	0.88	0%
4	23	22	1.50	4%	29	28	0.77	3%
5	28	28	1.29	0%	38	38	0.69	0%
6	35	34	1.23	3%	49	48	0.63	2%
8	48	48	1.07	0%	74	72	0.54	3%
12	83	83	0.87	0%	141	134	0.43	9%
16	127	126	0.75	1%	225	211	0.37	6%

n	wrapped BF(n) Iterations ($a =$) 1	wrapped BF(n) Iterations ($a =$) a_{opt}	wrapped BF(n) a_{opt}	wrapped BF(n) Savings	DB(n) Iterations ($a =$) 1	DB(n) Iterations ($a =$) a_{opt}	DB(n) a_{opt}	DB(n) Savings
3	11	10	2.23	9%	10	9	2.23	10%
4	16	14	2.31	12%	14	12	2.31	14%
5	20	19	2.35	5%	18	16	2.35	11%
6	25	24	2.37	4%	22	21	2.37	5%
8	36	35	2.39	3%	32	30	2.39	6%
12	63	60	2.40	5%	54	52	2.40	4%
16	98	95	2.41	3%	84	81	2.41	4%

Table 1. Number of iterations needed to calculate a balancing flow for unweighted graphs ($a = 1$) and optimal weighted graphs ($a = a_{opt}$). a_{opt} is the optimal edge weight for one edge type (see Section 3) assuming the other edges have weight 1.

The results for Grid and Torus given in Figures 5 and 6 differ from the others in the way that large savings of iterations are possible, what is due to the large value of a_{opt}. As shown in Table 2 and 3, savings up to 28% can be archived. Note, that by fixing one dimension and increasing the other dimension to infinity, the optimal value of a will grow quadratically with the cardinality of the graph in the second dimension leading to improvements up to a factor of 2. We have restricted our experiments to 2-dimensional Grid and Torus, but similar results can also be obtained for higher dimensional graphs of the same type. This is an interesting result, since these networks of about the same size are widely available. The hpcline [12] operated by the PC^2

in Paderborn, for example, is designed as an 8×12 torus. Hence, these improvements are directly applicable.

Size	Second Order Scheme (SOS), Single Source (SS)							
	Grid				Torus			
	Iterations (a =)		a_{opt}	Savings	Iterations (a =)		a_{opt}	Savings
	1	a_{opt}			1	a_{opt}		
4×4	15	15	1.00	0%	9	9	1.00	0%
4×8	31	26	3.85	16%	17	14	3.40	18%
4×12	48	38	8.60	21%	26	20	7.45	23%
4×16	66	51	15.20	23%	35	26	13.10	26%
4×32	137	105	60.80	23%	73	53	52.00	27%

Table 2. Iterations needed for calculating a balancing flow for $4 \times x$ Grid and Torus.

Size	Second Order Scheme (SOS), Single Source (SS)							
	Grid				Torus			
	Iterations (a =)		a_{opt}	Savings	Iterations (a =)		a_{opt}	Savings
	1	a_{opt}			1	a_{opt}		
8×8	35	35	1.00	0%	8	8	1.00	0%
8×12	52	45	2.20	13%	27	24	2.20	11%
8×16	71	58	3.95	18%	37	30	3.85	19%
8×32	148	112	15.80	24%	76	57	15.20	25%
8×64	310	228	63.10	26%	159	115	60.80	28%

Table 3. Iterations needed for calculating a balancing flow for $8 \times x$ Grid and Torus.

5. Conclusion

We have shown that optimal edge weights can improve diffusion load balancing schemes on selected network types. Although the benefit is only modest on Hypercubic topologies, Grid and Torus based networks can highly profit by this approach. Furthermore, these results cannot only help to improve load balancing software, but also can give valuable information on how to construct communication hardware. Since the amount of load that has to be transferred over a communication edge depends on its type [7], dimensioning the bandwidth accordingly could help to improve performance.

References

[1] F. Annexstein, M. Baumslag, and A.L.Rosenberg. Group action graphs and parallel architectures. *SIAM J. Computing*, 19:544–569, 1990.

[2] N. Biggs. *Algebraic Graph Theory*. Cambridge University Press, second edition, 1993.

[3] J.E. Boillat. Load balancing and poisson equation in a graph. *Concurrency - Practice & Experience*, 2:289–313, 1990.

[4] D.M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs*. Johann Ambrosius Barth, 3rd edition, 1995.

[5] G. Cybenko. Load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[6] C. Delorme and J.P. Tillich. The spectrum of de bruijn and kautz graphs. *European Journal of Combinatorics*, 19:307–319, 1998.

[7] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.

[8] R. Diekmann, S. Muthukrishnan, and M.V. Nayakkankuppam. Engineering diffusive load balancing algorithms using experiments. In G. Bilardi et al., editor, *IRREGULAR'97*, LNCS 1253, pages 111–122, 1997.

[9] R. Diekmann, F. Schlimbach, and C. Walshaw. Quality balancing for parallel adaptive fem. In *IRREGULAR'98*, Springer LNCS, 1998.

[10] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. Amestoy et al., editor, *EuroPar'99*, LNCS 1685, pages 280–290, 1999.

[11] G. Fox, R. Williams, and P. Messina. *Parallel Computing Works!* Morgan Kaufmann, 1994.

[12] Fujitsu-Siemens. hpcline at the pc^2. http://www.uni-paderborn.de/pc2/services/systems/psc/.

[13] B. Ghosh, S. Muthukrishnan, and M.H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *SPAA*, pages 72–81, 1996.

[14] Y.F. Hu, R.J. Blake, and D.R. Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice & Experience*, 10(6):467–483, 1998.

[15] R. B. Lehoucq, D. C. Sorensen, and C. Yang. Arpack users' guide: Solution of large scale eigenvalue problems with implicitly restarted arnoldi methods. Technical report, Computational and Applied Mathematics, Rice University, October 1997. Technical Report from http://www.caam.rice.edu/software/ARPACK/.

[16] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel diffusion schemes for repartitioning of adaptive meshes. In *EuroPar'97*, Springer, LNCS, 1997.

[17] C. Walshaw, M. Cross, and M. Everett. Dynamic load-balancing for parallel adaptive unstructured meshes. In *Proc. 8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[18] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[19] C. Xu and F.C.M. Lau. *Load Balancing in Parallel Computers*. Kluwer, 1997.