Deterministic Routing of h-relations on the Multibutterfly*

Andrea Pietracaprina
Dipartimento di Matematica Pura e Applicata
Università di Padova
Padova, Italy
andrea@artemide.dei.unipd.it

Abstract

In this paper we devise an optimal deterministic algorithm for routing h-relations on-line on an N-input/output multibutterfly. The algorithm, which is obtained by generalizing the circuit-switching techniques of [3], routes any h-relation with messages of X bits, in $O\left(h(X + \log N)\right)$ steps in the bit model, and in $O\left(h\lceil X/\log N\rceil + \log N\right)$ communication steps in the word model. Unlike other recently developed algorithms, our algorithm does not need extra levels of expanders, hence minimizes the layout area. Moreover, the network topology does not depend on h.

1. Introduction

A communication network can be regarded as a graph whose nodes represent input/output ports or internal switches, and whose edges represent direct links between pairs of nodes. A *routing problem* for such a network is defined as a set of point-to-point messages to be delivered from the inputs to the outputs. A solution to a routing problem requires selecting a path in the network for each message, and scheduling message transmissions along the selected paths. A generic routing problem can be modeled as an h-*relation*, where each input/output in the network sends/receives at most h messages. Although for a number of years the routing literature has focused on the special case of partial permutations (i.e., $h = 1$), more recently, the realization of the advantages gained by the use of batch communication, has shifted attention towards the more general scenario of arbitrary h-relations (see [1] and references therein). The efficient routing of h-relations is crucial for the performance of parallel programs executed on coarse-grained machines. In this respect the h-relation has been introduced by Valiant [12] as a fundamental primitive in the BSP model of parallel computation, which is extensively used for the development of portable parallel software [5]. Moreover, h-relations naturally arise in the concurrent access to shared data in distributed memory systems [13].

There exists a large body of literature on routing algorithms for a variety of interconnections. Most algorithms deal with the special case of 1-relations, although they can be often generalized to run for arbitrary h-relations, through standard techniques. Here, for brevity, we will recall only the results concerning the multibutterfly, which is the network considered in this paper, and refer the reader to [8] for an extensive and detailed account of the routing literature.

The *multibutterfly*, defined by Upfal in [11], is a bounded-degree multi-stage interconnection obtained by superimposing a number of butterflies, and by suitably permuting the edges to achieve certain expansion. The required expansion can be obtained by applying random permutations to the edges, or by using explicit deterministic constructions, which, however, increase the complexity of the network. In its seminal work [11], Upfal devised a deterministic algorithm to route 1-relations in optimal $O(\log N)$ worst-case time on an N-input/output $N(\log N + 1)$-node multibutterfly. He also showed that by augmenting the network with $\log N$ levels of N-node expanders, $O(\log N)$ time can be attained for routing 1-relations when the network is fully loaded, i.e., when all nodes act as inputs and outputs. The only deterministic algorithm previously known for routing permutations in $O(\log N)$ time on a bounded-degree network reduced the routing problem to sorting and required the AKS-based network [2, 7], resulting in impractical constants for both routing time and network topology. Upfal's algorithms could not be directly generalized to the case of arbitrary h-relations, which, hence, was left open.

Recently, Maggs and Vöcking [9] have shown that the multibutterfly can simulate an AKS network [2] of comparable size with constant slowdown. They also show that h-relations can be routed deterministically in optimal $O(h + \log N)$ time on a network with N inputs/outputs, obtained by augmenting an $N(\log N + 1)$-node multibutterfly

*This research was supported, in part, by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (project GEPPCOM).

with h levels of N nodes each, where consecutive levels are connected by expanders. Each node needs only a constant number of buffers, however the extra levels of expanders make the network topology dependent on the degree h of the relation. In a recent and independent work [6], Herley developed an optimal routing algorithm for h-relations by using only $\log N$ extra levels of expanders, thus avoiding the dependency on h, at the expense, however, of a rather involved protocol.

In all of the aforementioned results, running times are expressed in the *word model*, which assumes that in one step (*word step*) $O(1)$ words can be sent across a link and manipulated at a node. In contrast, the *bit model*, often used in circuit switching, assumes that in one step (*bit step*) only $O(1)$ bits can be sent across each edge, and each node, regarded as a finite automaton, can perform a transition to a new state. The switching capabilities of the multibutterfly have been studied by Arora Leighton and Maggs in [3]. Specifically, they show that a multibutterfly with $N(\log N+1)$ nodes and $\Theta(N)$ inputs and outputs is rearrangeable, in the sense that any set of one-to-one connections between inputs and outputs can be realized through node-disjoint paths. Moreover, they present an on-line deterministic algorithm to establish the paths in $O(\log N)$ steps in the bit model. The algorithm can be employed to route an arbitrary partial permutation optimally in $O(X+\log N)$ bit steps, where X is the maximum bit size of a message.

In this paper, we present a deterministic algorithm for routing h-relations on a multibutterfly with N inputs/outputs and $N(\log N+1)$ nodes. Specifically, we generalize the result of [3] and devise an optimal algorithm that routes any given h-relation along paths of congestion $O(h)$. The congestion bound is enforced by employing a novel weight-based technique that generalizes the one adopted in [3] for attaining node-disjoint paths. Our main result is stated below.

Theorem 1 *Any arbitrary h-relation, with messages of X bits, can be routed in $O\left(h(X+\log N)\right)$ bit steps on an N-input/output multibutterfly, using $\Theta\left(\min\{h, N^{\epsilon}\}\right)$ bits of storage at each node, for any positive constant $\epsilon<1$. In the word model, the routing can be accomplished in $O\left(h\lceil X/\log N\rceil+\log N\right)$ communication steps, using $\Theta\left(\min\{h+\log N, N^{\epsilon}\}\right)$ bits of storage at each node, for any positive constant $\epsilon<1$.*

Our algorithm is simple and attains optimal performance, although we must remark that in the word model we account only for communication steps (i.e., link traversals) which are conceivably more expensive than local computation steps. However, we can show that for long messages ($\Omega\left(\log^2 N\right)$ bits) optimal performance can be attained, in the word model, fully accounting for both communication and local computation.

Unlike the algorithms in [9] and [6], our algorithm attains optimal performance on the standard multibutterfly without requiring additional levels of expanders. As a consequence, our network can be laid out in $\Theta\left(N^2\right)$ area, whereas the networks used in [9] and [6] require $\Theta\left(hN^2\right)$ and $\Theta\left(N^2\log N\right)$ area, respectively. Furthermore, no dependency on h arises in the network topology, and the routing algorithm is fairly simple. Finally, we observe that all paths selected by our algorithm have length $\log N$, while the paths used in the aforementioned algorithms can be of length up to $\Theta(h+\log N)$.

2. Preliminaries

Let N be a power of 2 and let d be an integral constant. An (N,d)-*multibutterfly* [11] consists of $N(\log N+1)$ nodes arranged in N *rows*, numbered from 0 to $N-1$, and $\log N+1$ *columns*, numbered from 0 to $\log N$. A node is identified by a pair (r,c), where r and c denote the node's row and column, respectively. For $0\le c<\log N$, the nodes in columns c and $c+1$ are connected to form 2^c parallel c-*splitters*, namely $S_0^{(c)}, S_1^{(c)}\ldots, S_{2^c-1}^{(c)}$. A c-splitter $S_j^{(c)}$ is a bipartite graph (U, V_u, V_ℓ), where

$$\begin{aligned} U &= \{(r,c) \,:\, \lfloor r/(N/2^c)\rfloor = j\} \\ V_u &= \{(r,c+1) \,:\, \lfloor r/(N/2^{c+1})\rfloor = 2j\} \\ V_\ell &= \{(r,c+1) \,:\, \lfloor r/(N/2^{c+1})\rfloor = 2j+1\}. \end{aligned}$$

The nodes in U are referred to as the *inputs* of the splitter, while the nodes in V_u and V_ℓ are referred to as the *upper* and *lower outputs* of the splitter, respectively. Each node $x\in U$ is connected to d nodes of V_u (*up-neighbors of x*), and to d nodes of V_ℓ (*down-neighbors of x*), while each node in $V_u\bigcup V_\ell$ is connected to $2d$ nodes of U.

A splitter (U, V_u, V_ℓ) has (α,β)-*expansion* if every subset $X\subset U$, with $k=|X|\le\alpha|U|$, is adjacent to at least βk upper outputs and βk lower outputs. The multibutterfly is said to have (α,β)-*expansion* if every component splitter has such expansion. The following technical fact is shown in [3].

Fact 1 *A random splitter has (α,β)-expansion with nonzero probability, as long as d,α and β satisfy*

$$2\alpha\beta<1 \quad \textit{and} \quad d>\beta+1+\frac{\beta+1+\ln(2\beta)}{\ln\frac{1}{2\alpha\beta}}.$$

We define an additional property of a splitter, which is a variant of the r-neighbor property defined in [3]. A splitter (U, V_u, V_ℓ) has (α,δ,R)-*neighbor property* if for every subset $X\subset U$ with $k=|X|\le\alpha|U|$, there are two sets $X_u, X_\ell\subseteq X$ such that $|X_u|\ge\delta k$, $|X_\ell|\ge\delta k$, and each node in X_u (resp., X_ℓ) is adjacent to at least R nodes of V_u

(resp., V_ℓ) that have only one neighbor in X. The multibutterfly is said to have the (α, δ, R)-neighbor property if every component splitter exhibits such property. The following lemma can be proved by slightly modifying the proof of Lemma 5.5 in [3] (see also [4]).

Lemma 1 *A splitter with (α, β)-expansion has the (α, δ, R)-neighbor property with*

$$\delta \geq \frac{2\beta - d - R + 1}{d - R + 1}$$

3. The Routing Algorithm

In this section we present an algorithm for routing an h-relation on an (N, d)-multibutterfly. The input and output ports of the network, which are the sources and destinations of messages, are the nodes in column 0 and column $\log N$, respectively. We assume that the multibutterfly has (α, β)-expansion and the (α, δ, R)-neighbor property, where α, β, δ and R are constant such that $\alpha < 1$, $\beta > 1$, $\delta < 1$ and R is a power of 2 greater than 4. Based on Fact 1 and Lemma 1, a set of parameters that ensures these properties for a randomly-wired multibutterfly is, for example, $d = 40, \alpha = 1/(48e^2), \beta = 24, R = 8$ and $\delta = 1/33$. (No attempt is made here to optimize the parameters.)

As well known, for each message there is a unique logical path that connects its source to its destination, in the sense that the sequence of splitters that the path traverses is uniquely determined by the bits of the destination row. However, there are d distinct edges which a message can choose to move from one column to the next along its path. In our algorithm messages follow their unique paths and, exploiting the expansion properties of the network, select the actual edges they traverse isuch a way to minimize congestion. Let $L = 2/\alpha$. For convenience, we assume that only the input/output ports in rows iL, for $0 \leq i < N/L$ are active, i.e., send/receive messages. The case when all input/output ports are active requires minor modifications that increase the running time by only a constant factor.

Define $\bar{h}$ as the smallest power of 2 greater than $4dh/(R-4)$. Each node (r, c) consists of the following components: $2\bar{h}$ buffers, divided into $\bar{h}$ *upper* and $\bar{h}$ *lower buffers*; four *counters*, denoted by $W_u(r, c)$, $W_\ell(r, c)$, $g'(r, c)$ and $g''(r, c)$; and two *flags* $F_u(r, c)$ and $F_\ell(r, c)$. Buffers are used to store either real messages or special messages called *ghosts*. Specifically, a message/ghost at a node is put in an upper or lower buffer depending on whether it must proceed towards an up-neighbor or a down-neighbor. Ghosts are similar in spirit to Arora-Leighton-Maggs' placeholders, since they are used to trace paths for messages temporarily blocked somewhere in the network. They eventually disappear either because the messages they were tracing paths for have been sent along other paths, or because they are replaced by actual messages. A buffer is *full* if it contains a message or a ghost, and it is *empty* otherwise. A full buffer can be either *alive* or *dead*. A buffer containing a ghost is always alive. A buffer storing a message is alive until the message is forwarded to another buffer. At that time the buffer becomes dead and does not play any further role in the algorithm, i.e., it cannot be reused for storing other messages or ghosts. (Note that the number of buffers required at each node grows linearly with h. We will later indicate how this number can be made independent of h.)

The routing protocol is organized in *Stages*, numbered starting from 0. In Stage i only the nodes in columns $c \leq i + 1$ are active, and, if $i < \log N$, nodes in column $i + 1$ are activated (i.e., start receiving messages) in this stage. Initially, each node $(r, 0)$ partitions its messages (if any) between the upper and lower buffers, according to their destinations, and sets $W_u(r, 0)$ (resp., $W_\ell(r, 0)$) equal to the smallest power of two greater than the number of messages in the upper (resp., lower) buffers. Buffers containing messages are marked full and alive. All other buffers in the network are empty and all other counters are 0. Also, all flags are set to 0.

For $i \geq 0$, the operations of Stage i are organized in two consecutive *Phases*, described below.

Phase 1 The following sequence of steps is executed in parallel in every c-splitter, with $c \leq i$.

1. Every input (r, c) with $F_u(r, c) = 0$ and $W_u(r, c) \leq h$ *selects* R arbitrary up-neighbors and sets $F_u(r, c) = 1$.

2. Substeps $2.1 \div 2.3$ are repeated Z times, for a suitable integer $Z > 1$.

2.1. Every input (r, c) with $F_u(r, c) = 0$ and $W_u(r, c) > h$, sends a request to each up-neighbor, asking *permission* to route a batch of $W_u(r, c)/R$ messages to it.

2.2. Every upper output $(r, c+1)$ grants permission for each batch of size $2^j > h/R$, if and only if it received no other request for a batch of the same size.

2.3. Every input (r, c) with $F_u(r, c) = 0$ that got permissions from at least R up-neighbors *selects* R such neighbors and sets $F_u(r, c) = 1$. It also sends a *cancellation signal* to every up-neighbor, withdrawing the requests sent to it in Step 2.1.

3. Every input (r, c) that set $F_u(r, c)$ to 1 either in Step 1 or Step 2, assigns each full upper buffer to one of the selected up-neighbors, making sure that each up-neighbor is assigned at most $W_u(r, c)/R$ buffers.

4. Every upper output $(r, c + 1)$ computes $g'(r, c + 1) = \sum_{j\,:\,2^j > h/R} \min\{b_j, 1\}2^j$, where b_j denote the number of batches of size 2^j for which it received requests in the last it-

eration of Step 2 and for which no cancellations were given.

5. If $i < \log N$, each upper output $(r, i+1)$ of an i-splitter computes in $g''(r, i+1)$ the total number of messages held by the inputs of the splitter, which reside in buffers assigned to it in Step 3. It then fills $g'(r, i+1) + g''(r, i+1)$ empty upper buffers and $g'(r, i+1) + g''(r, i+1)$ empty lower buffers with ghosts, marking such buffers full.

(Steps $1 \div 5$ are repeated with respect to $F_\ell(r, c)$, $W_\ell(r, c)$, the lower buffer and the down neighbors.)

Phase 2

1. Substeps $1.1 \div 1.2$ are executed Y times, for a suitable integer $Y > 1$, in parallel in every c-splitter, with $c \leq i$.

1.1. Every input (r, c) with $F_u(r, c) = 1$ sends each message stored in a full and live upper buffer to the up-neighbor assigned to the buffer, and marks the buffer dead.

1.2. Every upper output $(r, c+1)$ stores each newly received message in a full buffer, upper or lower according to the message's destination, replacing a ghost. Moreover, it updates $g''(r, c+1)$ to account for the current number of ghosts held by the inputs of the splitter, which reside in buffers assigned to it. It then cleans up its buffers maintaining exactly $g'(r, c+1) + g''(r, c+1)$ ghosts in both upper and lower buffers, possibly deleting ghosts in excess of this number. For each ghost deleted, the corresponding buffer becomes empty.

2. For $i < \log N$, every node $(r, i+1)$, sets $W_u(r, i+1)$ (resp., $W_\ell(r, i+1)$) to the smallest power of 2 greater than the number of full upper (resp., lower) buffers.

(Steps 1 and 2 are repeated with respect to $F_\ell(r, c)$, the lower buffer and the down-neighbors.)

Note that at the end of Stage i, only nodes in column c, with $0 \leq c \leq i+1$ may have full buffers. For any such node (r, c), the number of ghosts in its full buffers is equal to $g'(r, c) + g''(r, c)$, where $g'(r, c)$, computed in Phase 1, represents the number of ghosts that the node must keep to trace the paths for messages possibly coming from neighbors in column $c-1$ which have their flags still set to 1, while $g''(r, c)$, computed in Phase 2, accounts for ghosts, hence future messages, coming from neighbors in column $c-1$ which have their flags set to 1. Since the destinations of messages for which the ghosts trace paths are not known, the node must keep the same number of ghosts in both upper and lower buffers.

4. Analysis

Observe that for a node (r, i), the counters $W_u(r, i)$ and $W_\ell(r, i)$, which are initially set to 0, are updated *only* in Stage $i-1$ (Phase 2, Step 2). In what follows, we will use $W_u(r, i)$ and $W_\ell(r, i)$ to denote the updated value after Stage $i-1$. It can be seen that such counters provide an upper bound to the number of messages/ghosts that will ever reside in the node's upper and lower buffers, respectively. For every i-splitter S, let us define

$$\begin{aligned} W_u(S) &= \sum_{(r,i)\in S} W_u(r, i) \\ W_\ell(S) &= \sum_{(r,i)\in S} W_\ell(r, i). \end{aligned}$$

For every $k \geq i$ and every input (r, i) of S, we use $F_u^{(k)}(r, i)$ and $F_\ell^{(k)}(r, i)$ to denote the values of flags $F_u(r, i)$ and $F_\ell(r, i)$, respectively, at the beginning of Stage k. Moreover, we define

$$\begin{aligned} S_u^{(k)}(x) &= \left\{(r, i) \in S \ : \ F_u^{(k)}(r, i) = 0 \text{ and } W_u(r, i) = x\right\} \\ S_\ell^{(k)}(x) &= \left\{(r, i) \in S \ : \ F_\ell^{(k)}(r, i) = 0 \text{ and } W_\ell(r, i) = x\right\}. \end{aligned}$$

Since at the beginning of Stage i all flags of nodes in column i are 0, we have $W_u(S) = \sum_{x>0} |S_u^{(i)}(x)|x$ and $W_\ell(S) = \sum_{x>0} |S_\ell^{(i)}(x)|x$. It is important to observe that if $S_u^{(k)}(x)$ and $S_\ell^{(k)}(x)$ are empty for every splitter in the network and every $x > 0$, then each message can proceed to its destination without being blocked.

The analysis of the algorithm presented in the previous section, relies on the following technical lemmas, whose proofs are omitted here, for brevity, and can be found in [10].

Lemma 2 *Let S be an i-splitter. For every $x > 0$ and $k \geq i$*

$$\begin{aligned} |S_u^{(k)}(x)| &\leq \rho^{k-i}|S_u^{(i)}(x)| \\ |S_\ell^{(k)}(x)| &\leq \rho^{k-i}|S_\ell^{(i)}(x)|, \end{aligned}$$

where $\rho = (1-\delta)^Z$.

Lemma 3 *For every node (r, i), we have $W_u(r, i) \leq \bar{h}$ and $W_\ell(r, i) \leq \bar{h}$.*

The next theorem is a consequence of the above lemmas.

Theorem 2 *Let $Z \geq \log_{1/(1-\delta)}(32d/R)$ and $Y > 2$. Then, $O(\log N)$ stages are sufficient to deliver all messages to the destinations. Moreover, no more than $2\bar{h}$ messages are routed through the same node.*

Proof: Observe that for every i-splitter S and every $x > 0$ $|S_u^{(i)}(x)|, |S_\ell^{(i)}(x)| \leq N/2^i$. By Lemma 2, we have that at the beginning of Stage k, with $k = i + 1 + \log(N/2^i)/\log(1/\rho) = O(\log N)$, both $S_u^{(k)}(x)$ and

$S_\ell^{(k)}(x)$ are empty. Thus, after $O(\log N)$ stages every message can proceed to its destination without being blocked, and at most $\log N$ additional stages are sufficient bring all messages to their destinations in column $\log N$. The congestion bound is an immediate consequence of Lemma 3 and the observation at the beginning of the section. □

We are now ready to prove the main result stated in the introduction as Theorem 1

Proof of Theorem 1: (*Sketch*) We consider only the case when one every $L = 2/\alpha$ inputs/outputs send/receive messages. The general case requires trivial modifications which only increase the running time by a constant factor. The correctness of the algorithm is easy to establish. As for the running time we will first analyze it in the bit model. Observe that in a stage a link is traversed by $O(h)$ messages and a constant number of values representable with $O(\log h)$ bits, while each node has to inspect $O(h)$ bits and perform a constant number of arithmetic operations on values representable with $O(\log h)$ bits. Clearly, the local operations can be accomplished in $O(h)$ bit steps, however, since messages are $X + \log N$ bits long, their movement across the links may take up to $O\left(h(X + \log N)\right)$ bit steps per stage, which is too much for our purposes. In order to attain the stated bit complexity we make a simple modification to the algorithm, preserving its correctness, so that messages traverse the network in a worm-like fashion. Specifically, in Step 1.1 of Phase 2, where message transmissions take place, we forward only two bits of each message, starting from the address bit needed to decide the next transition of the message. A buffer is declared dead only when the last bit of the message it stores is forwarded. Note that buffers of $O(1)$ bits are sufficient. It is easy to see that the new protocol is correct and that, based on Theorem 2, after $O(\log N)$ stages the first bit of each message reaches its destination. At this point, $O(X)$ additional stages are sufficient to deliver the entire message. Since each stage takes $O(h)$ bit steps, it follows that the algorithm takes $O\left(h(\log N + X)\right)$ bit steps, overall. Moreover, only $\Theta(h)$-bit storage is needed at each node.

Consider now the word model with $O(\log N)$-bit words. By suitably packing the bits transmitted over each link into words, we can ensure that at most $O\left(h/\log N\right)$ word steps are taken by link transfers in each stage. It is then easy to argue the algorithm takes $O\left(h\lceil X/\log N\rceil + \log N\right)$ word steps for communication, which is optimal. When $X = \Omega\left(\log^2 N\right)$ we can attain optimal local computation time, by first selecting the paths for the messages (in $O(h \log N)$ word-steps) and then delivering all messages along the selected paths as streams of words. Details are provided in [10]

Note that the storage required at each node is a linear function of h. In order to remove the dependency on h in the network, and use only $O(N^\epsilon)$-bit storage at each node, as claimed in the theorem, for any constant $0 < \epsilon < 1$, we can use the following strategy (a similar strategy is adopted in [9] for different purposes). We partition the nodes in column $\log N$ into N^ϵ groups of $N^{1-\epsilon}$ nodes each. The partition induces a partition of the rows in N^ϵ blocks. We pack together messages destined to the same group of nodes and deliver them to arbitrary nodes within the group. The packing induces an $O(N^\epsilon)$-relation with larger messages that can be routed using only $O(N^\epsilon)$-bit storage at each node. The same strategy is then applied recursively within each block of rows, to complete the routing. □

Acknowledgments The author is grateful to Alessio Gianelle, Kieran Herley and Geppino Pucci for many helpful discussions and constructive comments on this work.

References

[1] M. Adler, J. Byers, and R. Karp. Scheduling parallel communication: The h-relation problem. In *Proc. of the 20th International Symp. on Mathematical Foundations of Computer Science*, LNCS 969, pages 1–20, 1995.

[2] M. Ajtai, J. Komlòs, and E. Szemerèdi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, 1983.

[3] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a nonblocking network. *SIAM Journal on Computing*, 25(3):600–625, June 1996.

[4] A. Gianelle. Instadamento di messaggi in una multibutterfly. Undergraduate Thesis, Dipartimento di Matematica, Università di Padova, Feb. 1997.

[5] M. Goudreau, J. Hill, W. McColl, S. Rao, D. Stefanescu, T. Suel, and T. Tsantilas. A proposal for the BSP worldwide standard library. Technical report, Oxford University Computing Laboratory, Wolfson Building, Parks Rd., Oxford OX1 3QD, UK, 1996.

[6] K. Herley. A note on h-relation routing on the multi-butterfly. ESPRIT-9072 GEPPCOM Report, 1997.

[7] F. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Trans. on Computers*, C-34(4):344–354, Apr. 1985.

[8] F. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA, 1992.

[9] B. Maggs and B. Vöcking. Improved routing and sorting on multibutterflies. In *Proc. of the 29th ACM Symp. on Theory of Computing*, pages 517–530, May 1997.

[10] A. Pietracaprina. Deterministic routing of h-relations on the multibutterfly. Technical Report TR-9/97, Dipartimento di Matematica, Università di Padova, Padova, Italy, Sept. 1997.

[11] E. Upfal. An $O(\log N)$ deterministic packet-routing scheme. *Journal of the ACM*, 39(1):55–70, Jan. 1992.

[12] L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, Aug. 1990.

[13] L. Valiant. General purpose parallel computing. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, Vol. A, Ch. 18, pages 944–996. Elsevier, NL, 1990.