Towards Optimal Load Balancing Topologies*

Thomas Decker, Burkhard Monien, and Robert Preis

Department of Mathematics and Computer Science, University of Paderborn, Germany
{decker,bm,robsy}@uni-paderborn.de

Abstract. Many load balancing algorithms balance the load according to a certain topology. Its choice can significantly influence the performance of the algorithm. We consider a two phase balancing model. The first phase calculates a balancing flow with respect to a topology by applying a diffusion scheme. The second phase migrates the load according to the balancing flow. The cost functions of the phases depend on various properties of the topology; for the first phase these are the maximum node degree and the number of eigenvalues of the network topology, for the second phase these are a small flow volume and a small diameter of the topology. We compare and propose various network topologies with respect to these properties. Experiments on a Cray T3E and on a cluster of PCs confirm our cost functions for both balancing phases.

1 Introduction

Load balancing algorithms are typically based on a fixed topology which defines the load balancing partners in the system. Only processors that are neighbors in the topology exchange both information and load items during the load balancing process.

There are three main aspects that influence the choice of a load balancing topology. Firstly, their fitness for the application. Topologies can reveal data-dependencies between tasks. This allows the load balancing algorithm to preserve the locality of the tasks. Secondly, their fitness for the hardware. Topologies can reveal the structure of the communication network of the parallel machine and thus reduce the total network load generated by the load balancing algorithm. Thirdly, their fitness for the load balancing algorithm. In order to speed up the load balancing phases of the application, a topology can be chosen that allows a fast convergence of the load balancing algorithm.

We focus on the third aspect. Absence of data dependencies and negligible influence of the interconnection topology are assumptions that are valid for a wide range of applications and architectures. Moreover, we consider a static load balancing problem. Thus, during the load balancing phase, no load is processed or generated.

In the case of load distributions that do not require global movements, load balancing algorithms that involve only nearest-neighbor communication are potentially superior to algorithms that involve global communication [15]. The conceptually simplest local iterative load balancing algorithm is the *First Order Diffusion* scheme (FOS) introduced by Cybenko [4]. All commonly used diffusion schemes generate the unique

* Supported by German Science Foundation (DFG) Project SFB-376, EU ESPRIT LTR Project 20244 (ALCOM-IT) and by EU TMR-grant ERB-FMGE-CT95-0052 (ICARUS 2).

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 277–287, 2000.

l_2-minimal flow [6]. However, if diffusive schemes are used for direct load movement, they typically shift much more load items than necessary [6]. Therefore, the load balancing process is split into two phases. The first phase operates on a virtual load measure and, since this phase actually computes a network flow, we call it *flow computation phase*. In the *migration phase*, the real load items are distributed according to the computed flow.

Algorithms for the flow computation phase have been studied extensively. Many of them are local iterative schemes based on dimension exchange or diffusion [6, 7, 9, 12, 11, 18]. We consider the optimal diffusion scheme OPT [7] which computes the l_2-minimal flow. OPT is very simple and does only need $m - 1$ iterations where m is the number of distinct eigenvalues of the Laplacian matrix. As an extension, we introduce a *multiple diffusion scheme* (MD) which can be applied to topologies that are cartesian products of other graphs. MD applies the OPT scheme to each of the factor graphs one by one. Although the flow of MD is not necessarily l_2-minimal, the number of iterations can be decreased dramatically. Overall, we show that the time requirement of the flow computation phase depends on the maximum node degree and on the number of eigenvalues of the network.

During the migration phase we make use of the *Proportional Parallel Greedy*-heuristic (PPG) [6] that sends load preferably to that neighbor node that represents the largest sink. We show that both a small flow volume and a small diameter of the graph are important for this phase.

Overall, there is a demand for networks with small degrees, small numbers of distinct eigenvalues and small diameters for any value of node numbers. Furthermore, the networks should have the property that the applied balancing scheme can compute a flow with a small volume in the network. We discuss several graph classes such as cycles, hypercubes, cliques, tori or Cages. As we will see, the hypercubes have reasonably small values in all our measures. The hypercubes can be represented as many different cartesian products. By using the MD scheme with different numbers of dimension one can reduce the number of eigenvalues and establish a tradeoff between the time for calculating the balancing flow and the volume of the flow for the hypercube. There are special graphs for certain numbers of nodes such as the Petersen and the Hoffmann-Singleton [1] graphs with very low values for our measures, but they do not belong to a scalable graph class with similarly small values. Thus, our investigations are a first step to develop optimal load balancing topologies.

We integrated the algorithms into the load balancing library VDS [5]. We consider two platforms in order to cover the characteristics of both closely and loosely coupled systems. The Cray T3E captures the typical properties of massively parallel systems since its MPI implementation offers short message startup-times and small latency. Secondly, we consider a network of workstations (NOW) consisting of 96 Pentium II processors connected by Ethernet. The communication is based on PVM. The per-word transfer time of the NOW is about 15 times slower than for the Cray.

The following section defines the balancing flow and discusses the load balancing schemes OPT and MD. In Section 3 we develop a cost function for the flow calculation phase and recommend several topologies. The cost function of the migration phase together with several experiments on different load scenarios is discussed in Section 4.

2 Definitions and Background

Balancing Flow We represent the topology of the network by a connected, undirected graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = N$ edges. Let $w_i \in \mathbb{R}$ be the *load* of node $v_i \in V$ with vector $w = (w_1, \ldots, w_n)$. Denote with $\tilde{w} := \frac{1}{n}\sum_{i=1}^{n} w_i$ the average load and vector $\overline{w} := (\tilde{w}, \ldots, \tilde{w})$. Let $\boldsymbol{E}$ be the same edge set as E, but with an arbitrary fixed direction to represent the direction of the flow. Let x_e represent a flow on $e \in \boldsymbol{E}$ with $x \in \mathbb{R}^N$. We call x a *balancing flow* for a load w, if for all $v_i \in V$ it is $w_i + \sum_{e=(v_j,v_i)\in \boldsymbol{E}} x_e - \sum_{e=(v_i,v_j)\in \boldsymbol{E}} x_e = \tilde{w}$. Consider the quality measures

- $l_1(x) = \|x\|_1 = \sum_{e\in \boldsymbol{E}} |x_e|$, the total costs of communication.
- $l_2(x) = \|x\|_2 = \sqrt{\sum_{e\in \boldsymbol{E}} x_e^2}$. This is minimized by common diffusion algorithms.
- $l_\infty(x) = \|x\|_\infty = \max_{e\in \boldsymbol{E}} |x_e|$, the max. communication between any two nodes.
- node-flow $f(x) = \max_{v_i\in V} f(v_i)$ with $f(v_i) = \sum_{e=\{v_i,v_j\}\in E} |x_e|$. This is the max. communication at any node (in- and out-going flow, see also [16]).

We define the *migration graph* $MG(x) = (V, \tilde{E})$ with directed edges by $(v_i, v_j) \in \tilde{E}$ iff $(v_i, v_j) \in \boldsymbol{E} \wedge x_{(v_i,v_j)} > 0$ or $(v_j, v_i) \in \boldsymbol{E} \wedge x_{(v_j,v_i)} < 0$. The *migration flow* $\tilde{x} \in \mathbb{R}^N$ on $\tilde{E}$ is defined by setting $\tilde{x}_{(v_i,v_j)} = x_{(v_i,v_j)}$ if $(v_i, v_j) \in \boldsymbol{E}$, and $\tilde{x}_{(v_i,v_j)} = -x_{(v_j,v_i)}$ otherwise. It is obvious that the migration graphs of l_1- and l_2-minimal flows are directed acyclic graphs (DAGs).

The balancing flow may also be defined in matrix notation. Let $Z \in \{-1, 0, 1\}^{n\times N}$ be the node-edge *incidence matrix* of G. Every column has two non-zero entries 1 and -1 for the two nodes incident to the edge. The signs define the direction of the edges according to $\boldsymbol{E}$. x is a balancing flow iff $Zx = w - \overline{w}$. Let $A \in \{0, 1\}^{n\times n}$ be the symmetric *adjacency matrix* of G. The *Laplacian* $L \in \mathbb{Z}^{n\times n}$ of G is defined as $L := C - A$, with $C \in \mathbb{N}^{n\times n}$ containing the degrees as diagonal entries.

Overview of Load Balancing Schemes In [12], the set of linear equations $Ly = w - \overline{w}$ is directly solved for y and the balancing flow is then derived by $x = Z^T y$. Executing this globally on one node requires a large amount of communication for gathering and broadcasting. The system $Zx = w - \overline{w}$ can also be calculated in parallel by using a standard conjugate gradient algorithm [12]. Besides, a matrix-vector multiplication (can be done locally in some iterations) requires three global summations of scalars.

A different way is to iteratively balance the load of a node with its neighbors locally until the whole network is globally balanced. There are two major approaches. In the *diffusion* schemes [4] each node sends and receives the current load of all of its neighbors simultaneously. All commonly used diffusion schemes calculate the unique l_2-minimal flow [6]. Thus, the migration graph of a diffusion flow is a DAG. In the *dimension exchange* (DE) schemes [4, 18] each node balances its load with one neighbor after another. The flow of a DE scheme is not necessarily l_2-minimal.

Some further models can be used for networks that are a cartesian product of graphs such as tori or hypercubes. Here, $G = G_1 \times G_2$ with node set $V(G) = V(G_1) \times V(G_2)$ and edge set $E = \{((u_i, u_j), (v_i, v_j)); u_i = v_i \wedge (u_j, v_j) \in E(G_2)$ or $u_j = v_j \wedge (u_i, v_i) \in E(G_1)\}$. Cartesian products of lines and paths were previously discussed in [18]. The *Alternating Direction Iterative* (ADI) scheme [7] is a combination of diffusion

and dimension exchange for cartesian products. It is a common method of solving linear systems [17] that is applied to the load balancing problem. In each iteration a node first communicates with its neighbors according to G_1 and then with its neighbors according to G_2. Unfortunately, the resulting flow may not be a DAG, leading to fairly high flow values. In the *multi diffusion* (MD) scheme a node exchanges the load iteratively with its neighbors in dimension 1 until the subgraph G_1 to which it belongs is balanced. Then, it balances along dimension 2 until the whole network is balanced. For dimension 1 there are $|V(G_2)|$ and for dimension 2 there are $|V(G_1)|$ independent load balancing tasks. We use a standard diffusion scheme for G_1 and G_2 for these tasks.

Theorem 1. *Let graph G be a cartesian product $G = G_1 \times G_2$ and let x be a balancing flow of G calculated with the MD scheme using a sub-scheme which calculates a migration DAG for both directions. Then, the migration graph $MG(x)$ is a DAG, too.*

Proof. Assume it has a cycle. At least one edge of the cycle has to be in dimension 2, because dimension 1 is acyclic, due to the assumption. After having finished the balancing in dimension 1, all $|V(G_2)|$ subgraphs that are isomorphic to G_1 are balanced within themselves. Thus, the load balancing flow in dimension 2 has to have a cycle. This is false, because all $|V(G_1)|$ independent balancing tasks in dimension 2 are acyclic and, because of the same load distribution, all of them have the same flow direction. □

The MD scheme can easily be generalized for d-dimensional cartesian products $G_1 \times \ldots \times G_d$. Theorem 1 can be extended such that MD guarantees a DAG for any d.

Diffusive Load Balancing Schemes These schemes perform iterations with communication between adjacent nodes. In the first-order-scheme (FOS) [4, 9] node $v_i \in V$ performs $w_i^k = w_i^{k-1} - \sum_{\{v_i,v_j\}\in E} \alpha(w_i^{k-1} - w_j^{k-1})$ with flow $x^k_{e=\{v_i,v_j\}} = x_e^{k-1} + \alpha(w_i^{k-1} - w_j^{k-1})$. Here, w_i^k is the load of node i after k iterations, and x_e^k is the total load sent via edge e until iteration k. For an appropriate choice of α, FOS converges to the average load $\overline{w}$ [4]. The resulting flow is l_2-minimal [6].

FOS can be transcribed as $w^k = Mw^{k-1}$ with $M = I - \alpha L \in \mathbb{R}^{n\times n}$. Let $0 = \lambda_1 < \ldots < \lambda_m$ be the m distinct eigenvalues of the Laplacian L. It is known that $\lambda_1 = 0$ is simple (for a connected graph G) with eigenvector $(1, \ldots, 1)$ [1]. Furthermore, the minimum number of iterations for FOS can be obtained by $\alpha = \alpha_{opt} = \frac{2}{\lambda_2+\lambda_m}$. The main disadvantage of FOS is its slow convergence, even when α_{opt} is used. Furthermore, FOS never achieves the exactly balanced load of $\overline{w}$. FOS has to be terminated when the imbalance is small enough. This process requires a global termination detection.

A numerically stable optimal scheme OPS was introduced in [6]. OPS only needs $m - 1$ iterations and balances the load vector to exactly $\overline{w}$. It is also a diffusion scheme and calculates the l_2-minimal flow. Another optimal scheme OPT [7] with the same convergence properties was derived from OPS. Although OPT might trap in a numerical unstable situation, it was shown of how to avoid them. The difference between OPT and FOS iterations is the fact that the parameter α varies for each iteration. For iteration k we choose $\alpha = \alpha_k = \frac{1}{\lambda_k}$ with λ_k, $2 \le k \le m$ being the distinct non-zero eigenvalues of L. For each iteration a different eigenvalue of L is used. The order of the eigenvalues is arbitrary. We choose an order that achieves numerical stability [7]. The main disadvantage of OPT is the fact that all eigenvalues of the graph are required. Although the

Table 1. Graphs with $m-1=D$.

Graph	$\lvert V\rvert$	degrees	Spectrum of L_G	$m-1=D$
Path(n)	n	$1,2$	$2-2\cdot\cos(\frac{\pi}{n}j)\ \ j=0,\ldots,n-1$	$n-1$
Cycle(n)	n	2	$2-2\cos\left(\frac{2\pi}{n}j\right)\ \ j=0,\ldots,n-1$	$\lfloor\frac{n}{2}\rfloor$
Star(n)	n	$1,n-1$	$0,1,n$	2
complete k-partite(n)	n	$n-\frac{n}{k}$	$0, deg^{[n-k]}, n^{[k-1]}$	2
Clique(n)	n	$n-1$	$0, n^{[n-1]}$	1
Hyp(d)	2^d	d	$2j\ \ j=0,\ldots,d$	d
Lattice(k,d)	k^d	$d(k-1)$	$jk\ \ j=0,\ldots,d$	d
Petersen/$deg3D2$/MCage(3,5)	10	3	0,2,5	2
Hoff.-Sing./$deg7D2$/MCage(7,5)	50	7	0,5,10	2
MCage($d,6$), $d-1$ prime-power	$\frac{2(d-1)^3-2}{d-2}$	d	$0, d\pm\sqrt{d-1}, 2d$	3
MCage($d,8$), $d-1$ prime-power	$\frac{2(d-1)^4-2}{d-2}$	d	$0, d\pm\sqrt{2(d-1)}, d, 2d$	4
MCage($d,12$), $d-1$ prime-power	$\frac{2(d-1)^6-2}{d-2}$	d	$0, d\pm\sqrt{3(d-1)}, d\pm\sqrt{d-1}, d, 2d$	6

calculation of the eigenvalues can be time-consuming for large graphs, they can easily be computed for small graphs. Besides, they do only have to be calculated once for each graph and can be applied for any load situation. Furthermore, they are known for many classes of graphs that often occur as processor networks (see e. g. [3, 12]). Another optimal scheme is presented in [11].

As a byproduct, the optimal schemes lead to $D \leq m-1$ with D being the diameter of the graph. This fact has already been proven in a different way in [1, 3].

3 Flow Calculation

In this section, we discuss the first step of the balancing process, i. e. the calculation of the balancing flow. For cartesian products of graphs we also use the scheme MD in combination with OPT for each dimension. In each iteration of OPT a node has to send the information about its current load to all neighbors and it has to receive their load. Thus, its number of send/receive-operations per iteration is equal to its degree. We mainly discuss graphs $G=(V,E)$ with regular degree $deg(G)$. As discussed above, the number of iterations of OPT is $m-1$ with m being the number of non-zero distinct eigenvalues of the Laplacian of G. Thus, every node executes a total of $(m-1)\cdot deg(G)$ send- or receive-operations. The MD scheme for the cartesian product $G=G_1\times\ldots\times G_d$ of graphs G_i involves $(m_i-1)\cdot deg(G_i)$ operations in dimension i for every node. The total number of send- or receive-operations for each node is $M(G,d)=\sum_{i=1}^{d}(m_i-1)\cdot deg(G_i)$. Thus, networks with small deg and m are desirable. As stated in section 2, it holds $m-1\geq D$ with D being the diameter of the graph. Table 1 lists some graphs with $m-1=D$ (see also [1, 3, 12]).

Paths, cycles, stars, complete k-partite graphs and cliques either have a high maximum degree or a high number of distinct eigenvalues. Hypercubes have logarithmic values for both measures. The Lattice graphs are hypercubes over an alphabet of order k. Unfortunately, there are only hypercubes and Lattice graphs for some node numbers.

A well known problem is the construction of (deg,D)-graphs which are the graphs with the largest number of nodes for a given degree deg and diameter D [2]. A (deg,D)-graph has at most $\frac{deg(deg-1)^D-2}{deg-2}$ nodes (*Moore-Bound*). This bound is attained for

Graph G	dim. d	$deg(G_1)$	$m_1 - 1$	$M(G,d)$
Cycle(64)	1	2	32	64
Clique(64)	1	63	1	63
Torus(8,8)	1	4	12	48
Cycle(8)^2 (MD)	2	2	4	16
Hyp(6)	1	6	6	36
Hyp(2)^3 (MD)	3	2	2	12
Hyp(1)^6 (MD)	6	1	1	6
Lattice(4,3)	1	9	3	27
Lattice(4,3) (MD)	3	3	3	9
Butterfly(4)	1	4	10	40
DeBruijn(6)	1	4	17	68
Knödel(64)	1	6	33	198
Cage(6,6)	1	6	3	18
Knödel(62)	1	5	7	35

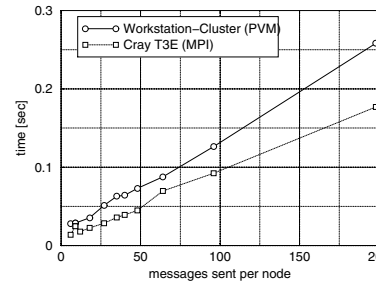


Fig. 1. Left: characteristics of graphs. The cartesian products have the identical subgraph G_1 with $M(G,d) = d(m_1 - 1)deg(G_1)$. Right: flow computation costs with respect to the number of messages sent per node. The times measured for the Cray T3E are scaled by a factor of 10.

cliques. For $d \geq 3$ and $D \geq 2$ it is only attained if $D = 2$ and $deg = 3$ (Petersen graph), $deg = 7$ (Hoffmann-Singleton graph, [1]) and (perhaps) $deg = 57$. There is no scalable construction for (deg, D)-graphs. Our experiments revealed that among the largest known (deg, D)-graphs only the stated graphs have a fairly small value of m. Related graphs are (deg, g)-*Cages* which are the smallest graphs with degree deg and shortest cycle (*girth*) g [1]. A Cage has at least $\frac{deg(deg-1)^{(g-1)/2}-2}{deg-2}$ nodes for an odd g and at least $\frac{2(deg-1)^{g/2}-2}{deg-2}$ nodes for an even g. A Cage with a size equal to this bound is called *minimum Cage* MCage(deg,g). There are only few graphs of this kind, such as the mentioned (deg, D)-graphs that achieve the Moore bound, cycles and complete bipartite graphs. Further MCages(deg,g) do only exist for $g = 6, 8, 12$ and $deg - 1$ prime power. For MCages it holds $m - 1 = D = \lfloor \frac{g}{2} \rfloor$. The sizes of (deg, g)-Cages are only known for a limited number of values of deg and g [14]. Our experiments revealed that among the known Cages only the MCages have fairly small values of m.

The Knödel graphs [13] can be constructed for any number of nodes. Their degree is $\lfloor \log_2 n \rfloor$ and their diameter is at most $\lceil \log_2 n \rceil$. For $n = 2^i$ with some $i \in \mathbf{N}$, their diameter is $\lceil \frac{\log_2 n+2}{2} \rceil$ [8]. This is an improvement over the hypercube. It has been shown that Knödel graphs of size $2^i - 2$ with $i > 1$ are edge-symmetric [10]. Our experiments show that only these Knödel graphs have a fairly low number of eigenvalues.

Fig. 1(left) presents a comparison of several graphs with 64 nodes (except for Cage(6,6) and Knödel(62) with 62 nodes). We also considered the 8×8 torus, the butterfly graph of dimension 4 and the DeBruijn graph of dimension 6. Several graphs can be expressed as a cartesian product such as Hyp(6) = Hyp(2)^3 = Hyp(1)^6. Here, MD can be performed in 1, 2 or 3 dimensions with different values $M(G,d)$. $M(G,d)$ decreases with increasing d, but the amount of flow increases (Sec. 4). For $d = 1$ the value $M(G,d)$ is higher, but the diffusion scheme calculates the l_2-minimal flow.

Fig. 1(right) presents the relation between $M(G,d)$ and the execution times of the flow computation for various graphs. The linear progress shows that $M(G,d)$ is a prac-

ticable cost function and, by applying linear regression, we obtain $T_{flow}(Cluster) \approx 1.2\text{ms} \cdot M(G,d) + 17\text{ms}$ and $T_{flow}(CrayT3E) \approx 0.084\text{ms} \cdot M(G,d) + 0.98\text{ms}$.

4 Flow Migration

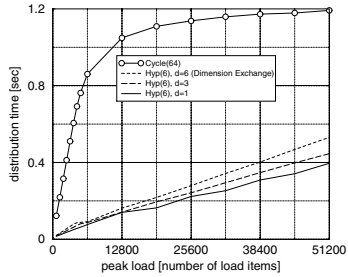
Once the flow is computed, the nodes start with the migration phase. Let $v \in V$ be a node with neighbors $v_1, \ldots, v_k$ in the migration graph. The node has to send a load of $\tilde{x}_{(v,v_i)}$ to neighbor v_i. Since a node v might need to send this load in several steps, it keeps track of the remaining number of load items out_{v,v_i} which have to be sent to v_i (out_{v,v_i} is initialized with $\tilde{x}_{(v,v_i)}$ rounded to the closest integer value). Each time node v sends load items to v_i, it updates out_{v,v_i}. This distribution algorithm is executed once the flow is computed and again whenever new load items arrive. The load migration phase terminates as soon as $out_{v,w} = 0$ for all $(v,\, w) \in \tilde{E}$. This condition is met after a finite number of steps (since the flows generated by MD are acyclic).

We introduce logical communication-rounds in order to determine the duration of the load migration phase. Let $r(v)$ denote the round-number of a node $v \in V$. Initially, $r(v) = 0$ for all $v \in V$. Each message is tagged with the round-number of the sending node plus one. Whenever a node v receives a message, its round-number $r(v)$ is set to the maximum of $r(v)$, and the round-number of the incoming message. We define the maximal round number $r := \max_{v \in V} r(v)$. Denote with $out_v := \sum_{(v,w) \in \tilde{E}} out_{v,w}$ the *out-going* flow of v. If out_v is not larger than the initial load w_v, the total out-going flow can be saturated in one step. A *conflict* occurs whenever out_v is larger than the current load of v. In this case, we have to decide on how much load to send to each of the neighbors. It is known that r is bounded by $\mathrm{O}(\sqrt{n})$ for all local greedy algorithms which always migrate all available load items [6]. The PPG-heuristic belongs to this class of algorithms. PPG moves the portion $out_{v,v_i}/out_v$ of the current load to node v_i. Consequently, the load is preferably moved in the direction of the largest sink.

We assume that the time needed to send or receive a message of size s is given by $t_o + s\, t_w$. Parameter t_o models a constant communication overhead (such as the startup time of the communication network). Parameter t_w models the per-word transfer time. We can bound the duration T_m of the migration phase as $T_m \leq (r \deg(G))\ t_o + f(x)\, t_w$. Both values r and $f(x)$ depend on the balancing flow which again depends on the topology and the balancing scheme.

In the following, we use two load scenarios and illustrate the cost function for several topologies. In the *peak*-scenario, we set w_1 to a positive value and $w_i = 0$, $2 \leq i \leq n$. Thus, the peak-scenario models applications which incorporate only few nodes with the majority of the load. In order to model applications that have a more balanced generation pattern, we use the *random*-scenario and draw the entries of w from a uniform random distribution.

The peak-scenario. Fig. 2(left) presents the timings for the migration phase that were obtained on the Cray T3E. Up to 51200 load elements were generated on node 0 (each load object consists of 150 bytes). The diagram shows five topologies with 64 nodes each, namely the cycle, three schemes for the hypercube, and the clique. In all cases, r is equal to the diameter of the network. Interestingly, this is not necessarily the case [6]. Moreover, for a fixed initial load, the node flow $f(x)$ is always the same. The



G	r	$l_2(x)$	migration time	
			Cray	NOW
Cycle(64)	32	35638	1.19	6.9
Hyp(6)	6	22755	0.40	2.1
Hyp(2)^3	6	32790	0.47	2.1
Hyp(1)^6	6	35919	0.53	2.1
Clique(64)	1	6350	0.38	2.0

Fig. 2. Duration of migration phase for various topologies and peak load. The diagram on the left displays the dependency of the duration of the migration phase on the extent of the peak load. The experiments were conducted on a Cray T3E. The table on the right relates the migration time to the diameter of the topologies and the l_2-norm of the flow. The times are given in seconds.

long duration of the migration time in case of the cycle topology is due to the large diameter. It can be reduced significantly, if we choose networks with smaller diameter.

The three timings for the hypercube refer to the different schemes, namely diffusion, multi diffusion (MD) with respect to Hyp$(2)^3$ and Hyp$(1)^6$ (dimension exchange). They differ in the way they distribute the load along the edges. The number of edges that take part in the process depends on the number of dimensions of the MD scheme. For the original diffusion, all edges are used. If we apply the MD scheme, only some of the edges are used. For example, the dimension exchange method does only use 63 edges which is about one third of all edges. The unbalanced distribution of the communication load generates a critical path that is responsible for the duration of the migration phase.

The $l2$-norm of the flow is a measure for the balance of the communication load. Fig. 2 (right) lists the diameter, the $l2$-norm of the flow, and the migration time for an initial load of 51200 items. It reveals that T_m is mainly influenced by the diameter.

The random-scenario. The distance between high- and low-loaded processors involved by the random-scenario is much smaller than the distance involved by the peak-scenario. Thus, the number of migration rounds is typically much smaller than the diameter. Fig. 3 (left) lists the average number $\overline{r}$ of migration rounds that are needed for various topologies. Except for the clique and the cycle, about two migration rounds suffice to balance the load. Thus, in contrast to the peak-scenario, r is an unimportant parameter. The dominating parameter is the amount of data the nodes have to communicate, i. e. the maximum node flow $f(x)$.

Fig. 3 (right) presents a strong correlation between the node flow and the migration time. We cannot expect a perfect correlation, since the node flow does only represent the second term of the cost function T_m. The first term depends on r and the degree of the network. The node flow is too pessimistic for the cycle and too optimistic for the clique. One reason for these deviations are the extreme degrees of these graphs.

The network flows listed in Fig. 3 (left) reveal that the node-flow depends on the topology and on the balancing scheme. The MD schemes produce a larger node flow than the diffusion scheme because MD balances the dimensions of the topology one

G	$\overline{\tau}$	$f(x)$	$l_2(x)$	T_m Cray [s]	T_m NOW [s]	$T_m/f(x)$ Cray [ms]	$T_m/f(x)$ NOW [ms]	$T_m/l_2(x)$ Cray [μs]	$T_m/l_2(x)$ NOW [ms]
Cycle(64)	**3.3**	**4067**	**7591**	**0.076**	**0.54**	**0.018**	**0.13**	**9.6**	**0.07**
Clique(64)	**1.0**	**875**	**478**	**0.026**	**0.60**	**0.030**	**0.69**	**53.6**	**1.27**
Torus(8,8)	1.8	1195	2338	0.024	0.51	0.020	0.43	10.4	0.22
Cycle(8)^2, MD	2.2	1641	3239	0.029	0.61	0.019	0.37	9.6	0.19
Clique(4)^3	1.3	929	1345	0.022	0.44	0.024	0.47	16.8	0.32
Clique(4)^3, MD	2.1	1950	1919	0.027	0.60	0.022	0.48	14.4	0.32
Hyp(6)	2.1	970	1679	0.022	0.49	0.023	0.50	13.6	0.29
Hyp(2)^3, MD	2.1	1282	2481	0.025	0.51	0.019	0.40	9.6	0.20
Hyp(1)^6, MD	1.3	1355	2733	0.028	0.64	0.020	0.47	10.4	0.23
Butterfly(4)	2.0	1214	2288	0.024	0.50	0.020	0.41	10.4	0.22
DeBruijn(6)	1.7	1219	2314	0.026	0.54	0.022	0.44	11.2	0.23
Gossip (64)	1.4	991	1676	0.021	0.52	0.022	0.53	12.8	0.31
Gossip (62)	1.4	998	1850	0.022	0.45	0.022	0.45	12.0	0.24
Cage(6, 6)	1.4	1063	1624	0.023	0.49	0.022	0.46	14.4	0.30

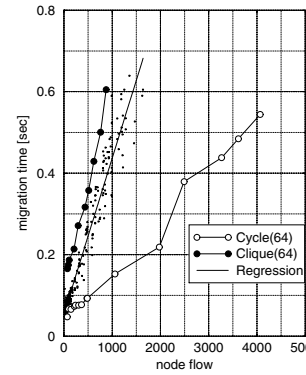


Fig. 3. Left: properties of the migration phase in case of the random load scenario. All entries are average values of 10 experiments with an average load of 800 load items. The diagram on the right correlates the node flow to the migration time measured on the PC-Cluster. The diagram is based on 300 experiments with various topologies including those listed in table on the left. Each point represents the average node flow and the corresponding average migration time of 10 experiments with the same initial total load.

after another. For example, consider the Cycle(8)^2 topology. In the first iteration, MD balances eight cycles in parallel. For a random load distribution, the system is balanced quite well after this first step. Experiments have shown that only about 20% of the total flow corresponds to the edges of the second dimension. This property has two consequences. Firstly, it leads to an unbalanced distribution of the communication load across the edges. We have already observed this effect in the scope of the peak scenario. Secondly, close migration partners for high loaded nodes may not be addressed in the first place since they are in a different dimension. This leads to unnecessary migrations and a high node flow. The diffusion scheme avoids these unnecessary migrations. Thus, schemes which use all edges in each step also generate small node flows (cf. the l_2-norm of the flows shown in Fig. 3(left)). Small values of $l_2(x)$ always imply small values of $f(x)$. We observe a strong correlation between these two measures. All experiments revealed that the quotient $f(x)/l_2(x)$ was always between 0.18 and 0.29. Its average value was 0.21. Thus, the diffusion schemes seem to generate flows that are close to optimal with respect to their node flow. In the case of the clique, the diffusion flow is in fact optimal with respect to the node flow.

Theorem 2. *The unique l_2-minimal balancing flow of the clique with n nodes and load vector $w \in \mathbb{R}^n$ has the node-flow $f(x) = \triangle w = \max_{1 \le i \le n} |w_i - \tilde{w}|$.*

Proof. The clique has the eigenvalues 0 and n. Thus, OPT calculates the l_2-minimal flow in one iteration and the flow over edge $\{u, v\}$ is $\frac{|w_u - w_v|}{n}$. The node-flow of a node v is $f(v) = \sum_{u \in V} \frac{|w_v - w_u|}{n}$. If $|\{u \in V; w_u \le w_v\}| < |\{u \in V; w_u > w_v\}|$, then $\sum_{u \in V} |w_v - w_u| \le \sum_{u \in V} |w_{min} - w_u|$, else $\sum_{u \in V} |w_v - w_u| \le \sum_{u \in V} |w_{max} - w_u|$, where w_{min} and w_{max} denote the minimum and maximum load of all nodes.

Thus, $f(v) = \sum_{u \in V} \frac{|w_v - w_u|}{n} \leq \max\{\sum_{u \in V} \frac{|w_{min} - w_u|}{n}, \sum_{u \in V} \frac{|w_{max} - w_u|}{n}\} \leq \triangle w$. Obviously, at least one node v has $f(v) \geq \triangle w$ which completes the proof. □

Thus, for a fixed number n of nodes and a load vector w, the l_2-minimal balancing flow of the clique has the minimal node-flow of any balancing flow on any topology.

Unfortunately, as we have seen before, the measure node-flow is too optimistic for the clique, due to the large degree. Fig. 3 shows that the migration time of the clique is larger than that of the diffusion scheme for the hypercube, although the node flow of the clique is smaller. Nevertheless, a small node flow is the main condition for a short migration phase; the l_2-optimal flow of the diffusion scheme implies a small node flow.

5 Conclusion

The choice of the topology and the load balancing scheme is closely related to the time requirement of the balancing process. For the flow-computation phase, a small *node degree* and a small *number of eigenvalues* of the network reduce the time requirement of the optimal scheme OPT. Besides, if the network can be represented as a cartesian product of several graphs, the scheme MD can be applied, using the much fewer eigenvalues of the factor graphs. The time-requirement of the migration phase depends on the load situation. In the peak-scenario, a small diameter of the topology is desirable. In the random scenario the migration phase is much faster and is dominated by the *node-flow*. Since the flows calculated by the diffusion schemes have low node-flow values, the diffusion schemes are particularly well suited.

Networks which simultaneously minimize the degree, the number of eigenvalues, the diameter, and the node-flow are desired. We proposed several graph classes, each of which minimizing some of these measures. Thus, our investigations in this paper are a first step to establish a set of topologies with small cost values.

References

[1] N. Biggs. *Algebraic Graph Theory, Second Edition*. Cambridge University Press, 1974/1993.
[2] F. Comellas. (degree,diameter)-graphs. http://www-mat.upc.es/grup_de_grafs/table_g.html.
[3] D.M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs*. Joh. Ambrosius Barth, 1995.
[4] G. Cybenko. Load balancing for distributed memory multiprocessors. *J. of Parallel and Distributed Computing*, 7:279–301, 1989.
[5] T. Decker. Virtual Data Space – Load balancing for irregular applications. *Parallel Computing*, 2000. To appear.
[6] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.
[7] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction load-balancing schemes. In *EuroPar'99*, LNCS 1685, pages 280–290, 1999.
[8] G. Fertin, A. Raspaud, H. Schröder, O. Sykora, and I. Vrto. Diamater of Knödel graph. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2000. to appear.
[9] 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.

[10] M. C. Heydemann, N. Marlin, and S. Perennes. Cayley graphs with complete rotations. Technical Report 1155, L.R.I. Orsay, 1997.
[11] Y.F. Hu and R.J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4):417–444, 1999.
[12] Y.F. Hu, R.J. Blake, and D.R. Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency: Prac. and Exp.*, 10(6):467–483, 1998.
[13] W. Knödel. New gossips and telephones. *Discrete Mathematics*, 13:95, 1975.
[14] G. Royle. Cages of higher valency. http://www.cs.uwa.edu.au/∼gordon/cages/allcages.html.
[15] P. Sanders. Analysis of nearest neighbor load balancing algorithms for random loads. *Parallel Computing*, 25:1013–1033, 1999.
[16] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. of Parallel and Distributed Computing*, 47(2):109–124, 1997.
[17] R.S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, 1962.
[18] C. Xu and F.C.M. Lau. *Load Balancing in Parallel Computers*. Kluwer, 1997.