

# Strongly Adaptive Token Distribution<sup>1</sup>

F. Meyer auf der Heide,<sup>2</sup> B. Oesterdiekhoff,<sup>2</sup> and R. Wanka<sup>2</sup>

**Abstract.** The token distribution (TD) problem, an abstract static variant of load balancing, is defined as follows: let  $M$  be a (parallel processor) network with set  $\mathcal{P}$  of processors. Initially, each processor  $P \in \mathcal{P}$  has a certain amount  $l(P)$  of tokens. The goal of a TD algorithm, run on  $M$ , is to distribute the tokens evenly among the processors. In this paper we introduce the notion of strongly adaptive TD algorithms, i.e., algorithms whose running times come close to the best possible runtime, the off-line complexity of the TD problem, for each individual initial token distribution  $l$ . Until now, only weakly adaptive algorithms have been considered, where the running time is measured in terms of the maximum initial load  $\max\{l(P) | P \in \mathcal{P}\}$ .

We design an almost optimal, strongly adaptive algorithm on mesh-connected networks of arbitrary dimension extended by a single 1-bit bus. This result shows that an on-line TD algorithm can come close to the optimal (off-line) bound for each individual initial load. Furthermore, we exactly characterize the off-line complexity of arbitrary initial token distributions on arbitrary networks. As an intermediate result, we design almost optimal weakly adaptive algorithms for TD on mesh-connected networks of arbitrary dimension.

**Key Words.** Parallel algorithms, Token distribution.

## 1. Introduction.

**1.1. Computation Model.** The underlying parallel computation model is the (parallel processor) network. Such a network consists of a set of  $p$  processors,  $\mathcal{P} = \{P_1, \dots, P_p\}$  pairs of which are connected via bidirectional links forming a communication graph  $M = (\mathcal{P}, E)$  with  $E$  denoting the set of links. We identify the network with its communication graph. The network is assumed to be synchronized; in a computation step, each processor can do a constant amount of internal computation and can send a message to a neighboring processor. We demand that each processor can receive at most one message per step.

**1.2. The Token Distribution Problem.** *Load balancing* is one of the basic tasks to be performed in order to achieve efficient execution of parallel programs on a network: if a processor is overloaded with work during a computation, it tries to reduce its load by shifting part of it to less busy processors. The aim is to keep the load balanced among the processors.

*Token distribution* (TD) is an abstraction of a static variant of load balancing. Initially each processor  $P$  has a number  $l(P)$  of tokens, i.e., the *initial load* is given by a function  $l: \mathcal{P} \rightarrow \mathbb{N}$ . We refer to  $N = \sum_{P \in \mathcal{P}} l(P)$  as the *total load*, to  $k = \max\{l(P) | P \in \mathcal{P}\}$  as the *maximum load*, and to  $N/p$  as the *average load*.

<sup>1</sup> This research was partially supported by DFG-Forschergruppe "Effiziente Nutzung massiv paralleler Systeme, Teilprojekt 4," by the ESPRIT Basic Research Action No. 7141 (ALCOM II), and by the Volkswagenstiftung. A preliminary version was presented at the 20th ICALP, 1993, see [9].

<sup>2</sup> Fachbereich Mathematik-Informatik and Heinz-Nixdorf-Institut, Universität-GH Paderborn, D-33095 Paderborn, Germany. {fmadh,brigitte,wanka}@uni-paderborn.de.

The goal of token distribution is to distribute the tokens given by the initial load so that the final load of each processor is close to the average load. More specifically, a TD algorithm is  $\delta$ -exact if finally no processor holds more than  $\lceil N/p \rceil + \delta$  tokens. Furthermore, we denote the maximum load difference of the processors with *discrepancy*.

**1.3. Complexity Measures of Token Distribution.** In order to be able to measure the performance of a TD algorithm, we first introduce a quantity which is a lower bound on the running time of any TD algorithm on  $M$  with initial load  $l$ . For this purpose, we consider off-line algorithms for token distribution. In this case, with  $M, l$  given, we allow an arbitrarily complex preprocessing that can be executed without being added to the complexity and that produces a protocol for each processor telling, for each time  $t$ , whether and, if yes, where to send a token. We assume that a processor can send and receive one token per time step. The *off-line complexity* of the TD problem  $(M, l)$ , the TD problem on  $M$  with initial load  $l$ , is  $T^{\text{off}}(M; l, \delta)$ , the running time of a fastest  $\delta$ -exact off-line TD algorithm.

On-line algorithms are designed for a fixed network  $M$ , but do not allow any free preprocessing given an initial load  $l$ . Thus, a TD algorithm executes *distribution steps*, where each processor can send and receive one token, and *computation steps*, where computation and communication can take place (e.g., to gather information about the current load distribution). No tokens are moved in such a step. The *on-line complexity* of the TD problem  $(M, l)$  is  $T(M; l, \delta)$ , the running time of a fastest  $\delta$ -exact on-line TD algorithm on  $M$  started with initial load  $l$ .

All papers mentioned below consider the following *adaptive complexity measure* for TD on  $M$ :

$$T_{\text{ad}}(M; N, k, \delta) = \max\{T(M; l, \delta) \mid l \text{ has maximum load } \leq k \text{ and total load } \leq N\}.$$

In this paper we consider an even stronger version of adaptivity: we want to design on-line TD algorithms which come close to the performance of off-line algorithms of each individual initial load function, i.e., we design *strongly adaptive* TD algorithms, showing that their performance comes close to the lower bound, i.e., the off-line complexity. The strongly adaptive complexity is the running time of a fastest  $\delta$ -exact on-line TD algorithm depending on the off-line complexity  $T^{\text{off}}(M; l, \delta)$ , the network  $M$ , and the maximum load  $k$  of the individual load  $l$ . We assume that the total load  $N$  is initially known to all processors.

**1.4. Known Results About Token Distribution.** The token distribution problem was introduced by Peleg and Upfal [10]. For arbitrary networks  $M$  with  $p$  processors and total load  $N = p$ , the same authors show in [11] that, for all  $k \geq 2$ ,

$$T_{\text{ad}}(M; p, k, 0) = O(p) \quad \text{and} \quad T_{\text{ad}}(M; p, k, 0) = \Omega(k + \text{diam}(M)),$$

with  $\text{diam}(M)$  denoting the diameter of  $M$ .

Matching upper bounds are shown in [11], by Herley [4], and by Broder *et al.* [3] for certain classes  $\mathcal{K}$  of expander-related, bounded-degree, low-diameter networks, i.e.,

$$T_{\text{ad}}(M; p, k, O(1)) = O(k + \log p)$$

for all  $M \in \mathcal{K}$ .

Aiello *et al.* [1] analyze a very simple algorithm for arbitrary graphs and show by using advanced and elegant techniques that, for constant-degree graphs  $G_\mu$  with  $p$  processors and expansion  $\mu$ ,

$$T_{\text{ad}}\left(G_\mu; N, k, O\left(\frac{1}{\mu} \log p\right)\right) = O\left(\frac{k \cdot p - N}{p \cdot \mu} \log(k \cdot p - N)\right).$$

Plaxton [12] investigates the TD problem on the  $d$ -dimensional hypercube  $Q_d$ . He shows that, for  $k \geq 2 \cdot N/p$ ,

$$T_{\text{ad}}(Q_d; N, k, 0) = \Omega\left(k \cdot \sqrt{\frac{d}{d + \log(k/N)}}\right)$$

and  $T_{\text{ad}}(Q_d; \text{poly}(p), k, 0) = O(k\sqrt{d} + d^2)$ . Werchner [16] deals with certain TD problems on  $Q_d$ . By carefully analyzing expansion properties of  $Q_d$ , he shows that  $T_{\text{ad}}(Q_d; k \cdot p^{1/2 - \Omega(1)}, k, 0) = O(k + d^2 \cdot \log d)$ .

Jájá and Ryu [5] investigate the TD problem for the cube-connected cycles network, shuffle-exchange network, and the butterfly network. Qiu and Akl [13] do so for the star and pancake networks.

Makedon and Symvonis [8] address the many-to-one routing problem on the two-dimensional  $\sqrt{p}$ -sided mesh  $M(\sqrt{p}, 2)$ . Their results can easily be modified to show that

$$T_{\text{ad}}(M(\sqrt{p}, 2); p, k, 0) = \Theta(\sqrt{k \cdot p}).$$

Very recently, the many-to-one routing problem for higher-dimensional meshes was investigated in [14], where similar bounds are shown.

A further approach to the TD problem is *counting networks* introduced by Aspnes *et al.* [2] and improved by Klugerman and Plaxton [6], where circuits for the TD problem are constructed similar to sorting circuits.

**1.5. New Results About Token Distribution.** In this paper we characterize almost exactly the off-line complexity of TD. Furthermore, we consider TD on the  $d$ -dimensional  $n$ -sided mesh-connected network  $M(n, d)$  and characterize its adaptive TD complexity up to a factor of  $O(d^2)$  (resp.  $O(d \cdot 2^d)$  if the maximum load is small). Finally, we present a strongly adaptive algorithm for TD on  $M(n, d)$  which only differs by a factor of  $O(d^3 \cdot \log k)$  (resp.  $O(d \cdot 2^d + d^3 \cdot \log k)$  if the maximum load is small) from the off-line bound. More specifically, we prove:

**THEOREM 1 (Off-line Complexity of TD).** *Let  $M$  be a network and let  $l$  be an initial*

load. For  $X \subseteq \mathcal{P}$ , let  $I(X) := \sum_{P \in X} l(P)$  be the load of  $X$ , and let  $N = I(\mathcal{P})$ .  $\text{match}(X)$  denotes the cardinality of a maximum matching between  $X$  and  $\mathcal{P} \setminus X$ .

$$(a) \quad T^{\text{off}}(M; l, 0) \geq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil.$$

$$(b) \quad T^{\text{off}}(M; l, 1) \leq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil.$$

Thus, we obtain an exact characterization of the off-line complexity of TD with the restriction that the upper bound only holds if we allow the final maximum load to be one token larger than desired. The following observation shows, if we allow only

$$\max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil$$

steps, that TD cannot be solved 0-exactly.

**OBSERVATION 1.** *There is an infinite family of TD problems  $(M, l)$  with the following properties:*

$$N = p,$$

$$k \leq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil = \frac{1}{2}p + 1 =: T,$$

and

$$T^{\text{off}}(M; l, 0) \geq T + 2 \cdot \sqrt{T-1} - 4.$$

**THEOREM 2 (Weakly Adaptive Complexity of TD on Meshes).**

$$(a) \text{ For } k \geq 2 \cdot (N/n^d), T_{\text{ad}}(M(n, d); N, k, 0) = \Omega((1/d) \cdot \sqrt[d]{N \cdot k^{d-1}}).$$

$$(b) T_{\text{ad}}(M(n, d); N, k, 0) = O(2^d \cdot \sqrt[d]{N \cdot k^{d-1}} + 2^d \cdot n).$$

$$\text{For } k \geq 2 \cdot (N/n^d), T_{\text{ad}}(M(n, d); N, k, 0) = O(d \cdot \sqrt[d]{N \cdot k^{d-1}} + d^2 \cdot n).$$

This theorem extends the result of [8] mentioned above. We use its basic ideas, but have to add further techniques to generalize for arbitrary total load and arbitrary dimension. Note that, for constant  $d$ , the bounds in Theorem 2 match. We extend our underlying parallel computation model by a global bus capable of storing one bit. Each processor is able to write on and to read from this bus in one time step. By using the weakly adaptive algorithms as subroutines, we show the following main result of this paper.

**THEOREM 3 (Strongly Adaptive Complexity of TD on Meshes).**  $T(M(n, d); l, 1) = O((d \cdot 2^d + d^3 \cdot \log k) \cdot T^{\text{off}}(M(n, d), l, 0))$ . For  $k \geq 2 \cdot (N/n^d)$ ,  $T(M(n, d); l, 1) = O((d^3 \cdot \log k) \cdot T^{\text{off}}(M(n, d), l, 0))$  for each individual load  $l$  with maximum load  $k$ .

This result shows that an on-line TD algorithm can come close to the optimal (off-line) bound for each individual initial load.

## 2. The Off-line Complexity of Token Distribution

**THEOREM 1 (Off-line Complexity of TD).** *Let  $M$  be a network, and let  $l$  be an initial load. For  $X \subseteq \mathcal{P}$ , let  $I(X) := \sum_{P \in X} l(P)$  be the load of  $X$ , and let  $N = I(\mathcal{P})$ .  $\text{match}(X)$  denotes the cardinality of a maximum matching between  $X$  and  $\mathcal{P} \setminus X$ .*

$$(a) \quad T^{\text{off}}(M; l, 0) \geq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil.$$

$$(b) \quad T^{\text{off}}(M; l, 1) \leq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil.$$

**2.1. Off-line Lower Bound.** The lower bound is based on expansion properties of  $M$  like Plaxton's lower bound [12]. It is similar to that mentioned by Peleg and Upfal [11].

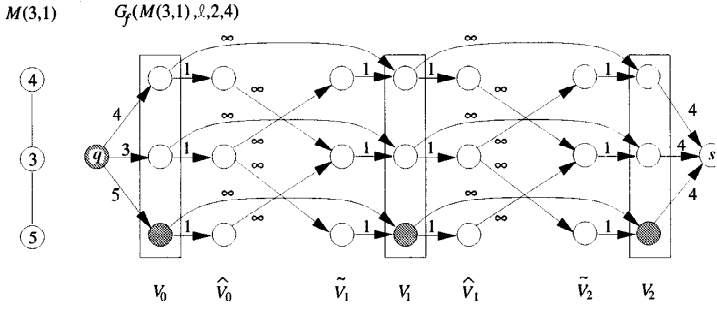
**PROOF OF THEOREM 1(a).** Consider an arbitrary set  $X$  of processors,  $\emptyset \subsetneq X \subsetneq \mathcal{P}$ . Initially,  $X$  contains  $I(X)$  tokens, finally at most  $\lceil N/p \rceil \cdot |X|$ . Thus,  $I(X) - \lceil N/p \rceil \cdot |X|$  tokens have to leave  $X$ . At most  $\text{match}(X)$  tokens can leave  $X$  per time step. Thus, any solution to the TD problem requires at least  $\max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \lceil (I(X) - \lceil N/p \rceil \cdot |X|) / \text{match}(X) \rceil$  distribution steps.  $\square$

In what follows, we show that this bound is tight for arbitrary networks, if we allow preprocessing, and if some processors may eventually hold one token more than the average load. Note that, in this case, the diameter of  $M$  does not influence the off-line running time in the following sense. Consider a graph consisting of  $K_{p/2}$ , the complete graph of  $\frac{1}{2}p$  nodes, and a linear array of length  $\frac{1}{2}p$  where one of the endnodes of the array is connected to an arbitrary node of  $K_{p/2}$ . The diameter of the graph is  $\frac{1}{2}p + 1 = \Theta(p)$ . If we have  $p$  tokens only distributed on  $K_{p/2}$  with initial maximum load  $k = O(\log p)$ , these tokens can be distributed on  $K_{p/2}$  in  $O(\log p)$  steps so that finally each processor has exactly two tokens, i.e., one token more than the average load. However, if it is required that each processor holds exactly one token, each algorithm needs at least  $\Omega(p)$  steps for the given problem.

### 2.2. An Almost Optimal Off-line Algorithm

**PROOF OF THEOREM 1(b).** To prove the theorem, we use a novel approach. We transform the TD problem on  $M = (\mathcal{P}, E)$  with initial load  $l$  and desired final maximum load  $m$  into a maximum flow problem on a directed flow graph  $G_f(M, l, t, m)$  (or  $G_f$  for short). The parameter  $t$  stands for the running time we allow to solve the TD problem. For the terminology of network flow methods, see, e.g., [15]. An example for the transformation of the linear array  $M(3, 1)$  of length 3 with initial load  $l$ ,  $l(P_1) = 4$ ,  $l(P_2) = 3$ ,  $l(P_3) = 5$ , into the flow graph  $G_f(M(3, 1), l, 2, 4)$  is given in Figure 1.

$G_f$  consists of main levels  $V_0, \dots, V_t$  and, associated with them, helping levels  $\widehat{V}_0, \dots, \widehat{V}_{t-1}, \widetilde{V}_1, \dots, \widetilde{V}_t$ . Each level consists of copies of all processors of  $M$ . The flow from the source  $q$  to a vertex in  $V_i$  represents the load of the corresponding processor



**Fig. 1.** Transformation of  $(M(3, 1), l)$  into the flow-graph  $G_f(M(3, 1), l, 2, 4)$ . The four marked nodes form a cut  $C$  with  $\text{cap}(C) = 13$ .

of  $M$  after  $i$  distribution steps of a possible TD algorithm. In particular, a path between two main levels  $V_i$  and  $V_{i+1}$  models a single distribution step.

To establish the initial distribution of tokens, we have edges from the source  $q$  to the vertices in  $V_0$ , where the capacities are chosen with respect to the initial load  $l$ . As the vertices in  $V_t$  represent the token distribution at the end of a possible TD algorithm, these vertices are connected to the sink  $s$  via edges with capacity  $m$ .

To model the constraints of a single distribution step, we use two helping levels  $\widehat{V}_i$  and  $\widetilde{V}_{i+1}$  between each  $V_i$  and  $V_{i+1}$ ,  $i \in \{0, \dots, t-1\}$ . The reason is that we have to take care of the following mode of communication among processors as described in the Introduction:

1. Each processor can send at most one token per time step (ensured by the edges with capacity 1 between  $V_i$  and  $\widehat{V}_i$ ).
2. Each processor can receive at most one token per time step (ensured by the edges with capacity 1 between  $\widetilde{V}_{i+1}$  and  $V_{i+1}$ ).
3. Each processor holds the tokens not transmitted (ensured by the edges with infinite capacity between  $V_i$  and  $V_{i+1}$ ).

See Figure 1 for the details of realizing 1–3. Finally, the helping levels are connected with respect to the links of  $M$  by edges with capacity  $\infty$ .

As all capacities are integers, the Integrality Theorem (see [15]) ensures that there is a maximum flow such that all flows along all edges are integers. Therefore, if there is a maximum flow with value  $N$  in  $G_f$ , the  $N$  tokens can be redistributed among  $M$  in  $t$  steps, so that eventually the maximum number of tokens in any processor is at most  $m$ . Of course, we want  $m = \lceil N/p \rceil$  (average load) and  $t$  as small as possible. The following lemma implies Theorem 1(b).

**LEMMA 1.** *The value of a maximum flow on  $G_f = G_f(M, l, T, \lceil N/p \rceil + 1)$  is  $N$ , if*

$$T = \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil.$$

PROOF. We use the well-known Maxflow–Mincut Theorem (see [15]). Let  $V_f$  be the vertices and  $c$  the capacities in  $G_f$ . Let  $C \subset V_f$  be an arbitrary  $(q, s)$ -cut in  $G_f$ , i.e.,  $q \in C$  and  $s \notin C$  (see Figure 1). Let  $out(C)$  denote the set of edges from nodes in  $C$  to nodes in  $V_f \setminus C$ , and  $cap(C) = \sum_{e \in out(C)} c(e)$  the capacity of  $C$ . We show that  $cap(C) \geq N$ .

For  $u \in P$ , let  $u_i \in V_i$  denote the copy of  $u$  in  $V_i$ . Let  $C_i = C \cap V_i$ ,

$$orig(C_i) = \{u \in V \mid u_i \in C_i\}.$$

If  $C_0 = \emptyset$ , or  $C_i = V_i$ , or  $out(C)$  contains an edge with infinite capacity,  $cap(C) \geq N$  is obviously true.

Now assume that  $C_0 \neq \emptyset$ ,  $C_i \neq V_i$ , and  $out(C)$  contains none of the edges with infinite capacity. As a consequence, we have  $C_{i+1} \supseteq \{u_{i+1} \mid u_i \in C_i\}$ . If  $C_{i+1} = \{u_{i+1} \mid u_i \in C_i\}$ , the subgraph induced by  $V_i$ ,  $V_{i+1}$ , and the corresponding helping levels contributes at least  $match(orig(C_i))$  to  $cap(C)$ , because a maximum matching between  $orig(C_i)$  and  $orig(V_i \setminus C_i)$  corresponds directly to a system of vertex-disjoint paths between  $V_i$  and  $V_{i+1}$ , each path leaving  $C$ . That means for each edge  $\{u, v\}$  of the maximum matching between  $orig(C_i)$  and  $orig(V_i \setminus C_i)$ ,  $u_i \in C_i$  and  $v_{i+1} \notin C_{i+1}$ . Thus, there is a cut of at least 1 on the path between  $u_i$  and  $v_{i+1}$ . If  $C_{i+1} \supsetneq \{u_{i+1} \mid u_i \in C_i\}$ , each vertex in  $C_{i+1} \setminus \{u_{i+1} \mid u_i \in C_i\}$  decreases the capacity  $match(orig(C_i))$  by at most 1 because the paths to them do not leave  $C$ . Thus, the contribution of the edges between  $V_i$  and  $V_{i+1}$  to  $cap(C)$  is at least  $match(orig(C_i)) - (|C_{i+1}| - |C_i|)$  for all  $i < T$ . Using this fact and that

$$T \geq \frac{I(orig(C_i)) - \lceil N/p \rceil \cdot |C_i|}{match(orig(C_i))} \quad \text{for all } i,$$

we have the following estimation for  $cap(C)$ :

$$\begin{aligned} cap(C) &= \sum_{u \in orig(V_0 \setminus C_0)} l(u) + \sum_{i=0}^{T-1} \left( match(orig(C_i)) - (|C_{i+1}| - |C_i|) \right) \\ &\quad + \left( \left\lceil \frac{N}{p} \right\rceil + 1 \right) \cdot |C_T| \\ &= I(orig(V_0 \setminus C_0)) + |C_0| + \left\lceil \frac{N}{p} \right\rceil \cdot |C_T| + \sum_{i=0}^{T-1} match(orig(C_i)) \\ &\geq I(orig(V_0 \setminus C_0)) + |C_0| + \left\lceil \frac{N}{p} \right\rceil \cdot |C_T| + T \cdot match(orig(C_j)) \\ &\geq \underbrace{I(orig(V_0 \setminus C_0)) + I(orig(C_j))}_{\geq N} + |C_0| + \left\lceil \frac{N}{p} \right\rceil \cdot \underbrace{(|C_T| - |C_j|)}_{\geq 0} \\ &\geq N, \end{aligned}$$

where  $j$  is such that  $match(orig(C_j)) = \min\{match(orig(C_i)) \mid 0 \leq i \leq T - 1\}$ . Thus,  $C = \{q\}$  is a minimum cut with  $cap(C) = N$ . □

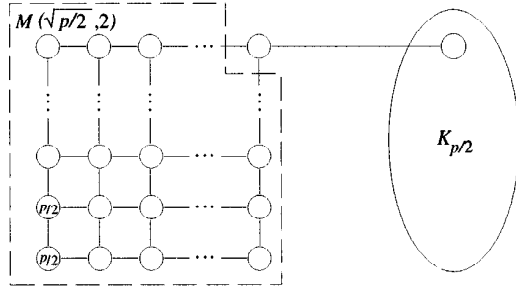


Fig. 2. 0-exact TD requires more than \$T\$ steps.

OBSERVATION 1. *There is an infinite family of TD problems \$(M, l)\$ with the following properties:*

$$N = p,$$

$$k \leq \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil = \frac{1}{2}p + 1 =: T,$$

and

$$T^{\text{off}}(M; l, 0) \geq T + 2 \cdot \sqrt{T - 1} - 4.$$

PROOF. Consider the network \$M\$ shown in Figure 2 and the given initial load. \$K\_{p/2}\$ denotes the complete network consisting of \$\frac{1}{2}p\$ processors. For this TD problem, we have

$$T = \max_{\emptyset \subsetneq X \subsetneq \mathcal{P}} \left\lceil \frac{I(X) - \lceil N/p \rceil \cdot |X|}{\text{match}(X)} \right\rceil = \frac{1}{2}p + 1$$

(see the marked region). Because of the diameter of the marked region, at least \$\frac{1}{2}p + 1 + 2 \cdot \sqrt{\frac{1}{2}p} - 4\$ distribution steps are necessary.  $\square$

### 3. The Adaptive Complexity of Token Distribution on Meshes.

THEOREM 2 (Weakly Adaptive Complexity of TD on Meshes).

- (a) For \$k \ge 2 \cdot (N/n^d)\$, \$T\_{\text{ad}}(M(n, d); N, k, 0) = \Omega((1/d) \cdot \sqrt[d]{N \cdot k^{d-1}})\$.
  - (b) \$T\_{\text{ad}}(M(n, d); N, k, 0) = O(2^d \cdot \sqrt[d]{N \cdot k^{d-1}} + 2^d \cdot n)\$.
- For \$k \ge 2 \cdot (N/n^d)\$, \$T\_{\text{ad}}(M(n, d); N, k, 0) = O(d \cdot \sqrt[d]{N \cdot k^{d-1}} + d^2 \cdot n)\$.

#### 3.1. Lower Bound

PROOF OF THEOREM 2(a). We construct a poor input for the \$d\$-dimensional \$n\$-sided mesh \$M(n, d)\$ as follows: Concentrate all tokens in a corner region of \$M(n, d)\$ that

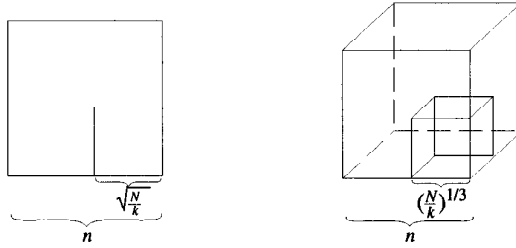


Fig. 3. All tokens are located at a corner of the mesh.

is isomorphic to  $M((N/k)^{1/d}, d)$ . (The situations for  $d = 2$  and  $d = 3$  are shown in Figure 3). This corner region has  $d \cdot (N/k)^{(d-1)/d}$  outgoing links. Arguing as in the proof of Theorem 1(a), any solution to the problem described above requires at least

$$\begin{aligned} \frac{N - (N/n^d) \cdot (N/k)}{d \cdot (N/k)^{(d-1)/d}} &= \left(1 - \frac{N}{n^d \cdot k}\right) \cdot \frac{1}{d} \cdot \sqrt[d]{N \cdot k^{d-1}} \\ &\geq \frac{1}{2} \cdot \frac{1}{d} \cdot \sqrt[d]{N \cdot k^{d-1}} \\ &= \Omega\left(\frac{1}{d} \cdot \sqrt[d]{N \cdot k^{d-1}}\right) \end{aligned}$$

distribution steps, if  $k \geq 2 \cdot (N/n^d)$ . □

Note that the condition  $k \geq 2 \cdot (N/n^d)$  is true for each TD problem on  $M(n, d)$  with total load  $N = n^d$ .

### 3.2. Adaptive Algorithm

PROOF OF THEOREM 2(b). We first present a TD algorithm requiring  $O(2^d \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$  steps. Note that this is optimal when  $d$  is constant. Concluding this section, we present an algorithm that is faster for the case that  $k \geq 2 \cdot (N/n^d)$ .

Let  $M_1, M_2, \dots, M_n$  be a partition of  $M(n, d)$  into  $n$  copies of  $M(n, d - 1)$ , and let  $A_1, \dots, A_{n^{d-1}}$  be a partition into  $n^{d-1}$  linear arrays of length  $n$  where each vertex of an array belongs to a different  $M_r$ . The partition of  $M(n, d)$  into  $M_1, M_2, \dots, M_n$  and one  $A_s$  is shown in Figure 4.

The following algorithm, started on  $M(n, d)$ , generates a distribution with discrepancy 1. Note that such a distribution is 0-exact.

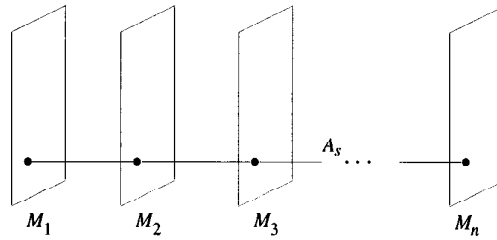


Fig. 4. Partition of  $M(n, d)$ .

### TD algorithm

**if**  $d = 1$  **then**

1. Distribute the tokens with discrepancy 1.

**if**  $d > 1$  **then**

2. In each  $M_r$ , color  $\min\{a_r, \sqrt[d]{N^{d-1} \cdot k}\}$  arbitrary tokens of the  $a_r$  tokens in  $M_r$  black, and all remaining tokens white.

3. Recursively distribute the black tokens in each  $M_r$  with discrepancy 1.

4. Let  $k_r$  be such that each processor in  $M_r$  contains  $k_r$  or  $k_r + 1$  black tokens. Select  $k_r$  of the black tokens in each processor of  $M_r$ .

5. Distribute the selected black tokens in each  $A_s$  with discrepancy 1.

*Comment:* Note that the current distribution of the black tokens has overall discrepancy 2.

6. Distribute the white tokens in each  $A_s$  with discrepancy 1.

7. Let  $h_s$  be such that each processor in  $A_s$  contains  $h_s$  or  $h_s + 1$  white tokens. Select  $h_s$  of the white tokens in each processor of  $A_s$ .

8. Recursively distribute the selected white tokens in each  $M_r$  with discrepancy 1.

*Comment:* Note that the current distribution of the white tokens has overall discrepancy 2. Thus, the current distribution of all tokens has discrepancy 4.

9. Distribute the tokens in  $M(n, d)$ , so that the discrepancy decreases to 1.

LEMMA 2. *The above algorithm solves each TD problem on  $M(n, d)$  with maximum load  $k$  in  $O(2^d \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$  steps with discrepancy 1.*

PROOF. ( $d = 1$ ) Obviously, Step 1 can be executed in time  $O(N + n)$  (see [7]).

( $d > 1$ ) The method of the algorithm is to split up the tokens into two sets and to balance one set at first in each  $M_r$  and then across each  $A_s$  and the other set at first across each  $A_s$  and then in each  $M_r$ .

In Step 2 we choose at most  $\sqrt[d]{N^{d-1} \cdot k}$  black tokens and the rest of tokens are white, because this splitting guarantees that the time complexity for distributing the tokens in the  $(d - 1)$ -dimensional mesh  $M_r$  (Steps 3 and 8) is not too large. This splitting for the

case  $d = 2$  and  $N = n^2$  is used by Makedon and Symvonis [8] for many-to-one routing similarly.

To execute Step 2, the tokens in each  $M_r$  are counted first. Then each processor knows how many of its at most  $k$  tokens it has to color black. In order to count the tokens, a prefix computation and a broadcast have to be done on a  $d$ -dimensional  $n$ -sided mesh. Both of these computations can be realized in  $O(d \cdot n)$  steps by running  $d$  phases in each phase communicating on linear arrays of  $n$  processors. For these procedures, see [7]. Thus, Step 2 requires  $O(d \cdot n + k) = O(d \cdot n + \sqrt[d]{N \cdot k^{d-1}})$  steps.

In Step 3 at most  $\sqrt[d]{N^{d-1} \cdot k}$  black tokens in each  $M_r$  will be balanced with discrepancy 1. By inductive assumption, balancing  $\sqrt[d]{N^{d-1} \cdot k}$  black tokens in Step 3 needs  $O(2^{d-1} \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$  steps.

In order to do Step 4, the value  $k_r$  has to be known by each processor of  $M_r$ . Obviously, this can be done in  $O(d \cdot n)$  steps by a prefix computation and a broadcast.

Since the numbers of selected black tokens are the same in each  $A_s$  and there are exactly  $N$  tokens in the mesh, there are at most  $N/n^{d-1} \leq \sqrt[d]{N \cdot k^{d-1}}$  selected black tokens in every  $A_s$ . Thus, balancing in Step 5 requires  $O(\sqrt[d]{N \cdot k^{d-1}} + n)$  time steps.

Since in each  $M_r$  up to  $\sqrt[d]{N^{d-1} \cdot k}$  tokens are black, and since there are exactly  $N$  tokens in the mesh, there are less than  $N/\sqrt[d]{N^{d-1} \cdot k} = (N/k)^{1/d}$  submeshes  $M_r$  containing white tokens. Thus, the total number of white tokens in each  $A_s$  is at most  $(N/k)^{1/d} \cdot k = \sqrt[d]{N \cdot k^{d-1}}$ . Thus, Step 6 requires  $O(\sqrt[d]{N \cdot k^{d-1}} + n)$  steps.

Step 7 can be done in the same way as Step 4. Since the number of selected white tokens are the same in each  $M_r$ , there are at most  $N/n \leq \sqrt[d]{N^{d-1} \cdot k}$  selected white tokens in each  $M_r$ , after Step 7. Thus, Step 8 requires  $O(2^{d-1} \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$  steps.

In subsection 3.4.3 of [7], a monotone routing algorithm on  $d$ -dimensional hypercubes requiring  $O(d)$  steps is described. It can easily be generalized for meshes, needing  $O(d \cdot n)$  steps on  $M(n, d)$ . In Step 9 we redistribute the tokens in at most three phases, each phase reducing discrepancy by at least 1, performing monotone routing. Thus, Step 9 requires  $O(d \cdot n)$  steps.

Altogether, the TD algorithm requires  $O(2^d \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$  steps.  $\square$

For completeness, we present an adaptive TD algorithm achieving a better performance for a large maximum load, i.e.,  $k \geq 2 \cdot (N/n^d)$ . It requires  $O(d \cdot \sqrt[d]{N \cdot k^{d-1}} + d^2 \cdot n)$  steps. Note that the condition  $k \geq 2 \cdot (N/n^d)$  is always true for TD problems with total load  $N = n^d$ .

### Faster TD algorithm

**if  $d = 1$  then**

1. Distribute the tokens with discrepancy 1.

**if  $d > 1$  then**

2. In each  $M_r$ , color  $\min\{a_r, ((\sqrt[d]{2} - 1)/\sqrt[d]{2}) \sqrt[d]{N^{d-1} \cdot k}\}$  arbitrary tokens of the  $a_r$  tokens in  $M_r$  black, and the remaining tokens white.
3. Distribute the white tokens in each  $A_s$  with discrepancy 1.
4. Let  $k_s$  be such that each processor in  $A_s$  contains  $k_s$  or  $k_s + 1$  white tokens. Select  $k_s$  of the white tokens in each processor of  $A_s$ .

5. Recursively distribute the black tokens together with the selected white tokens in each  $M_r$  with discrepancy 1.
6. Let  $h_r$  be such that each processor in  $M_r$  contains exactly  $h_r, h_r + 1,$  or  $h_r + 2$  tokens. Select  $h_r$  of the tokens in each processor of  $M_r$ .
7. Distribute the selected tokens in each  $A_s$  with discrepancy 1.

*Comment:* Note, the current distribution has discrepancy at most 3.

8. Distribute the tokens in  $M(n, d)$ , so that the discrepancy is reduced to 1.

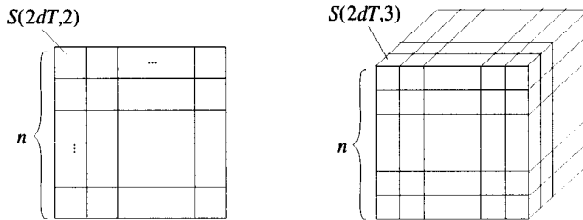
The splitting into black and white tokens in Step 2 and the condition  $k \geq 2 \cdot (N/n^d)$  guarantee that the time complexity for distributing the tokens in Step 5 is not too large. By methods similar to those used in the proof of Lemma 2, the correctness and the time complexity of the above algorithm can be proved. Thus, Theorem 2(b) is proved.

#### 4. The Strongly Adaptive Complexity of Token Distribution on Meshes

**THEOREM 3** (Strongly Adaptive Complexity of TD on Meshes).  $T(M(n, d); l, 1) = O((d \cdot 2^d + d^3 \cdot \log k) \cdot T^{\text{off}}(M(n, d), l, 0))$ . For  $k \geq 2 \cdot (N/n^d)$ ,  $T(M(n, d); l, 1) = O(d^3 \cdot \log k) \cdot T^{\text{off}}(M(n, d), l, 0)$  for each individual load  $l$  with maximum load  $k$ .

**PROOF.** In Section 3.2 it is shown that the (adaptive) complexity of the TD problem on  $M(n, d)$  is  $O(2^d \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$ , and for the case  $k \geq 2 \cdot (N/n^d)$  the complexity can be improved to  $O(d^2 \cdot (\sqrt[d]{N \cdot k^{d-1}} + n))$ . Let  $T := T^{\text{off}}(M(n, d), l, 0)$  be the off-line complexity of the TD problem on  $M(n, d)$ . Now we present a strongly adaptive algorithm solving the problem 1-exactly in  $O((d \cdot 2^d + d^3 \cdot \log k) \cdot T)$  steps and in  $O(d^3 \cdot \log k \cdot T)$  steps, if  $k \geq 2 \cdot N/n^d$ , respectively. For that purpose, we partition  $M(n, d)$  into submeshes  $S(2dT, d)$  that are isomorphic to  $M(2dT, d)$ . For example, we have illustrated such partitionings for  $d = 2$  and  $d = 3$ , respectively, in Figure 5.

The following lemma is the key observation for our method. It shows that if we solve the TD problem locally in each  $S(2dT, d)$ , then the whole  $M(n, d)$  is balanced 1-exactly.



**Fig. 5.** Partitioning into submeshes.

LEMMA 3. *Let  $S$  be an arbitrary submesh isomorphic to  $M(m, d)$  in  $M(n, d)$ . Let  $L$  be the load of an arbitrary processor in  $S$  after balancing  $S$ . Then  $\lfloor \lfloor N/n^d \rfloor - 2dT/m \rfloor \leq L \leq \lceil \lceil N/n^d \rceil + 2dT/m \rceil$ .*

PROOF. Let  $P$  be the total number of tokens initially located in an arbitrary submesh  $S$  of  $M(n, d)$ .  $T$  is the real off-line complexity for solving the TD problem on  $M$  with initial load  $l$  0-exactly. Thus, an optimal off-line algorithm balances during its execution on  $M(n, d)$  each  $S$  in  $T$  off-line steps. At most  $2d \cdot m^{d-1}$  tokens can leave  $S$  per time step, and at most  $\lceil N/n^d \rceil \cdot m^d$  tokens may be in  $S$  after  $T$  time steps. Thus,  $P - 2dT \cdot m^{d-1} \leq \lceil N/n^d \rceil \cdot m^d$ . Analogously, at most  $2d \cdot m^{d-1}$  tokens can enter  $S$  per time step, and at least  $\lfloor N/n^d \rfloor \cdot m^d$  tokens must be in  $S$  after  $T$  time steps. Thus,  $P + 2dT \cdot m^{d-1} \geq \lfloor N/n^d \rfloor \cdot m^d$ . After balancing each  $S$ , either  $L = \lfloor P/m^d \rfloor$  or  $L = \lceil P/m^d \rceil$ . The estimation for  $L$  is concluded directly.  $\square$

By Lemma 3, to solve the TD problem on  $M(n, d)$  1-exactly, it is sufficient to solve the problem locally in each  $S(2dT, d)$ . Here, the weakly adaptive algorithm described in Section 3.2 could be used locally. However, this direct application results in a running time that becomes quite large. Therefore, we use a more complicated procedure to solve the TD on  $M(n, d)$  1-exactly, which is described in the following algorithm.

#### Algorithm Local-TD ( $T$ )

1. For each  $S(2dT, d)$ : Determine the local maximum load  $k_s$  and broadcast this value to all processors in  $S(2dT, d)$ .
2. Consider a partition of each  $S(2dT, d)$  into submeshes  $S(2 \cdot (2dT/k_s), d)$ . Balance the tokens in each  $S(2 \cdot (2dT/k_s), d)$  with discrepancy 1 by using one of the weakly adaptive algorithms described in Section 3.2.
3. **for**  $l := 1$  **to**  $\log k_s - 1$  **do**  
 Combine  $2d$  neighboring balanced  $S(2^l \cdot (2dT/k_s), d)$  to one  $S(2^{l+1} \cdot (2dT/k_s), d)$  and reduce the discrepancy between all  $S(2^l \cdot (2dT/k_s), d)$  of each  $S(2^{l+1} \cdot (2dT/k_s), d)$  in the following way:
  - (a) **for**  $p := 0$  **to**  $d - 1$  **do**  
 Determine the discrepancy between the two  $S(2^l \cdot (2dT/k_s), d)$  neighbored in dimension  $p$ , and reduce the discrepancy between these two submeshes to at most 2.  
*Comment:* After (a) is executed, the discrepancy in each  $S(2^{l+1} \cdot (2dT/k_s), d)$  is at most  $d + 1$ .
  - (b) **for**  $p := d$  **downto**  $1$  **do**  
 Distribute the tokens in each  $S(2^{l+1} \cdot (2dT/k_s), d)$ , so that the discrepancy is decreased to at most  $p$ .

LEMMA 4. *Local-TD ( $T$ ) solves the TD problem on  $M(n, d)$  1-exactly in  $O((d \cdot 2^d + d^3 \cdot \log k) \cdot T)$  steps and in  $O(d^3 \cdot \log k \cdot T)$  steps, if  $k \geq 2 \cdot (N/n^d)$ , respectively.*

**PROOF.** To execute Phase 1 a prefix computation and a broadcast have to be performed on a  $d$ -dimensional  $2dT$ -sided mesh. Both of these computations can be realized in  $O(d^2 \cdot T)$  steps by running  $d$  phases in each phase operating on linear arrays of length  $2dT$ . Since in each  $S(2 \cdot (2dT/k_s), d)$  the maximum load is at most  $k_s$ , and since the total number of tokens is at most  $k_s \cdot (2 \cdot (2dT/k_s))^d$ , the (adaptive) algorithm in Phase 2 requires  $O(d \cdot 2^d \cdot T)$  steps and  $O(d^3 \cdot T)$  steps, if  $k \geq 2 \cdot (N/n^d)$ , respectively. Because of Lemma 3, we know that when pass  $l$  of the loop in Phase 3 is executed, the load of an arbitrary processor in each  $S(2^{l+1} \cdot (2dT/k_s), d)$  is at least  $\lfloor N/n^d \rfloor - k_s/2^{l+1}$  and at most  $\lceil N/n^d \rceil + k_s/2^{l+1}$ . Hence, the discrepancy at the beginning of pass  $l$  is  $O(k_s/2^l)$ . Determining the discrepancy in Phase 3(a) can be done in  $O(d^2 \cdot T)$  steps in the same way as in Phase 1. Balancing in Phase 3(a) can be done by moving one token of each processor in  $S(2^l \cdot (2dT/k_s), d)$  to the corresponding processor in the neighbored  $S(2^l \cdot (2dT/k_s), d)$ . Since the discrepancy is at most  $O(k_s/2^l)$ , balancing in Phase 3(a) needs  $O(d \cdot T)$  steps. Thus, Phase 3(a) requires  $O(d^3 \cdot T)$  steps. The distribution of tokens in Phase 3(b) in order to decrease the discrepancy can be implemented in the same way as in the algorithms in Section 3.2. Hence, Phase 3(b) needs  $O(d^3 \cdot T)$  steps. Thus, Phase 3 requires  $O(d^3 \cdot \log k \cdot T)$  steps.  $\square$

In the above algorithm we have assumed that the value of  $T$  is known by each processor in advance. In order to avoid this, we extend our network model by a global bus capable of storing one bit. Each processor is able to write on and to read from this bus in one time step. If the processors write different values, then the bus receives the logical AND of these values. Although the modified model does not rigorously fit into the fixed-connection network model, it is simple to realize in practice. The following algorithm solves the TD problem on  $M(n, d)$  1-exactly in  $O((d \cdot 2^d + d^3 \cdot \log k) \cdot T)$  and in  $O(d^3 \cdot \log k \cdot T)$  steps, if  $k \geq 2 \cdot (N/n^d)$ , respectively, without knowing the value of  $T$  in advance by using the modified network model.

### Algorithm TD

$t := 1.$

**Do**

- $t := 2 \cdot t.$
- Local-TD ( $t$ ).
- Each processor  $P$  with load  $L_P$  writes the following bit on the bus:

$$\text{bus} := \begin{cases} 1 & \text{if } \left\lfloor \frac{N}{n^d} \right\rfloor - 1 \leq L_P \leq \left\lceil \frac{N}{n^d} \right\rceil + 1, \\ 0 & \text{otherwise.} \end{cases}$$

**until** bus = 1.

Note that each processor has to know the total number  $N$  of tokens.

**LEMMA 5.** *Algorithm TD solves the token distribution problem on  $M(n, d)$  1-exactly in  $O((d \cdot 2^d + d^3 \cdot \log k) \cdot T)$  steps and in  $O(d^3 \cdot \log k \cdot T)$ , if  $k \geq 2 \cdot N/n^d$ , respectively.*

PROOF. In TD the algorithm Local-TD is started for  $t = 2, t = 2^2, \dots, t = 2^{\lceil \log T \rceil}$ . Thus, TD requires  $\sum_{i=1}^{\lceil \log T \rceil} O((d \cdot 2^d + d^3 \cdot \log k) \cdot 2^i) = O((d \cdot 2^d + d^3 \cdot \log k) \cdot T)$  time steps and  $\sum_{i=1}^{\lceil \log T \rceil} O(d^3 \cdot \log k \cdot 2^i) = O(d^3 \cdot \log k \cdot T)$  time steps, respectively.  $\square$

This concludes the proof of Theorem 3. Note that with  $O(d \cdot n \cdot \log T)$  extra time, the additional global bus may be eliminated.  $\square$

## References

- [1] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic asynchronous networks. *Proceedings of the 25th ACM STOC*, 1993, pp. 632–641.
- [2] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks and multi-processor coordination. *Proceedings of the 23rd ACM STOC*, 1991, pp. 348–350.
- [3] A. Z. Broder, A. M. Frieze, E. Shamir, and E. Upfal. Near-perfect token distribution. *Proceedings of the 19th ICALP*, 1992, pp. 308–317.
- [4] K. T. Herley. A note on the token distribution problem. *Inform. Process. Lett.*, 38:329–334, 1991.
- [5] J. Jájá and K. W. Ryu. Load balancing and routing on the hypercube and related networks. *J. Parallel Distrib. Comput.*, 14:431–435, 1992.
- [6] M. Klugerman and C. G. Plaxton. Small-depth counting networks. *Proceedings of the 24th ACM STOC*, 1992, pp. 417–428.
- [7] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.
- [8] F. Makedon and A. Symvonis. Optimal algorithms for the many-to-one routing problem on two-dimensional meshes. *Microprocessors and Microsystems*, 17:361–367, 1993.
- [9] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. *Proceedings of the 20th ICALP*, 1993, pp. 398–409.
- [10] D. Peleg and E. Upfal. The generalized packet routing problem. *Theoret. Comput. Sci.*, 53:281–293, 1987.
- [11] D. Peleg and E. Upfal. The token distribution problem. *SIAM J. Comput.*, 18:229–243, 1989.
- [12] C. G. Plaxton. Load balancing, selection and sorting on the hypercube. In *Proceedings of the ACM SPAA*, 1989, pp. 64–73.
- [13] K. Qiu and S. G. Akl. Load balancing, selection, and sorting on the star and pancake interconnection networks. *J. Par. Alg. Appl.*, 2:27–42, 1994.
- [14] J. F. Sibeyn and M. Kaufmann. Deterministic 1- $k$ -routing on meshes. *Proceedings of the 11th STACS*, 1994, pp. 237–248.
- [15] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.
- [16] R. Werchner. Balancieren und Selection auf Expandern und auf dem Hyperwürfel. Diplomarbeit, J. W. Goethe-Universität, Frankfurt, Jan. 1991 (in German).