Improved Routing and Sorting on Multibutterflies

Bruce M. Maggs[1]
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
bmm@cs.cmu.edu

Berthold Vöcking[2]
International Computer
Science Institute
Berkeley, CA 94704-1198
voecking@icsi.berkeley.edu

Abstract

This paper shows that an N-node AKS network (as described by Paterson) can be embedded in a $\frac{3N}{2}$-node twinbutterfly network (i.e., a multibutterfly constructed by superimposing two butterfly networks) with load 1, congestion 1, and dilation 2. The result has several implications, including the first deterministic algorithms for sorting and finding the median of $n \log n$ items on an n-input multibutterfly in $O(\log n)$ time, a work-efficient deterministic algorithm for finding the median of $n \log^2 n \log\log n$ items on an n-input multibutterfly in $O(\log n \log\log n)$ time, and a three-dimensional VLSI layout for the n-input AKS network with volume $O(n^{3/2})$. While these algorithms are not practical, they provide further evidence of the robustness of multibutterfly networks. We also present a separate, and more practical, deterministic algorithm for routing h-relations on an n-input multibutterfly in $O(h + \log n)$ time. Previously, only algorithms for solving h one-to-one routing problems were known. Finally, we show that a twinbutterfly, whose individual splitters do not exhibit expansion, can emulate a bounded-degree multibutterfly with (α, β)-expansion, for any $\alpha \cdot \beta < 1/4$.

[1]Bruce Maggs is supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contracts F196828-93-C-0193 and F19623-96-C-0061, by ARPA Contract N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute and Sun Microsystems. This research was conducted in part while he was visiting the Heinz Nixdorf Institute, University of Paderborn, Germany, with support provided by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen".

[2]Berthold Vöcking is supported by a DAAD postdoc fellowship. This research was conducted in part while he was staying at the Heinz Nixdorf Institute, with support provided by DFG-Sonderforschungsbereich 376 and EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

1 Introduction

In 1983, Ajtai, Komlós, and Szemerédi (AKS) devised a network for sorting n items in $O(\log n)$ depth [1]. This result was surprising because no improvement in the asymptotic depth of sorting networks had been made since Batcher's invention of the $O(\log^2 n)$-depth bitonic sorting network 15 years earlier [5]. Indeed, the difficulty of improving on Batcher's construction led Knuth to conjecture that there was no sorting network with depth $O(\log n)$ [25, p. 243].

The AKS sorting network differed from previous constructions in one crucial respect: it incorporated *expansion* into its structure. Expansion is a graph-theoretic notion. An $l \times r$ bipartite graph is said to have (α, β)-*expansion* if every set of x nodes on the left side has at least βx neighbors on the right side, provided that $x \leq \alpha l$, where α and β are constants, $\alpha < 1$, and $\beta > 1$. This property is most interesting when $r \leq l$, for when $r \gg l$, it is easy to construct graphs with expansion. As it happens, a random k-regular $l \times l$ bipartite graph is likely to be an expander for any $k \geq 3$, [44]. Explicit constructions were first discovered by Margulis [34, 35], and have since been greatly improved. So far, however, the expansion achieved by the explicit constructions is still about a factor of two smaller than the expected expansion of a random graph. A nice summary of the state of the art in expander graphs can be found in [23].

One drawback to the AKS network is that the big-O notation hides large constant factors. In contrast, the depth of the bitonic sorting network is $(\log^2 n)/2 + (\log n)/2$ [14, p. 650]. Some progress has been made in simplifying the AKS network and in improving the constant factors in its depth [42], but for practical values of n, the depth of bitonic sort is much smaller. To date, however, all $O(\log n)$-depth sorting networks are based on the AKS construction.

Two notable AKS-based sorting networks are Leighton's sorting network [27], and Ma's fault-tolerant sorting network [32]. Leighton showed how to construct an N-node degree-3 network capable of sorting N items in $O(\log N)$ steps. His network implements the *columnsort* algorithm, and uses a $\Theta(N/\log N)$-input AKS network in a pipelined fashion. Ma showed how to construct an n-input sorting network with $O(\log n)$ depth that can sustain constant-probability *passive* faults at its comparators, and still sort correctly with high probability. In the passive fault model, a faulty-comparator can be viewed as having been removed from the network.

Another network that incorporates expansion into its structure is the *multibutterfly*. The basic structure of this network was introduced by Bassalygo and Pinsker [4], who showed that two back-to-back multibutterflies form an $O(\log n)$-depth nonblocking network. Here n is the number of input and output terminals of the network. A network is called *nonblocking* if every unused input terminal can be connected by a path through unused edges (or nodes) to any unused output terminal, regardless of which inputs and outputs have already been connected. Bassalygo and Pinsker did not use the term multibutterfly, and their network differed from the multibutterflies considered in the rest of this paper in one technical detail: although the out-degree of each node in the network was bounded, the in-degree was not necessarily so. It is not difficult, however, to modify their construction so that the degree of all nodes is bounded.

The term "multibutterfly" was introduced by Upfal [51]. In his seminal paper, Upfal proved that an n-input multibutterfly can route any permutation of n packets from the inputs to the outputs of a multibutterfly in $O(\log n)$ steps deterministically. (In fact, he showed that even a collection of $\log n$ permutations can be routed in $O(\log n)$ time.) Because it can sort, the AKS network can also solve these problems in $O(\log n)$ time. In the AKS network, however, the running time of the algorithm cannot be separated from the size and depth of the network. In the multibutterfly, on the other hand, although the $O(\log N)$ bound on the running time hides some moderately large constants, the network itself can be constructed by merging just two copies of the ordinary butterfly network (hence the name multibutterfly). Furthermore, simulations show that the running time of the routing algorithm is actually smaller than the $O(\log N)$ upper bound implies [29, 31]. Hence, a case can be made for the practicality of multibutterflies, and several studies have explored their implementation [12, 13, 16, 17].

Although no deterministic $O(\log n)$-step sorting algorithm for multibutterflies was previously known, the network was known to have some capabilities that the AKS network was not known to have. For example, Leighton and Maggs showed that multibutterflies are highly fault tolerant [29]. In particular, they showed that even if an adversary is permitted to place f worst-case *fail-stop* faults in a multibutterfly, there is still some set of $n - O(f)$ inputs and $n - O(f)$ outputs between which any permutation of packets can be routed in $O(\log n)$ steps. In the fail-stop fault model, a faulty node cannot communicate with its neighbors at all. As a consequence, fail-stop faults are more difficult to tolerate than passive faults. Leighton and Maggs also showed that even if every node in the network fails with some small, but constant, probability, with high probability there is still some set of $\Theta(n)$ inputs and $\Theta(n)$ outputs between which any permutation can be routed in $O(\log n)$ time. As Bassalygo and Pinsker showed, the multibutterfly can also be used to construct a nonblocking network. Arora, Leighton, and Maggs termed two back-to-back multibutterflies a multi-Beneš network, and showed that not only is a multi-Beneš network nonblocking, but any set of new paths can be established in this network in $O(\log n)$ steps, even if many requests for new paths are made simultaneously [2]. The algorithms for reconfiguring a multibutterfly with faults and for establishing disjoint paths were later improved in [20] and [45], respectively.

1.1 Our results

In this paper, we show that multibutterfly networks are at least as powerful as the AKS sorting network. In particular, we show that an N-node AKS network can be embedded in a $\frac{3N}{2}$-node twinbutterfly (i.e., a multibutterfly constructed by superimposing two butterfly networks) with load 1, congestion 1, and dilation 2. As a consequence, an N-node twinbutterfly can emulate an N-node AKS network with constant slowdown.

The embedding has several other immediate implications. The emulation of the AKS network by the twinbutterfly, along with Leighton's columnsort algorithm [27], yields the first deterministic $O(\log N)$-step algorithm for sorting N items on an N-node twinbutterfly. The sorting algorithm can then be used to construct the first deterministic $O(\log N)$-step algorithms for finding the median of N items and for routing with combining on multibutterflies. It also yields a work-efficient deterministic algorithm for finding the median of $N \log N \log\log N$ items in $O(\log N \log\log N)$ time on an N-node twinbutterfly. Because the embedding of the AKS network into the twinbutterfly has constant load and congestion, bounds on the VLSI layout area and volume for the multibutterfly translate to the AKS network as well. An n-input multibutterfly network can be laid out in two dimensions with area $O(n^2)$, and in three dimensions with volume $O(n^{3/2})$, and these bounds are tight. The two-dimensional layout area of the AKS network was known before [9, 10], but the three-dimensional layout is new.

We also present a deterministic algorithm for solving h-to-one routing problems on an n-input multibutterfly, augmented by h extra levels, in $O(h + \log n)$ time. Previous routing algorithms could solve h one-to-one problems in a pipelined fashion [29, 51], but assumed that each packet carried the label of the one-to-one problem to which it belonged. An algorithm for solving h-to-one routing problems can also be used to route h-relations. In an h-relation, each source sends at most h packets, and each destination receives at most h packets. One motivation for designing algorithms that route h-relations is that routing an h-relation is the primitive communication step in the BSP model of computation [52], for which there are growing libraries of parallel programs [6, 21, 36, 41].

Finally, we show that a twinbutterfly whose individual splitters do not exhibit expansion can emulate a bounded-degree multibutterfly with an (α, β)-expansion property, for any $\alpha \cdot \beta < 1/4$.

The fact that an N-node multibutterfly network contains an N-node AKS network does not imply that the multibutterfly is an inherently impractical network. Although the sorting algorithm implied by the embedding is not practical, there is no requirement that the multibutterfly be used in this fashion. Indeed, independent of the sorting algorithm, the multibutterfly is an efficient and highly fault-tolerant routing network.

1.2 Other related results

Prior to this work, the fastest deterministic algorithm for sorting N items on an N-node multibutterfly was the Sharesort algorithm of Cypher and Plaxton [15]. This algorithm was designed to run on the butterfly network, or on any other hypercubic network (e.g., the shuffle-exchange network and the hypercube). Since the multibutterfly network contains a butterfly network, it applies to multibutterflies as well (but doesn't take advantage of the expansion in the multibutterfly). There are several variants of this algorithm. The fastest uniform version runs in $O(\log N(\log\log N)^2)$ time, but there is a non-uniform version that runs in $O(\log N \log\log N)$ time. Our embedding result yields an $O(\log N)$-time algorithm for the multibutterfly. Note that the sorting problem can also be solved on an N-node butterfly (or multibutterfly) in $O(\log N)$ time using the randomized Flashsort algorithm of Reif and Valiant [30, 49].

Prior to this work, the fastest deterministic selection algorithm for multibutterflies was the algorithm of Berthomé, Ferreira, Maggs, Perennes, and Plaxton [7]. This algorithm selects the kth largest item from among N items on an N-node butterfly (or any other hypercubic network) in $O(\log N \log^* N)$ time. Like the Sharesort algorithm, this algorithm does not make use of expansion when run on a multibutterfly. Since the selection problem can be solved in linear time sequentially [11], this algorithm, which performs $N \log N \log^* N$ work, is not work efficient. Furthermore, Plaxton [46] showed that any deterministic algorithm for solving the selection problem on a N-node hypercubic network requires $\Omega((M/N)\log\log N + \log N)$ time in the worst case, where M is the number of input items. This translates to a lower bound of $\Omega(M \log\log N + N \log N)$ on the work required. Hence, there can be no deterministic work-efficient selection algorithm on a hypercubic network. (This lower bound does not apply, however, to multibutterlies.) Recently, Plaxton showed that for $M/N = \log N$, any deterministic algorithm for selection on a bounded-degree N-node hypercubic network requires $\Omega(\log^{3/2} N)$ steps [47]. He also presents an algorithm that runs in $O(\log^{3/2} N(\log\log N)^2)$ time on any N-node hypercubic network.

For bounded-degree expander-based networks, two optimal deterministic algorithms for selection are known. For the case of finding the kth largest out of N items on an N-node network, the AKS sorting network combined with columnsort can be used to sort the items (and hence solve the selection problem) in $O(\log N)$ time [27]. This algorithm is optimal because selection on any bounded-degree N-node network requires $\Omega(\log N)$ time. The kth largest of M items, $M \geq N$, can be found in $O((M/N) + \log N \log\log(M/N))$ time on an N-node expander-based network using an implementation of a PRAM algorithm due to Vishkin [53] that invokes the AKS sorting network and columnsort as subroutines [46]. This algorithm is work-optimal for $M/N \geq \log N \log\log(M/N))$. Our embedding result implies that a multibutterfly network can perform both of these algorithms. Note that the latter algorithm beats Plaxton's lower bound for hypercubic networks, thus implying a separation in power between expander-based networks and hypercubic networks. Rappoport [48] has recently proved an even larger separation, namely that the largest butterfly that can efficiently emulate an N-node multibutterfly has fewer than N^ϵ nodes, for all constants $\epsilon > 0$. For $\omega(1) \leq M/N \leq o(\log N \log\log(M/N))$ the asymptotic complexity of selection on bounded-degree networks is currently not known.

Recently, Herley and Pietracaprina improved on our result on routing h-relations on the multibutterfly network with n input nodes. Herley [22] shows how the dependence of the network on h can be eliminated by using only $\log n$ instead of h extra levels, at the expense, however, of a rather involved protocol. Pietracaprina [43] presents an algorithm that does not need any extra levels but requires an $\Theta(\min\{h + \log n, \sqrt{n}\})$-bit storage at each node.

1.3 Outline

The remainder of this paper is organized as follows. In Sections 2 and 3 we define the multibutterfly and AKS networks, respectively. Our embedding of an AKS network into a twinbutterfly network is presented in Section 4. Algorithms for routing h-relations on multibutterflies are described in Section 5. In Section 6 we show that a twinbutterfly can emulate a multibutterfly with (α, β)-expansion. We conclude in Section 7 with some open problems.

2 Multibutterfly networks

A *d-dimensional multibutterfly network (MBF)* consists of $d+1$ *levels*, each consisting of 2^d nodes. We view these levels as being stacked vertically, with level 0 at the top, and level d at the bottom. For $0 \leq \ell \leq d$ and $0 \leq j \leq 2^d - 1$, let (ℓ, j) be the label of the jth node on level ℓ. Within a level, we view the nodes as being arranged from left to right in order of increasing labels. The nodes on level 0 of a d-dimensional multibutterfly are called *input nodes*, and the nodes on level d are called *output nodes*.

The nodes on each level ℓ are partitioned into 2^ℓ sets $A_{\ell,0}, \ldots, A_{\ell,2^\ell-1}$, where

$$A_{\ell,i} := \left\{ (\ell, j) \;\middle|\; \lfloor j/2^{d-\ell} \rfloor = i \right\} \quad .$$

The nodes in $A_{\ell,i}$ are connected to the nodes in $A_{\ell+1,2i}$ and $A_{\ell+1,2i+1}$. The subgraph induced by the nodes in these three sets is called the *splitter of* $A_{\ell,i}$. It consists of two *concentrators*, a *left* one and a *right* one. The left concentrator is defined as the subgraph induced by the nodes in $A_{\ell,i}$ and $A_{\ell+1,2i}$, and the right concentrator is defined as the subgraph induced by the nodes in $A_{\ell,i}$ and $A_{\ell+1,2i+1}$. All edges of a multibutterfly network are inside its concentrators, i.e., each concentrator is a bipartite graph $G = (A \cup B, E)$ with $A = A_{\ell,i}$ and $B = A_{\ell+1,2i}$ or $B = A_{\ell+1,2i+1}$, for $0 \leq \ell \leq d-1$. The edges in a concentrator can be chosen in an arbitrary fashion, provided that each node in A has degree k, and each node in B has degree $2k$, for some constant integer k. This defines a multibutterfly of degree $4k$.

The multibutterfly structure is very similar to that of the butterfly network. The *d-dimensional butterfly* consists of $d+1$ levels each of which includes $n = 2^d$ nodes. Each node has a distinct label (ℓ, w) where ℓ is the level of the node ($0 \leq \ell \leq d$) and w is a d-bit binary number that denotes the *column* of the node. All nodes of the form (ℓ, w), $0 \leq \ell \leq d$, are said to belong to column w. Two nodes (ℓ, w) and (ℓ', w') are linked by an edge if $\ell' = \ell + 1$ and either w and w' are identical or w and w' differ only in the bit in position ℓ', where the bit positions are numbered 1 through d, the most significant bit being numbered 1.

The butterfly network is a special case of the multibutterfly of degree 4. The basic advantage of the multibutterfly compared to the butterfly is that the multibutterfly may satisfy some expansion properties if the edges inside the concentrators are chosen properly. Let $\Gamma(X)$, for a subset of nodes X, denote the set of the neighbors of the nodes in X. Then we say a concentrator $G = (A \cup B, E)$ has (α, β)-*expansion* if for any set $X \subseteq A$ with $|X| \leq \alpha|A|$, we have $|\Gamma(X)| \geq \beta|X|$. A multibutterfly is said to have (α, β)-expansion if all of its concentrators have (α, β)-expansion. Upfal [51] shows that for any k, α, and β with $2\beta < k - 1$, and $\alpha < (2\beta)^{-1}(2\beta e^{1+2\beta})^{-1/(k-2\beta-1)}$, there exists a multibutterfly of degree $4k$ with (α, β)-expansion. Note that, for very small α, the bound implies that β approaches $k - 1$, which means that the expansion is nearly optimal (k would be optimal), and that for sufficiently large k, the product $\alpha\beta$ approaches $1/2$, which is the largest possible value, because otherwise $\alpha|X|$ nodes would expand to more than $|X|/2$ nodes.

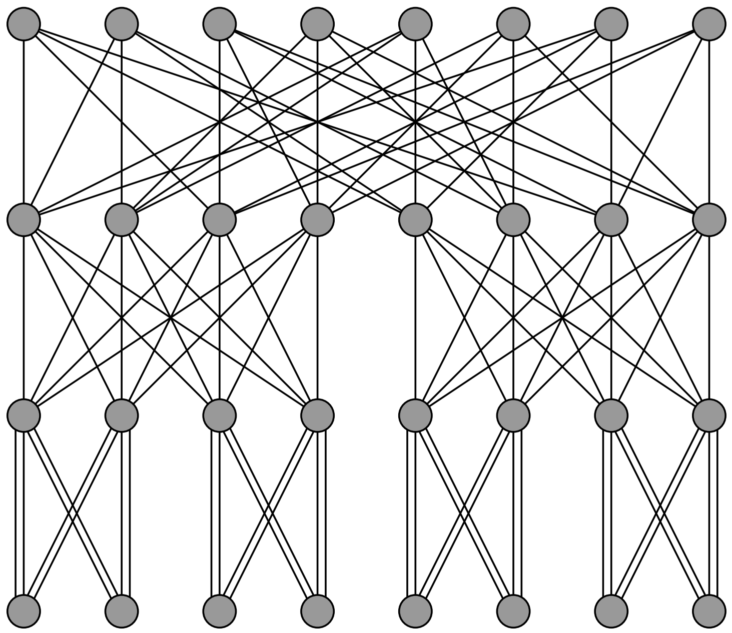

Figure 1: Example of a 3-dimensional twinbutterfly.

Finally, we define a subclass of the multibutterfly networks that includes those multibutterflies that can be constructed by superimposing butterfly networks. Suppose the edges of a d-dimensional multibutterfly of degree $4k$ can be colored by k colors such that the network induced by the edges of each color are isomorphic to the d-dimensional butterfly. Then this multibutterfly is called a *k-folded butterfly* since it can be constructed by folding k butterfly networks. By folding, we mean that the labels of the nodes in the $A_{\ell,i}$ sets of each of these butterflies are permuted and the k nodes with the same label in distinct butterflies are merged together to form a multibutterfly node. The k butterfly networks that define a k-folded butterfly are called *underlying butterflies* and we denote them by $BF_1, \ldots, BF_k$. A 2-folded butterfly is also called a *twinbutterfly*. Figure 2 gives an example of a 3-dimensional twinbutterfly.

The twinbutterfly that we use for the embedding of the AKS network (Section 4) and the multibutterfly that we use for the routing of h-relations (Section 5) have a special feature, which is, all of the splitters on any particular level are identical. However, the twinbutterfly in which we embed a multibutterfly with expansion (Section 6) does not have this feature.

3 The AKS network

Our description of the AKS network is based on Paterson's description [42]. Ours is a little more general than Paterson's because we do not describe the building blocks, i.e., the separators and sorters, in detail.

The AKS network is a sorting network that consists of $h \cdot T$ rows that are partitioned into $T = O(\log n)$ stages of width n and of constant height h. By *width* n we mean that each row contains n nodes, and by *height* h we mean that each stage consists of h rows. Let

$$V_t := \{(j + t \cdot h, i) \mid 0 \leq j \leq h - 1, 0 \leq i \leq n - 1\}$$

be the set of nodes on stage t, for $0 \leq t \leq T - 1$. Then each node (j, i) is connected via a *forward edge* to node $(j+1, i)$, for $0 \leq j \leq h \cdot T - 2$ and $0 \leq i \leq n - 1$. In addition to the forward edges, the network contains *compare-exchange edges* which connect nodes in the same row, i.e., each compare-exchange edge connects a node (j, i) with a node (j, i'), for $0 \leq j \leq h \cdot T - 1$ and $0 \leq i < i' \leq n - 1$. Each node is incident to at most one compare-exchange edge.

The AKS network sorts n items in $2 \cdot h \cdot T - 1 = O(T) = O(\log n)$ steps. Before step 0, the items are located at the nodes in row 0. In each even step, the two items located at the endpoints of each compare-exchange edge are compared, and the items are exchanged if they are in the wrong order. In each odd step, the items are moved along the forward edges to the next row. After step $2 \cdot h \cdot T - 2$, the items are located in sorted order on the nodes in row $h \cdot T - 1$.

Each stage of the AKS network consists of several independent *building blocks*. All of the compare-exchange edges are inside these building blocks. We initially describe the widths of these blocks as if they were real numbers. Ultimately, we will replace these *ideal* values by appropriate integers. Most of the building blocks are *separators*, some are *sorters*, and some are *forward blocks*. We give a brief overview of these blocks without going into details. Each separator partitions its input items into four parts, FL (far-left), CL (center-left), CR (center-right), and FR (far-right). which is described later on in more detail. The sorters return the input items in sorted order. It is convenient to implement the sorters as Batcher's bitonic sorting network [5]. All sorters have constant width, so they can be implemented in constant height h. The forward blocks include only forward edges and no compare-exchange edges.

In the following, we describe the widths of the building blocks and which output partitions of the blocks in stage t are connected to which blocks in stage $t + 1$, for $0 \leq t < T - 1$. Our description is based on an oblivious sorting algorithm structured about a complete binary tree B of depth $\log n$ which we imagine with the root at the top (on level 0) and leaves below (on level $\log n$). The algorithm proceeds in $\log n$ stages of time. Each stage of the AKS network implements the operations specified for the corresponding stage in the algorithm.

The underlying binary tree B has a "bag" at each node. Initially, the set of n items to be sorted is contained in the bag at the root. We first show how the items can be sorted in $\log n$ stages each of which, however, requires more than constant time. During these stages, the items squeeze down the tree. In each stage, each node of the tree with a non-empty bag partitions the items in its bag into two halves, the smaller items, and the larger items, and then sends the smaller items to its left child and the larger items to its right child. Then the items will arrive in sorted order at the leaves of the tree after $\log n$ stages. Unfortunately, it is not possible to split the items exactly into the two halves at each tree node in constant time. The strategy of the AKS algorithm is to make an approximate partition of items such that each stage takes only constant time. The items that are sent to the wrong child in a stage are sent back in later stages.

In the following, we are interested only in the flow of the items between the bags. The proof that the algorithm sorts can be found in Paterson's article [42]. We define the *size of a bag* to be the number of items stored in that bag, and the *capacity of a bag* to be the maximum number of items that can be stored in that bag. During most stages, a bag is either empty or filled to its capacity that is decreasing with time. In particular, the capacity of each bag at level ℓ is $x \cdot a^{\ell}$, for a value of x that is decreasing with the stage number (and will be specified later) and some constant $a > 1$, e.g., $a = 3$.

Special situations occur at the highest and lowest nonempty levels of the tree, so we start with a description of the sorting process at intermediate levels. The algorithm works in T stages beginning with stage 0. In odd stages, some odd levels are full and all the bags at the even levels are empty. The opposite holds in even stages. In each stage, the items in any full bag are partitioned by a separator into the four partitions FL, CL, CR, and FR. The FL and the FR parts are sent back to the parent bag and the CL and CR parts are transferred down to the left and right child bags, respectively. Suppose a bag is filled up to its capacity b. Then the size of FL and FR is $\lambda \cdot b$ and the size of CL and CR is $(1-\lambda) \cdot b$, where, e.g, $\lambda = 1/8$.

Consider a bag with capacity b that is empty at the beginning of some stage and that is filled to its new capacity νb at the end of the stage, as shown in Figure 2. Then

$$\nu b = 2\lambda b a + \frac{(1-\lambda) \cdot b}{2a} \ ,$$

which gives

$$\nu = 2\lambda a + \frac{1-\lambda}{2a} \ .$$

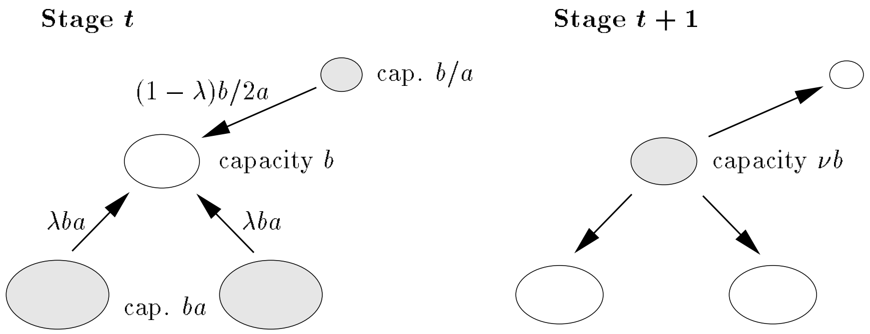

Figure 2: Reduction of bag capacities after each stage.

We assume that $\nu < 1$, e.g., $\nu = 43/48$. Thus, the capacities diminish in each stage and the items are squeezed down the tree in the course of the algorithm. We define the capacity of each bag at level ℓ at the beginning of stage t to be

$$c_\ell(t) \quad := \quad \left(1 - \frac{1}{4a^2}\right) \cdot n \cdot \nu^t \cdot a^\ell \quad . \tag{1}$$

(Hence, the x that should be specified later is $(1 - (1/4a^2)) \cdot n \cdot \nu^t$.)

At the beginning of the algorithm all bags except for the root are empty. The root is filled to its capacity, i.e., it contains $(1 - 1/(4a^2)) \cdot n$ items. Since we would like the root to behave as if it were an ordinary node, we place above it a subset of the items of size $(1/4a^2) \cdot n$. This subset we call the *cold storage*. The root exchanges items with the cold storage as with a parent. Therefore, in odd stages, the capacity of the cold storage is half the root's parent's capacity plus 1/8th the root's grandgrandparent's capacity, and so on. In even stages, its capacity is 1/4th the root's grandparent's capacity, plus 1/16 the root's grandgrandgrandparent's capacity, and so on. This means the capacity of the cold storage in stage t is

$$\frac{1}{2} \cdot c_{-1}(t) + \frac{1}{8} \cdot c_{-3}(t) + \ldots = \frac{n \cdot \nu^t}{2a}$$

if t is odd, and

$$\frac{1}{4} \cdot c_{-2}(t) + \frac{1}{16} \cdot c_{-4}(t) + \ldots = \frac{n \cdot \nu^t}{4a^2}$$

if t is even.

During the course of the algorithm the items migrate down through the tree. We will arrange that there is at most one partially filled level. Above this, the levels are alternately empty and full as already described; below all the levels are empty. To achieve this, we require that at the *partial level* each bag should send up to its parent the normal number of items (i.e., λb with b denoting the bag's capacity) if it has sufficiently many. After this requirement is met, any remaining items can be sent down to its children in equal numbers.

In the final stages, some of the separators are replaced by sorters and forward blocks. In particular, if the capacity of the root bag is smaller than r, for some constant r, e.g., $r = 160$, and the bag is non-empty, then the set of items in the root bag and the cold storage is sorted and separated into a left and right half. From these halves the root and the cold storage for each subtree can be immediately formed. This event is called a *splitting*. A splitting divides the problem into two independent subproblems, each of which has its own root (the left child or right child of the old root) and its own cold storage.

After the first splitting, a new splitting will be required at regular intervals of constant length, i.e., when the capacity of a bag becomes smaller than r, the separator is replaced by a sorter and the items are split into two halves. The stage in which the bags on the ℓth level of the tree are split is called the *ℓth splitting stage.* Note that the number of items split by a sorter is at most r plus the number of items in the cold storage, $r/(4a^2 \cdot (1 - 1/(4a^2)))$, which sums to $r/(1 - (1/4a^2))$. Thus, the width of the sorter doing the splitting is $O(1)$, and its height is $O(1)$, too, in particular the height is at most h, e.g., $h = 36$.

The algorithm finishes after stage $T - 1$, in which the items of the bags on some level are sorted and all bags below this level are empty. Stage t of the algorithm is implemented in stage t of the AKS network. For each stage t of the algorithm, each bag of the tree corresponds to a separator or sorter in stage t of the AKS network. The items in the cold storage do not have to be separated or sorted (except during a splitting stage), so that they can move from stage to stage through forward blocks. The widths of the building blocks correspond to the sizes of the respective bags and cold storages, which can be extracted from the above description. Paterson gives a simple recipe for replacing the real numbers by integers without straying far from the ideal values. For each subtree rooted at a nonempty node, if the ideal total size of the subtree is α, then the actual size is $2\lceil\alpha/2\rceil$.

4 Embedding the AKS network into a multibutterfly

In this section, we embed an AKS network into a multibutterfly network. An embedding maps a guest graph G to a host graph H. Nodes of G are mapped to nodes of H, and edges of G are mapped to paths in H. The *load* of an embedding is the maximum number of nodes of G mapped to any node of H. The *congestion* of an embedding is the maximum number of paths that use any edge in G. The *dilation* of an embedding is the length of the longest path. In general, the smaller the load, congestion, and dilation, the better the embedding. It is not difficult to show that if the load, congestion, and dilation of an embedding are constant, then the host H can emulate the guest G with constant slowdown. Many previous work deals with graph embeddings and network simulations, e.g., [3, 8, 18, 19, 26, 30, 37, 38, 39, 40, 48, 50]. Surveys on these topics can be found, e.g., in [24, 28, 33].

We denote the width of the AKS network by n, the number of stages by T, and the height of each stage by h. We assume that the widths of the building blocks, which are equivalent to the sizes of the bags, are defined by the parameters λ, a, ν, and r, as described in Section 3.

Theorem 4.1 *An AKS network of size N can be embedded into a twinbutterfly of size $M \leq \kappa \cdot N + o(N)$ with load 1, dilation 2, and congestion 1, where κ is a small constant depending on the AKS parameters ν, a, r, and h.*

Suppose that the AKS parameters are chosen according to Paterson's recommendation, which should minimize the size of the AKS network, i.e., $\nu = 43/48$, $a = 3$, $r = 160$, and $h \geq 36$. Then κ is at most $1.351\ldots$. In the following, we describe the embedding and prove the result on the relationship of the network sizes.

4.1 Rough embedding

The description of the AKS network is structured about a binary tree. The nodes of this tree represent bags whose sizes vary from stage to stage, i.e., over time. Instead of looking at one binary tree B with growing and shrinking bag sizes, however, we can imagine that we have T trees $B_0, \ldots, B_{T-1}$ of fixed sized bags, such that the bags in the tth tree represent the building blocks of the tth AKS stage. Figure 3 depicts this abstraction. The size and capacity of each node u in the tree B_t are equal to the size and capacity of the corresponding node u in the binary tree B at stage t. (Recall that the *size of a bag* is defined to be the number of items stored in that bag, and that the *capacity of a bag* is defined to be the maximum number of items that can be stored in that bag.) Hence, each bag of tree B_t with size s is realized as a building block of width s and height h in stage t.

A natural partition of the AKS building blocks is to divide the blocks according to their stages. Then each partition corresponds to one of the T trees. In fact, this partition is the one implemented in the AKS network. For the embedding into the MBF, however, we partition the blocks of the AKS network into sets $P_0, \ldots P_{\log n}$, according to their levels in the tree. This means that partition P_ℓ includes all $2^\ell \cdot T$ building blocks that are associated with some node on the ℓth tree-level in one of the T trees. Let s_ℓ denote the ℓth splitting stage. For $\ell \geq 0$, we add the building blocks of the cold storage in each stage t with $s_\ell \leq t < s_{\ell+1}$ to partition P_ℓ. Besides we add the cold storage in each stage t with $0 \leq t < s_0$ to partition P_0. Define the *size of a partition* P_ℓ to be to the sum of the sizes of all bags on the respective tree level ℓ. This size is denoted by $|P_\ell|$. Note that some bags in each partition have size 0, i.e., all bags below the partial level for any stage t, all bags of odd levels of the tree in even stages, and all bags of even levels of the tree in odd stages.

The AKS building blocks associated with the bags of partition P_ℓ are embedded in the ℓth level of the MBF. Of course, we have to define more precisely which nodes in the building blocks in partition P_ℓ are mapped onto which nodes of the MBF in level ℓ, and how the two butterflies are folded. Divide each partition P_ℓ into equal-sized subpartitions $P_{\ell,0}, \ldots, P_{\ell,2^\ell-1}$

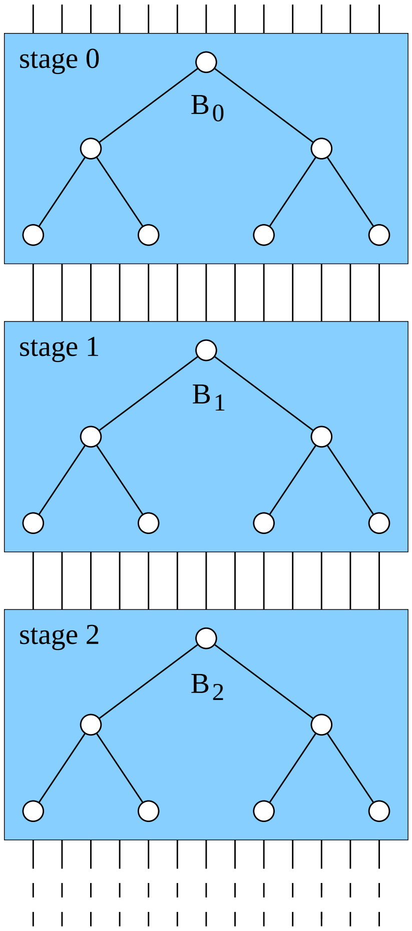

Figure 3: Abstract view on the AKS network. Here each box represents a stage, which implements a set of comparisons (corresponding to the different types of building blocks that we have defined) specified by the algorithm that is described in terms of a binary tree.

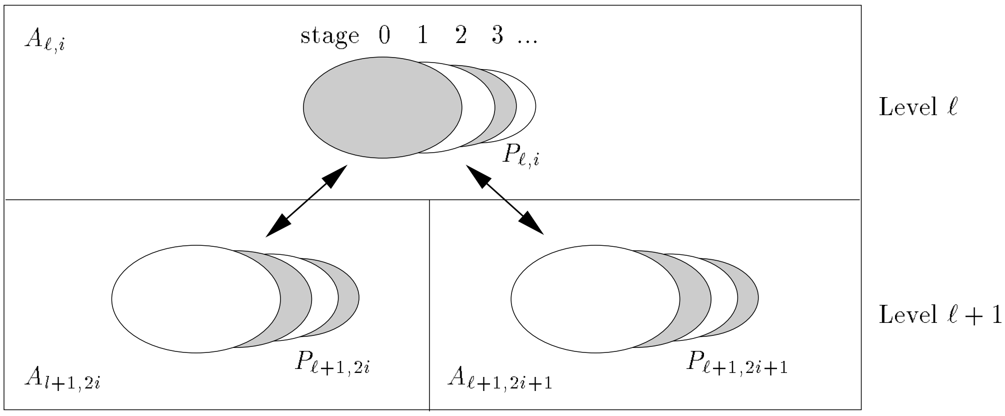

Figure 4: Rough embedding of the AKS network into the multibutterfly.

such that subpartition $P_{\ell,i}$ includes all blocks that correspond to the ith node on level ℓ of the AKS tree B. Then for $0 \le i \le 2^\ell - 1$ and for $0 \le \ell \le \log n - 1$, subpartition $P_{\ell,i}$ includes all parent bags of the bags in the subpartitions $P_{\ell+1,2i}$ and $P_{\ell+1,2i+1}$.

We map the AKS nodes of partition $P_{\ell,i}$ onto the MBF-nodes in set $A_{\ell,i}$. This is illustrated in Figure 4. (Recall that MBF level ℓ is partitioned into 2^ℓ subsets $A_{\ell,0}, \ldots, A_{\ell,2^\ell-1}$.) In order to get an embedding with load 1, it is required that $|A_{\ell,i}| \ge h \cdot |P_{\ell,i}|$ which is the number of nodes represented by the partition $P_{\ell,i}$. It will be seen later that the size of $A_{\ell,i}$ has to be a little bit larger than this value.

Suppose for the moment that we could add all AKS edges to the MBF regardless of the multibutterfly structure, i.e., suppose we could connect each pair of MBF nodes representing a pair of adjacent AKS nodes by an edge. Then each AKS edge that connects a node of a parent bag in subpartition $P_{\ell,i}$ to a node of a child bag in subpartition $P_{\ell+1,2i}$ or $P_{\ell+1,2i+1}$ would be represented by an edge inside the multibutterfly splitter containing the sets $A_{\ell,i}$, $A_{\ell+1,2i}$, and $A_{\ell+1,2i+1}$. In addition, the AKS edges inside the building blocks, and thus inside the subpartitions, would be represented by edges inside the $A_{\ell,i}$ sets. This means that any two MBF nodes that represent two adjacent AKS nodes are in the same splitter. Therefore, we can restrict ourselves in the following to give a description of the embedding inside the splitters.

4.2 Fine embedding

Consider a splitter consisting of the sets $A := A_{\ell,i}$, $L := A_{\ell+1,2i}$ and $R := A_{\ell+1,2i+1}$. Define $k := |A|$. Assume w.l.o.g. that the nodes in A are labeled $(\ell, 0), (\ell, 1), \ldots, (\ell, k-1)$, the nodes in L are labeled $(\ell+1, 0), \ldots, (\ell+1, k/2-1)$, and the nodes in R are labeled $(\ell+1, k/2), \ldots, (\ell+1, k-1)$. The edges leading from A to L we call *left edges* and the edges leading from A to R we call *right edges*. The edges between A, L, and R are defined by two butterfly networks BF_1 and BF_2 that are folded together to form a multibutterfly such that each node in A is incident to a left and a right BF_1 edge and a left and a right BF_2 edge. We assume that the left BF_1 edge of a node $(\ell, v) \in A$ connects (ℓ, v) to node $(\ell + 1, v \bmod (k/2)) \in L$, and the right BF_1 edge connects it to node $(\ell + 1, k/2 + v \bmod (k/2)) \in R$. (These are the standard butterfly connections.) The edges of BF_2 will be defined later.

We assume inductively that the embedding is done for the levels $\log n$ through $\ell + 1$, and that the folding of BF_1 and BF_2 is specified for these levels. We have to describe the mapping of the AKS nodes in subpartition $P := P_{\ell,i}$ onto the MBF nodes in A, and we have to determine the BF_2 edges between A and $L \cup R$ such that each AKS edge can be mapped to a path of length 1 or 2, with congestion 1. The embedding into the two submultibutterflies below L and R can be assumed to be isomorphic since the two subtrees below a node in the AKS tree are symmetric. Define $L' \subset L$ and $R \subset R$ to be the set of nodes that host nodes from $P_{\ell+1}$ that are adjacent to nodes in P, and define $A' \subset A$ to be the set of nodes above the nodes in L' and R', i.e., $A' := \{(\ell, v) \mid (\ell + 1, v) \in L' \cup R'\}$.

At first, we describe how to map the AKS nodes in P to the nodes in A and how to implement the compare-exchange edges within the AKS building blocks of partition P. At second, we show how to implement the forward edges within the AKS building blocks of P. At third, we describe how to implement the forward edges between building blocks of P and building blocks of $P_L := P_{\ell+1,2i}$ and $P_R := P_{\ell+1,2i+1}$. For the latter two topics, we make use of the freedom to determine the folding of the two butterflies, i.e., we specify the edges of BF_2 in an appropriate fashion. Afterwards we show that the specified BF_2 edges are *admissible*, i.e., consistent with BF_2 being isomorphic with a butterfly network.

1. Each node in P is mapped onto a node in $A \setminus A'$ such that the following condition is met. Suppose u and v are two AKS nodes in P connected by a compare-exchange edge. Let (ℓ, u') and (ℓ, v') denote the MBF nodes that host u and v, respectively. Then we require that $v' - u' = k/2$. In this way, the AKS edge between u and v can be embedded in a path of length two including only left edges of BF_1. The path is
$$(\ell, u') \overset{BF_1}{\rightarrow} (\ell+1, u') \overset{BF_1}{\rightarrow} (\ell, u' + k/2) = (\ell, v') \ .$$

2. Now we implement the forward edges inside the building blocks. Consider an AKS node $u \in P$. Suppose u is a node in row r of the AKS network and u is connected by a forward edge to a node v in row $r+1$. Let u be embedded in node (ℓ, u') and v in node (ℓ, v') of $A \setminus A'$. Then we map the forward edge connecting u and v to a path of length two between (ℓ, u') and (ℓ, v'). This path consists of a right BF_2 edge and a right BF_1 edge. The path is
$$(\ell, u') \overset{BF_2}{\rightarrow} (\ell+1, k/2 + v' \bmod (k/2)) \overset{BF_1}{\rightarrow} (\ell, v') \ .$$

3. Finally, we implement the forward edges between distinct building blocks of level ℓ and level $\ell+1$. Let $u \in P_\ell$ and $v \in P_{\ell+1}$ denote two adjacent AKS nodes of different building blocks. Let $(\ell, u') \in A \setminus A'$ denote the MBF node hosting u, and $(\ell+1, v') \in L' \cup R'$ denote the MBF node hosting v. Then we map the forward edge (u, v) to an BF_2 edge that connects (ℓ, u') and $(\ell+1, v')$.

We have to show that the specified BF_2 edges are admissible. Each of the nodes in A is incident to at most one specified BF_2 edge. A node in R, however, may be incident to two BF_2 edges specified in (2). However, none of these edges is incident to a node in R' because the mapping of the nodes ensures that $(\ell, v') \in A \setminus A'$ and, hence, $(\ell+1, k/2 + v' \bmod (k/2)) \notin R'$. (Recall that the embedding in L and R is assumed to be isomorphic.) In (3), we only have specified edges that are incident to nodes in $L' \cup R'$. Each of these nodes is incident to at most one specified BF_2 edge. Altogether, for each pair of nodes $(\ell+1, v) \in L$ and $(\ell+1, v+k/2) \in R$, we have specified at most two edges that are incident to the nodes of the pair. This ensures that the set of the specified BF_2 edges can be completed in an admissible fashion.

Obviously, the embedding has load 1 and dilation 2. Further, only left BF_1 edges are used for the embedding of the compare-exchange edges in (1) whereas only right BF_1 edges are used for the embedding of the forward edges in (2). Besides the BF_2 edges specified for the embedding in (2) and (3) are disjoint. Therefore, the congestion of the embedding is 1.

In order to implement the embedding we have to assume that the size of A is not too small, or the other way around, that the size of partition P is not too large. In particular, the equation $h \cdot |P| \leq |A \backslash A'|$ must be satisfied since we have to map the $h \cdot |P|$ nodes associated with partition P onto the nodes in $A \backslash A'$. Each column (i.e., a sequence of forward edges) of a building block has two *border nodes*, i.e., nodes that are connected to nodes in the stages above or below P. Each node in A' hosts a border node. Since the number of border nodes is at most $2 \cdot |P|$, we have $|A'| \leq 2 \cdot |P|$. (Note that possibly $|A'| > |P|$.) Further, $|P| = |P_\ell|/2^\ell$, and $|A| \leq m/2^\ell$, with m denoting the number of nodes on a multibutterfly level. Thus, the above embedding can be implemented if the equation

$$(h+2) \cdot |P_\ell| \quad \leq \quad m \tag{2}$$

holds for every tree level ℓ of the AKS network. This yields a constraint on the relationship between the size of the AKS network and the multibutterfly.

4.3 Properties of the AKS network

In order to determine the size of the largest AKS network that can be embedded into a given multibutterfly, we first calculate some properties of the AKS network, including the stage numbers at which the bags on level ℓ first become non-empty, first become full, and then split. We also calculate the total number of stages, and the last level to perform a perfect split.

Define the capacity $C_\ell(t)$ of a level ℓ in the AKS tree to be the sum of the capacities of all bags on this tree level during stage t. Then

$$C_\ell(t) \quad = \quad 2^\ell \cdot c_\ell(t) \overset{(1)}{=} \left(1-(2a)^{-2}\right) \cdot n \cdot \nu^t \cdot (2a)^\ell \ . \tag{3}$$

Note that the cold storage simulates a bag half the size of the root's parent, one quarter the size of the root's grandparent, and so on. Thus, we can imagine the cold storage as partitioned into an infinite number of virtual levels -1, -2, -3, and so on, such that the above equation for $C_\ell(t)$ holds for any integer $-\infty \le \ell \le \log n$, and $t \ge 0$.

In the following, we say two tree levels ℓ and ℓ' are *congruent* if $\ell \cong \ell' \pmod 2$. Analogously, we say a tree-level ℓ and a stage t are *congruent* if $\ell \cong t \pmod 2$. For short we write $\ell \cong \ell'$ or $\ell \cong t$, respectively. In each stage t, each tree level ℓ above the partial level is filled to its capacity if $\ell \cong t$, and is empty if $\ell \not\cong t$. All tree levels below the partial level are empty.

For a stage t, define $\bar{C}_\ell(t)$ to be the sum of the capacities of all tree levels above level ℓ and congruent to t. Then

$$\begin{aligned}
\bar{C}_\ell(t) \quad &= \quad C_{\ell-2}(t) + C_{\ell-4}(t) + C_{\ell-6}(t) + \ldots \\
&= \quad \left(1-(2a)^{-2}\right) \cdot n \cdot \nu^t \cdot (2a)^{(\ell-2)} \cdot \sum_{i=0}^{\infty} (2a)^{-2i} \\
&= \quad n \cdot \nu^t \cdot (2a)^{\ell-2}
\end{aligned} \tag{4}$$

if $\ell \cong t$, and

$$\begin{aligned}
\bar{C}_\ell(t) \quad &= \quad C_{\ell-1}(t) + C_{\ell-3}(t) + C_{\ell-5}(t) + \ldots \\
&= \quad \left(1-(2a)^{-2}\right) \cdot n \cdot \nu^t \cdot \sum_{i=0}^{\infty} (2a)^{(\ell-1)-2i} \\
&= \quad n \cdot \nu^t \cdot (2a)^{\ell-1}
\end{aligned} \tag{5}$$

if $\ell \not\cong t$.

For $\ell \ge 0$, define t_0^ℓ to be the first stage in which tree level ℓ is the partial level, i.e., t_0^ℓ is the first stage in which the size of the bags on tree level ℓ is larger than 0. All of the bags in congruent levels above ℓ are filled up to their capacity in this stage. As a consequence, $\bar{C}_\ell(t_0^\ell) < n$. Further, for every stage t in which ℓ is below the partial level, i.e., for every stage

$t < t_0^\ell$, $\bar{C}_\ell(t) \geq n$ because all items are stored in levels above ℓ. Thus, t_0^ℓ is the smallest integer congruent to ℓ satisfying

$$n \quad > \quad \bar{C}_\ell(t_0^\ell) \overset{(4)}{=} n \cdot \nu^{t_0^\ell} \cdot (2a)^{\ell-2} \ .$$

This gives

$$t_0^\ell \quad = \quad (\ell - 2) \cdot \log_{(1/\nu)}(2a) + x \ , \tag{6}$$

for some $x \in [0, 2]$.

Define t_1^ℓ to be the first congruent stage in which tree level ℓ is filled to its capacity. Then t_1^ℓ is the first stage t in which the number of items stored in bags on tree level ℓ or above this level is $C_\ell(t) + \bar{C}_\ell(t)$. Hence, $C_\ell(t_1^\ell) + \bar{C}_\ell(t_1^\ell) \leq n$. Further, for each stage $t < t_1^\ell$ congruent to ℓ it holds $C_\ell(t) + \bar{C}_\ell(t) > n$. Thus, t_1^ℓ is the smallest integer congruent to ℓ satisfying

$$\begin{aligned} n \quad &\geq \quad C_\ell(t_1^\ell) + \bar{C}_\ell(t_1^\ell) \\ &\overset{(3)+(4)}{=} \left(1 - (2a)^{-2}\right) \cdot n \cdot \nu^{t_1^\ell} \cdot (2a)^\ell + n \cdot \nu^{t_1^\ell} \cdot (2a)^{\ell-2} \\ &= \quad n \cdot \nu^{t_1^\ell} \cdot (2a)^\ell \ . \end{aligned}$$

Therefore,

$$t_1^\ell \quad = \quad \ell \cdot \log_{(1/\nu)}(2a) + x \ , \tag{7}$$

for some $x \in [0, 2]$.

Define t_2^ℓ to be the splitting stage of level ℓ. Then t_2^ℓ is the smallest integer congruent to ℓ satisfying

$$r \quad \geq \quad c_\ell(t_2^\ell) \overset{(1)}{=} \left(1 - (2a)^{-2}\right) \cdot n \cdot \nu^{t_2^\ell} \cdot a^\ell \ .$$

This is because a bag is split in the first congruent stage in which its capacity is not larger than r. Therefore,

$$t_2^\ell \quad = \quad \ell \cdot \log_{(1/\nu)} a + \log_{(1/\nu)} \left(\left(1 - (2a)^{-2}\right) \cdot n/r\right) + x \ , \tag{8}$$

for some $x \in [0,2]$.

The AKS algorithm finishes as soon as the bags on some tree level are split, i.e., are sorted, and all levels below this level are empty. We denote the tree level split in the last stage by ℓ^*. The splitting stage of ℓ^* is $t_2^{\ell^*} = T-1$. Level ℓ^* is the first stage fulfilling $C_{\ell^*}(t_2^{\ell^*})+\bar{C}_{\ell^*}(t_2^{\ell^*}) \geq n$, because the splitting stage of this level is the first splitting stage in which all n items are stored in bags of the split tree level or above this level. Therefore, ℓ^* is the smallest integer satisfying

$$\begin{aligned} n &\leq C_{\ell^*}(t_2^{\ell^*}) + \bar{C}_{\ell^*}(t_2^{\ell^*}) \\ &= n \cdot \nu^{t_2^{\ell^*}} \cdot (2a)^{\ell^*} = \frac{r \cdot \nu^x \cdot 2^{\ell^*}}{1-(2a)^{-2}} , \end{aligned}$$

for some $x \in [0,2]$. This gives

$$\begin{aligned} \ell^* &= \left\lceil \log_2\left(\left(1-(2a)^{-2}\right) \cdot n/r\right) + 2 \cdot \log_2(1/\nu) \right\rceil \\ &= \log_2 n - \Theta(1) \ . \end{aligned} \tag{9}$$

Consequently,

$$\begin{aligned} T &= t_2^{\ell^*} + 1 \\ &\overset{(8)}{=} \ell^* \cdot \log_{(1/\nu)} a + \log_{(1/\nu)} n - \Theta(1) \\ &\overset{(9)}{=} \log_2 n \cdot \log_{(1/\nu)}(2a) - \Theta(1) \ . \end{aligned} \tag{10}$$

4.4 The size of the AKS network

In this section, we calculate an upper bound on the size of the AKS network that can be embedded into a multibutterfly with m nodes on each level according to the above description. That means we are looking for the smallest AKS network that fulfills Equation 2, i.e., $(h+2) \cdot |P_\ell| \leq m$, for every level ℓ of the AKS tree. Thus, we have to bound the size of each partition P_ℓ. We first assume ideal bag sizes and show later that the results for these values are close to the results for the correct integer values.

A special situation occurs for partition P_0 as this partition includes the root bag that is filled up to its capacity from the beginning. The size of the root bag in an even stage t is $C_0(t)$, and in odd stages the size is 0. Besides P_0 includes the cold storage from stage 0 to the stage before the splitting stage of level 1.The size of the cold storage in a stage t is $\bar{C}_0(t)$. Hence, we have

$$
\begin{aligned}
|P_0| \quad & \leq \quad \sum_{t=0}^{\infty} C_0(2t) + \sum_{t=0}^{\infty} \bar{C}_0(t) \\
& \overset{(3)+(4)+(5)}{=} \quad \left(\left(1 - \frac{1}{4a^2}\right) + \frac{1}{4a^2} + \frac{\nu}{2a} \right) \cdot n \cdot \sum_{t=0}^{\infty} \nu^{2t} \\
& = \quad n \cdot \underbrace{\left(1 + \frac{\nu}{2a}\right)\left(\frac{1}{1-\nu^2}\right)}_{=: \, \kappa_1(\nu, a)} \; .
\end{aligned}
$$

Now we bound the size of partition P_ℓ, for $1 \leq \ell \leq \ell^*$. We first ignore the effects of the splitting, i.e., we assume that $r = 0$. Define

$$
t_*^\ell \;\; := \;\; t_0^\ell + \lceil \log_{(1/\nu)}(2a) \rceil \cdot 2 \; \approx \; t_1^\ell \; . \tag{11}
$$

Then the size of P_ℓ can be bound as follows.

- In each stage $t \cong \ell$ with $t_0^\ell \leq t \leq t_*^\ell - 2$, the size of the tree level is at most $n - \bar{C}_\ell(t)$.
- In each stage $t \cong \ell$ with $t \geq t_*^\ell$, the size of the tree level is at most $C_\ell(t)$.
- In all other stages the size is 0.

Note that the first and the second bound, $n - \bar{C}_\ell(t)$ and $C_\ell(t)$, actually hold for all stages. We have just chosen to use one for the first set of time stages and the other for the second one. For $1 \leq \ell \leq \ell^*$, we get

$$
\begin{aligned}
|P_\ell| \quad & \leq \quad \sum_{t=0}^{\frac{t_*^\ell - t_0^\ell}{2} - 1} (n - \bar{C}_\ell(t_0^\ell + 2t)) + \sum_{t=0}^{\infty} C_\ell(t_*^\ell + 2t) \\
& \overset{(3)+(4)}{\leq} \quad n \cdot \frac{t_*^\ell - t_0^\ell}{2} - n \cdot \nu^{t_0^\ell} \cdot (2a)^{\ell-2} \cdot \sum_{t=0}^{\frac{t_*^\ell - t_0^\ell}{2} - 1} \nu^{2t} + n \cdot \nu^{t_*^\ell} \cdot (2a)^\ell \cdot \sum_{t=0}^{\infty} \nu^{2t}
\end{aligned}
$$

$$
\begin{aligned}
&\overset{(11)}{\leq} && n \cdot \frac{t_*^\ell - t_0^\ell}{2} + n \cdot \nu^{t_0^\ell} \cdot (2a)^{\ell-2} \cdot \left(\sum_{t=0}^{\infty} \nu^{2t} - \sum_{t=0}^{\frac{t_*^\ell - t_0^\ell}{2}-1} \nu^{2t} \right) \\
&\overset{(6)+(11)}{\leq} && n \cdot \left(\log_{(1/\nu)}(2a) + 1 \right) + n \cdot \left(\sum_{t=0}^{\infty} \nu^{2t} - \sum_{t=0}^{\log_{(1/\nu)}(2a)-1} \nu^{2t} \right) \\
&= && n \cdot \underbrace{\left(\log_{(1/\nu)}(2a) + 1 + \frac{1}{4a^2 \cdot (1-\nu^2)} \right)}_{=:\, \kappa_2(\nu, a)}
\end{aligned}
$$

under the assumption that $r = 0$. Now we assume $r > 0$. This means that the size of partition P_ℓ is increased by the size of the cold storage in each stage from the splitting stage of level ℓ, which is t_2^ℓ, to the stage before the splitting stage of level $\ell + 1$, which is $t_2^{\ell+1} - 1$. The size of the cold storage in stage t is $\bar{C}_\ell(t)$. Therefore, the above bound on $|P_\ell|$ is increased by an additive amount of

$$
\begin{aligned}
\sum_{t=t_2^\ell}^{t_2^{\ell+1}-1} \bar{C}_\ell(t) &\overset{(4)+(5)}{\leq} \sum_{\substack{t=t_2^\ell \\ t \equiv t_2^\ell}}^{t_2^{\ell+1}-1} n \cdot \nu^t \cdot (2a)^{\ell-2} + \sum_{\substack{t=t_2^\ell \\ t \not\equiv t_2^\ell}}^{t_2^{\ell+1}-1} n \cdot \nu^t \cdot (2a)^{\ell-1} \\
&\leq \left\lceil \frac{t_2^{\ell+1} - t_2^\ell}{2} \right\rceil \cdot n \cdot \nu^{t_2^\ell} \cdot (2a)^{\ell-2} + \left\lfloor \frac{t_2^{\ell+1} - t_2^\ell}{2} \right\rfloor \cdot n \cdot \nu^{t_2^\ell+1} \cdot (2a)^{\ell-1} \\
&\overset{2a\nu \geq 1}{\leq} \left(t_2^{\ell+1} - t_2^\ell \right) \cdot n \cdot \nu^{t_2^\ell} \cdot (2a)^{\ell-2} \cdot \frac{1+2a\nu}{2} \\
&\overset{(8)}{\leq} \left(\log_{(1/\nu)} a + 2 \right) \cdot \frac{r \cdot 2^{\ell-2}}{a^2 \cdot (1-(2a)^{-2})} \cdot \frac{1+2a\nu}{2} \\
&\overset{\ell \leq \ell^*}{\leq} \frac{(\log_{(1/\nu)} a + 2) \cdot r \cdot 2^{\ell^*-1} \cdot (1+2a\nu)}{4a^2 \cdot (1-(2a)^{-2})} \\
&\overset{(9)}{\leq} n \cdot \underbrace{\frac{(\log_{(1/\nu)} a + 2) \cdot (1+2a\nu)}{4a^2\nu^2}}_{=:\, \kappa_3(\nu, a)} .
\end{aligned}
$$

Up to now we have assumed that all bags have ideal sizes as real numbers. But the integer sizes can be larger than the ideal ones. In particular, we choose the integer sizes according to the following rule given by Paterson [42]: if the ideal total size of a subtree of the binary tree is α, then the actual size is $2\lceil \alpha/2 \rceil$. As a consequence, each bag of ideal size b has integer size at most $b + 2$ [42]. Furthermore, the size of the partition representing the highest non-empty tree level including the cold storage is not increased by the integer rounding, because the total number of items is bounded by n and the total size of the subtrees below this level is not decreased. Hence, the upper bound on the size of P_0 is not affected by the rounding, and for P_ℓ with $\ell \geq 1$, we only have to take care for the error due to rounding in those stages in which ℓ is not the highest non-empty tree level. Therefore, we only consider the stages t_0^ℓ to $t_2^\ell - 1$, for $\ell \geq 1$.

We first bound the deviation due to integer rounding in the stages from t_0^ℓ to $t_1^\ell - 1$ and then the deviation in the stages from t_1^ℓ to $t_2^\ell - 1$. The number of congruent stages from stage t_0^ℓ to $t_1^\ell - 1$ is $(t_1^\ell - t_0^\ell)/2 \leq \log_{(1/\nu)}(2a) + 1$, and the number of bags on level ℓ in a stage is at most $2^\ell \leq 2^{\ell^*}$. For each of these bags in each of these stages we have to add at most 2 items to the ideal size in order to get an upper bound for the integer size. Thus, the integer size of a tree level deviates by an additive amount of at most

$$2 \cdot \left(\log_{(1/\nu)}(2a) + 1\right) \cdot 2^{\ell^*} \overset{(9)}{\leq} n \cdot \underbrace{\left(\frac{4 \cdot (\log_{(1/\nu)}(2a) + 1) \cdot (1 - (2a)^{-2})}{r \cdot \nu^2}\right)}_{=: \kappa_4(\nu, a, r)}$$

from its ideal size during the stages from t_0^ℓ to $t_1^\ell - 1$. In the stages from t_1^ℓ to $t_2^\ell - 1$, all bag sizes are not smaller than r. Thus, the relative error in these stages is at most $2/r$.

Putting it all together, for $0 \leq \ell \leq \ell^*$, we have

$$\begin{aligned} |P_\ell| &\leq \max\{n \cdot \kappa_1, (n \cdot \kappa_2 + n \cdot \kappa_3) \cdot (1 + 2/r) + n \cdot \kappa_4)\} \\ &\leq n \cdot \underbrace{\max\{\kappa_1, (\kappa_2 + \kappa_3) \cdot (1 + 2/r) + \kappa_4)\}}_{=: \bar{\kappa}(\nu, a, r)} . \end{aligned}$$

As shown in Section 4.2 an AKS network can be embedded into a multibutterfly network with m nodes per level if Equation 2 holds, i.e., $(h+2) \cdot |P_\ell| \leq m$ is satisfied for every level ℓ. Thus, the embedding is possible if we choose

$$n := \left\lfloor \frac{m}{(h+2) \cdot \bar{\kappa}} \right\rfloor .$$

The size of the multibutterfly is $M = (\log_2 m + 1) \cdot m$, and the size of the AKS network is $N = h \cdot T \cdot n$. Thus, $m = M/(\log_2 m + 1)$ and $n = N/(h \cdot T)$. In addition, $\log_2 m = \log_2 n + \Theta(1)$

and $T \geq \log_2 n \cdot \log_{1/\nu}(2a) - \Theta(1)$. Thus, we have

$$\begin{aligned} N &\geq \frac{M \cdot h \cdot T}{(h+2) \cdot \bar{\kappa} \cdot (\log_2 m + 1)} \\ &\geq \frac{M \cdot h \cdot \left(\log_2 n \cdot \log_{1/\nu}(2a) - \Theta(1)\right)}{(h+2) \cdot \bar{\kappa} \cdot (\log_2 n + \Theta(1))} \\ &= M/\kappa - o(M) \end{aligned}$$

for $\kappa(\nu, a, r, h) := (1 + 2/h) \cdot \bar{\kappa}/\log_{1/\nu}(2a)$, which is at most $1.351\ldots$, for $\nu = 43/48$, $a = 3$, $r = 160$, and $h \geq 36$ as suggested in [42]. This completes the proof of Theorem 4.1.

5 Routing h-relations on multibutterflies

In this section, we give a deterministic algorithm for routing h-relations on a multibutterfly with (α, β)-expansion. Given a d-dimensional multibutterfly, define V_ℓ to be the set of the $n = 2^d$ nodes on level ℓ, for $0 \leq \ell \leq d$. The nodes in V_0 are called *input nodes*, and the nodes in V_d are called *output nodes*. Then an h-relation is a set of pairs of input and outputs nodes $R \subseteq V_0 \times V_d$ such that each node $v_0 \in V_0$ and each node of $v_d \in V_d$ appears in at most h of the pairs in R. Each pair $(v_0, v_d) \in R$ represents a packet that should be routed from an input node v_0 on level 0 to an output node v_d on level d. In one unit of time, each edge of the multibutterfly can transmit one packet in each direction.

We assume each multibutterfly node can store only a constant number of packets, and the multibutterfly has $h-1$ additional levels $-(h-1), \ldots, -1$ which serve as the initial storage for the at most $h \cdot n$ packets. Let V_ℓ denote the set of nodes on level ℓ, for $-(h-1) \leq \ell \leq -1$. Each level ℓ with $-(h-1) \leq \ell \leq -1$ is connected to level $\ell + 1$ by an (α, β)-expander, i.e., for any $X \subseteq V_\ell$ with $|X| \leq \alpha n$ it holds $\Gamma(X) \cap V_{\ell+1} \geq \beta |X|$, e.g., these expanders are copies of the splitter connecting level 0 with level 1. In the following, these expanders are viewed as splitters.

Each node on a level ℓ with $-(h-1) \leq \ell \leq 0$ holds a packet that should be routed to a node on level d. The packets starting in the column of an input node v of level 0 are viewed as the packets of v. (Note that, starting from a configuration in which v initially holds all of its h packets in a local buffer rather than storing these packets in the column above v, the packets can distributed in $h-1$ steps among the nodes of the column.) Each node of level d is the destination of at most h packets.

Upfal [51] presents a deterministic algorithm for routing a permutation, or 1-relation, in $O(\log n)$ steps on an n-input multibutterfly. By adding $h-1$ additional levels, Upfal's algorithm is able to route an h-relation in $O(h+\log n)$ steps provided that the packets are partitioned into appropriate batches of size $\Theta(n)$ such that no more than αm packets from each batch are routed through any splitter of size m. These batches, however, are difficult to compute if the destinations of the packets starting from the same level of the initial storage do not form a permutation. Our algorithm uses Upfal's algorithm as a subroutine, and we show how the batches can be computed efficiently.

Upfal's algorithm. The algorithm routes a set of packets from the input nodes to the output nodes of a multibutterfly with (α,β)-expansion for any constants $\alpha>0$ and $\beta>1$.

The rough routing paths can be explained as follows: a packet stored in a splitter aims to move along an edge of the left concentrator if its destination is in the submultibutterfly below the left half of the splitter, and it aims to move along an edge of the right concentrator if its destination is in the submultibutterfly below the right half of the splitter. Within the splitters of the initial storage, the packets aim to use an arbitrary edge leading to the next level.

Upfal's algorithm requires that the packets are partitioned into L batches $B(0),\ldots,B(L-1)$. The indices of the batches are used as priority keys. A packet in batch $B(i)$ has higher priority than packets in $\bigcup_{j>i} B(j)$. The edges of each splitter are colored with $2k$ colors so that no two edges of the same color are incident to one node. (Recall that the node degree in each splitter is $2k$.) The algorithm works in iterations. In each iteration, each node holds at most one packet. In odd iterations, the edges connecting odd levels to even levels are activated. In even iterations, the edges connecting even levels to odd levels are activated. Edges are activated

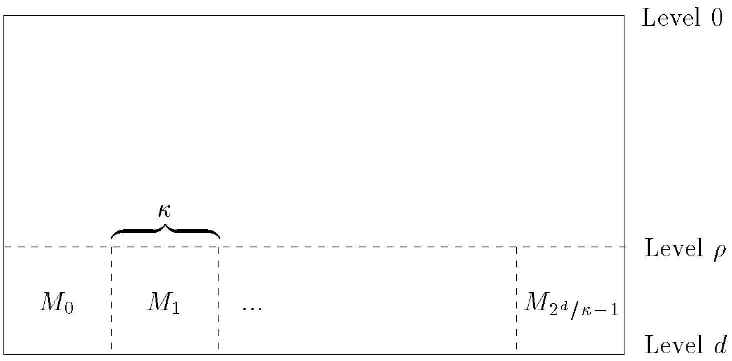

Figure 5: The submultibutterflies of size κ on the levels ρ to d.

one after the other according to the color order. Thus, in each step, only one edge incident to each node is activated. When an edge from node (ℓ, u) to node $(\ell+1, v)$ is activated, if node (ℓ, u) holds in its buffer a packet with higher priority than the packet stored in the buffer of $(\ell+1, v)$, the two nodes exchange packets. (An empty buffer is considered to be a packet with the lowest priority.) We extract the following Lemma from Upfal's analysis [51].

Lemma 5.1 *Suppose the batches are chosen so that no more than αm packets from each batch are routed through any splitter of size m. Then each packet reaches its destination in time $O(\log n + L)$.*

For permutations, it is easy to decompose the packets into $O(1)$ batches that fulfill the above condition. As a consequence, several permutations can be pipelined so that Upfal's algorithm takes time $O(\log n + h)$ for routing h permutations. Note that any h-relation can be decomposed into h disjoint permutations, but it is not clear how to decompose an h-relation into h disjoint permutations on the multibutterfly. Thus, the main problem of routing h-relations is to split the packets into appropriate batches.

The new algorithm. Define κ to be the smallest power of 2 with $\kappa \geq h/\alpha$, and define $\rho := d - \log \kappa$. For $0 \leq i \leq 2^d/\kappa - 1$, define M_i to be the $(\log \kappa)$-dimensional submultibutterfly with node set $\{(\ell, j) \,|\, \rho \leq \ell \leq d, \lfloor j/\kappa \rfloor = i\}$. Each M_i has κ inputs on level ρ and κ outputs on level d. $A_{\rho,i}$ is the input set of M_i. Figure 5 illustrates these definitions. Our algorithm works in three phases.

- *Phase 1:* Partition the packets into $L := 2\kappa$ batches $B(0), \ldots, B(L-1)$ such that $B(i)$ contains the packets with destination nodes in the set $\{(d, v) \mid v \bmod L = i\}$.

  Route the packets with Upfal's algorithm into the "correct" submultibutterfly whose inputs lie on level ρ, i.e., route each packet with destination (d, v) to an arbitrary node in $A_{\rho,\lfloor v/\kappa \rfloor}$.

  For each node (ρ, v) on level ρ, store all arriving packets in the *column* of (ρ, v), i.e., at a node (ℓ, v) with $-(h-1) \leq \ell \leq \rho$, such that each node has to store at most a constant number of packets. (The nodes in the same column are assumed to be connected by a linear array.)

- *Phase 2:* Give each of the packets with the same destination a unique *rank*, i.e., for each submultibutterfly M_i and each output node (d, v) of M_i, number the packets with destination (d, v) from 0 to $h-1$. This is done by κ prefix computations on M_i, one for each output node. (The overall time taken for these computations is $O(\kappa)$ if they are done in a pipelined fashion.)

- *Phase 3:* Partition the packets into $L' := h \cdot \lceil 2/\alpha \rceil$ batches $B(i + j \cdot h) := B(i, j)$ with $0 \leq i \leq h-1$ and $0 \leq j \leq L'-1$. $B(i, j)$ contains each packet p with rank i and a destination in $\{(d, v) \mid v \bmod L' = j\}$.

  Finally, complete the routing with Upfal's protocol according to the new batches.

Intuitively, we have decomposed the h-relation in Phase 2 into h disjoint relations $R_0, \ldots, R_{h-1}$ according to their ranks so that all packets in R_i have distinct destinations.

Theorem 5.2 *The above algorithm routes an arbitrary h-relation in time $O(\log n + h)$.*

Proof: We have to prove that none of the splitters of size m is traversed by more than αm packets in any batch of Phase 1 or Phase 3. In Phase 1, the number of packets from a batch

$B(j)$ passing through the ith splitter of size m on level ℓ is at most

$$h \cdot \left|\left\{v \;\middle|\; v \bmod L = j,\ \left\lfloor \frac{v}{m} \right\rfloor = i\right\}\right| \leq \frac{hm}{L} + h \leq \frac{\alpha m}{2} + h\,.$$

Since the packets route only through the levels 0 to $\rho - 1 = d - \log\kappa - 1$, we have to consider only splitters of size $m \geq 2\kappa$. Hence, $h \leq \alpha\kappa \leq \alpha m/2$, and thus the number of packets passing through a splitter of size m is at most $\alpha m/2 + h \leq \alpha m$. As a consequence, Phase 1 can be done in time $O(\log n + L) = O(\log n + h)$.

Now we consider the number of packets from a batch passing through a splitter in Phase 3. After Phase 2, we have assigned ranks to the packets so that there is at most one packet with each rank bound for each destination. The number of packets of each of these batches passing through a splitter of size m is at most

$$\frac{m}{\lceil \frac{2}{\alpha} \rceil} + 1 \leq \alpha m\ ,$$

for $m \geq 1/\alpha$. (Splitters of size $m < 1/\alpha$ are assumed to be completely connected bipartite graphs.) Hence, Phase 3 takes time $O(\log n + L') = O(\log n + h)$.

Note that the bound on the routing time for Phase 1 also guarantees that all packets received by a node on level ρ can be stored in the respective column such that each node has to store a constant number of packets. This is because each column consists of $\log n + h$ nodes, and each node on level ρ can receive at most one packet per time step.

Finally, we have to show how the ranks in Phase 2 can be computed efficiently. Note that each packet crosses one of the nodes in level ρ in Phase 1. Thus, these nodes can count the number of packets destined for each of the κ destinations in the respective submultibutterfly, which can be done with constant memory size at each node by distributing the κ values among the at least $\kappa/2$ nodes in the column of the counting node. In Phase 2, the unique ranks of the packets directed to output node (d, u) are calculated by a prefix computation in the submultibutterfly including (d, u). The κ prefix computations are done in a pipelined fashion in each of the submultibutterflies. This takes time $\kappa + \log\kappa = O(h)$. Thus, the ranks can be computed and distributed among the at most $O(h + \log n)$ packets stored in a column in time $O(h + \log n)$. $\square$

A more practical solution. If $h = O(\log n)$, another practical solution is to replace the κ-input submultibutterflies with $\kappa \times \kappa$ *meshes of trees*.

A $\kappa \times \kappa$ mesh of trees consists of an array of nodes with κ rows and κ columns. The nodes in each row serve as the leaves of a complete binary tree called a *row tree*, and the nodes in each column serve as the leaves in a *column tree*. Hence, node (i, j) in the array serves as both the ith leaf in the jth column tree, and the jth leaf in the ith row tree. An h-relation can be routed between the roots of the column trees and the roots of the row trees in $O(h + \log \kappa)$ steps by simply routing each packet down its column tree to the appropriate row, and then up through the row tree to its root.

In our application, the roots of the column trees in a mesh of trees replace the inputs of a κ-input submultibutterfly, and the roots of the row trees replace its outputs. A $\kappa \times \kappa$ mesh of trees has $3\kappa^2 - 2\kappa = \Theta(\kappa^2)$ nodes. Since there are n/κ meshes of trees, they contain a total of $\Theta(n \cdot \kappa)$ nodes. For $h = O(\log n)$ (and hence $\kappa = O(\log n)$), this total is $O(n \log n)$, the same as the number of nodes in an n-input multibutterfly. Thus, replacing the submultibutterflies by the meshes of trees does not increase the asymptotic number of nodes. Also, the VLSI layout area of a $\kappa \times \kappa$ mesh of trees is $\Theta(\kappa^2 \log^2 \kappa)$. Since there are n/κ of them, their total VLSI layout area is $\Theta(n \cdot \kappa \log^2 \kappa)$. Since the layout area of the multibutterfly is $\Theta(n^2)$, replacing the submultibutterflies with the meshes of trees does not increase the asymptotic VLSI layout area.

Networks besides the mesh of trees could be plugged in as well, e.g., a $\kappa \times \kappa$ mesh would work, but its routing algorithm would be slightly more complicated.

6 Simulating expansion on a twinbutterfly

The concentrators of a twinbutterfly have poor expansion. This can be shown as follows. Consider a concentrator $G = (A \cup B, E)$ of a twinbutterfly with input set A of size m and output set B of size $m/2$. G can be constructed from a bipartite $m \times m$ graph $G' = (A \cup B', E')$ of degree 2 by merging together two nodes from B' to form each node in B. In G', for any $i \leq |A|$, there exists a subset $X \subseteq A$ where $|X| = i$ such that $|\Gamma(X)| \leq i+1$, because a degree-2 graph consists only of node disjoint cycles. The same bound on the expansion holds for the concentrator G because merging nodes in B' can only reduce the expansion. For example, Upfal's algorithm requires (α, β)-expansion for some constant $\beta > 1$, which is not present if there are sets of size i with only $i+1$ neighbors for all i. However, the following theorem shows that the effective expansion of a twinbutterfly can be improved by embedding multibutterflies of higher degree.

Theorem 6.1 *For any α and β with $\alpha\beta < 1/4$, there exists a twinbutterfly TBF in which an equal-sized multibutterfly MBF having (α, β)-expansion can be embedded with constant congestion and dilation.*

Proof: We describe a d-dimensional twinbutterfly TBF and an equal-sized multibutterfly MBF of degree $4k$, for constant k, such that MBF can be embedded into TBF with constant load, constant dilation, and constant congestion. TBF and MBF will be constructed randomly, and we will prove that the probability that MBF has (α, β)-expansion is larger than 0.

Consider the first k levels of the twinbutterfly TBF. We define these levels by describing the underlying butterfly networks BF_1 and BF_2, i.e., the two butterflies from which TBF is constructed. We assume that BF_1 has the "usual" butterfly node labels, i.e., the edges of BF_1 connect a node $(\ell, v_0 \ldots v_{d-1})$ on level ℓ to the nodes $(\ell+1, v_0 \ldots v_\ell \ldots v_{d-1})$ and $(\ell+1, v_0 \ldots \bar{v_\ell} \ldots v_{d-1})$ on level $\ell+1$.

BF_2 is defined randomly. For any $1 \leq \ell \leq k$ and $w \in \{0,1\}^k$, suppose $\phi_{\ell,w}$ is a permutation chosen randomly and uniformly from the set of permutations on $\{0,1\}^{d-k}$. Then each node

$(\ell-1, v)$ with $v = v_0 \dots v_{d-1} \in \{0,1\}^d$ is connected by a BF_2-edge to node

$$(\ell, v_0 \dots v_{k-1}\, \phi_{\ell, v_0 \dots v_{k-1}}(v_k \dots v_{d-1})) \ ,$$

for $1 \le \ell \le k$. (The second BF_2-edge on the same level can be chosen arbitrarily.) Intuitively, traversing one of these edges randomly modifies the last $d-k$ bits of the node labels.

Next we define the first k levels of the degree-$4k$ multibutterfly MBF. Consider level ℓ of MBF with $0 \le \ell \le k-1$. Suppose $\pi_{i,w}$ is a permutation chosen randomly and uniformly from the set of permutations on $\{0,1\}^{k-\ell}$, for $1 \le i \le k$ and $w \in \{0,1\}^{d-(k-\ell)}$. Let (ℓ, v) be a node on level ℓ with $v = v_0 \dots v_{d-1} \in \{0,1\}^d$. Define $x := v_0 \dots v_{\ell-1}$, $y := v_{\ell+1} \dots v_k$, and $z := v_{k+1} \dots v_{d-1}$. Further, define $y'_i := \pi_{i,x0z}(y)$, and $z'_i := \phi^{-1}_{i,x0y'}(z)$, for $1 \le i \le k$, where ϕ^{-1} denotes the inverse of ϕ. Intuitively, the π-permutations randomly switch the y-bits, and the ϕ^{-1}-permutations randomly switch the z-bits. We connect $(\ell, v) = (\ell, x\,\{0,1\}\,y\,z)$ with $2k$ nodes on level $\ell+1$, i.e., with the nodes

$$(\ell+1, x\,0\,y'_i\,z'_i) \quad \text{and} \quad (\ell+1, x\,1\,y'_i\,z'_i) \ ,$$

for $1 \le i \le k$. Note that all edges are inside the splitters and that each node on level $\ell+1$ is the endpoint of $2k$ edges.

The embedding has constant congestion and dilation because there is a path in TBF of length at most $2k+1$ from (ℓ, v) to any node adjacent in MBF on level $\ell+1$. For $1 \le i \le k$ and $b, b' \in \{0,1\}$, this path can be constructed as follows

$$\begin{array}{rcl}
(\ell, v) & = & (\ell, x\,b\,y\,z) \\
 & \rightarrow \overset{BF_1}{\dots} \rightarrow & (k, x\,b'\,y'_i\,z) \\
 & \rightarrow \overset{BF_1}{\dots} \rightarrow & (i, x\,b'\,y'_i\,z) \\
 & \overset{BF_2}{\rightarrow} & (i-1, x\,b'\,y'_i\,z'_i) \\
 & \rightarrow \overset{BF_1}{\dots} \rightarrow & (\ell+1, x\,b'\,y'_i\,z'_i) \ ,
\end{array}$$

where $\rightarrow \overset{BF_1}{\dots} \rightarrow$ denotes a path using BF_1 edges, and $\overset{BF_2}{\rightarrow}$ denotes a single $BF2$ edge.

We now investigate the expansion of MBF. Consider one of the concentrators in the first k levels. It consists of a node set $A = A_{\ell,i}$ and a node set $B = A_{\ell+1,2i}$ or $B = A_{\ell+1,2i+1}$ with $0 \leq \ell \leq k-1$ and $0 \leq i \leq 2^{\ell}-1$. Define $m := |A|$. Suppose that π_i and ϕ_i, for $1 \leq i \leq k$, are random functions such that each node in A is connected with k nodes chosen independently and randomly from B. Then the probability that all edges that are incident to nodes in a subset $X \subseteq A$ have their endpoints in a subset $Y \subseteq B$ is

$$\left(\frac{|Y|}{|B|}\right)^{k\cdot|X|} \quad = \quad \left(\frac{2|Y|}{m}\right)^{k\cdot|X|} \quad .$$

Actually, π_i and ϕ_i, for $1 \leq i \leq k$, are independent random permutations instead of random functions. However, this does not increase the above probability. As a consequence, the probability that the concentrator has no (α,β)-expansion is at most

$$\begin{aligned} & \sum_{\mu=1}^{\lfloor\alpha\cdot m\rfloor} \sum_{\substack{X\subseteq A\\|X|=\mu}} \sum_{\substack{Y\subseteq B\\|Y|=\lfloor\beta\cdot\mu\rfloor}} \left(\frac{2\cdot\beta\cdot\mu}{m}\right)^{k\cdot\mu} \\ \leq \quad & \sum_{\mu=1}^{\lfloor\alpha\cdot m\rfloor} \binom{m}{\mu}\cdot\binom{m/2}{\lfloor\beta\cdot\mu\rfloor}\cdot\left(\frac{2\cdot\beta\cdot\mu}{m}\right)^{k\cdot\mu} \\ \leq \quad & \sum_{\mu=1}^{\lfloor\alpha\cdot m\rfloor} \left(\alpha^{k-1-\beta}\cdot e^{1+\beta}\cdot(2\beta)^{k-\beta}\right)^{\mu} \quad . \end{aligned}$$

We choose $k > (\beta\cdot\log(\mathrm{e}/(2\alpha\beta))+\log(4\mathrm{e}/\alpha))/\log(1/(4\alpha\beta))$. Then the above term bounding the probability of a bad event in one concentrator is smaller than $2^{-(k+1)}$. Thus, the probability that all 2^{k+1} concentrators of the first k levels have (α,β)-expansion is greater than 0. Consequently, we can choose the edges of the twinbutterfly TBF so that the first k levels of a multibutterfly with (α,β)-expansion can be embedded with constant congestion and dilation. The levels k to $d-1$ of TBF can be viewed as 2^k independent twinbutterflies of dimension $d-k$. Applying the above scheme recursively to these butterflies completes our proof. □

7 Open problems

We conclude with a few open problems.

1. Can an N-node multibutterfly whose splitters have the (α, β)-expansion property be embedded with constant load, congestion, and dilation, in an $O(N)$-node AKS network whose building blocks have (α, β) (or better) expansion?

2. What is the complexity of selecting the kth largest item from among M items on an N-node bounded-degree network for $\omega(1) \leq M/N \leq o(\log N \log\log(M/N))$?

8 Acknowledgements

The authors would like to thank Friedhelm Meyer auf der Heide and Christian Scheideler for many helpful discussions.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3:1–19, 1983.

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

[3] M. J. Atallah. On multidimensional arrays of processors. *IEEE Transactions on Computers*, 37(10):1306–1309, Oct. 1988.

[4] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

[5] K. Batcher. Sorting networks and their applications. In *Proceedings of the AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314, 1968.

[6] O. Bonorden, B. Juurlink, I. von Otte, and I. Rieping. The Paderborn University BSP (PUB) library – design, implementation and performance. In *Proceedings of 13th Inter-*

national Parallel Processing Symposium & 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP), San Juan, Puerto Rico, April, 1999.

[7] P. Berthomé, A. Ferreira, B. M. Maggs, S. Perennes, and C. G. Plaxton. Sorting-based selection algorithms for hypercubic networks. In *Proceedings of the 7th International Parallel Processing Symposium*, pages 89–95, April 1993.

[8] S. N. Bhatt, F. R. K. Chung, J.-W. Hong, F. T. Leighton, B. Obrenić, A. L. Rosenberg, and E. J. Schwabe. Optimal emulations by butterfly-like networks. *Journal of the ACM*, 43(2):293–330, Mar. 1996.

[9] G. Bilardi and F. P. Preparata. A minimum VLSI network for $O(\log N)$ time sorting. *IEEE Transactions on Computers*, C–34(4):336–343, April 1985.

[10] G. Bilardi and F. P. Preparata. The VLSI optimality of the AKS sorting network. *Information Processing Letters*, 20(2):55–59, February 1985.

[11] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[12] E. A. Brewer, F. T. Chong, and F. T. Leighton. Scalable expanders: Exploiting hierarchical random wiring. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 144–152, May 1994.

[13] F. Chong, E. Egozy, and A. DeHon. Fault tolerance and performance of multipath multistage interconnection networks. In T. F. Knight, Jr. and J. Savage, editors, *Advanced Research in VLSI: Proceedings of the MIT/Brown Conference 1992*. MIT Press, March 1992. To appear.

[14] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[15] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Journal of Computer and System Sciences*, 47(3):501–548, December 1993.

[16] A. DeHon, T. F. Knight Jr., and H. Minsky. Fault-tolerant design for multistage routing networks. In *Proceedings of the International Symposium on Shared Memory Multiprocessing*, pages 60–71. Information Processing Society of Japan, April 1991.

[17] A. DeHon, T. F. Knight Jr., and H. Minsky. Fault-tolerant design for multistage routing networks. In Norihisa Suzuki, editor, *Shared Memory Multiprocessing*, chapter 20, pages 483–503. MIT Press, Cambridge, MA, 1992.

[18] R. Feldmann and W. Unger. The cube-connected cycles network is a subgraph of the butterfly network. *Parallel Processing Letters*, 2(1):13–19, 1992.

[19] M. R. Fellows. *Encoding Graphs in Graphs.* PhD thesis, Department of Computer Science, University of California, San Diego, CA, 1985.

[20] A. V. Goldberg, B. M. Maggs, and S. A. Plotkin. A parallel algorithm for reconfiguring a multibutterfly network with faulty switches. *IEEE Transactions on Computers*, 43(3):321–326, March 1994.

[21] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the BSP model. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 1–12, June 1996.

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

[23] N. Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, September 1995.

[24] R. R. Koch, F. T. Leighton, B. M. Maggs, S. B. Rao, A. L. Rosenberg, and E. J. Schwabe. Work-preserving emulations of fixed-connection networks. *Journal of the ACM*, 44(1):104–147, Jan. 1997.

[25] D. E. Knuth. *The Art of Computer Programming*, volume 3, Sorting and Searching. Addison–Wesley, Reading, MA, second edition, 1973.

[26] C. P. Kruskal and K. J. Rappoport. Bandwidth-based lower bounds on slowdown for efficient emulations of fixed-connection networks. In *Proceedings of the 6th ACM Symposium on Parallel Algorithms and Architectures*, pages 132–139, June 1994.

[27] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C–34(4):344–354, April 1985.

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

[29] F. T. Leighton and B. M. Maggs. Fast algorithms for routing around faults in multibutterflies and randomly-wired splitter networks. *IEEE Transactions on Computers*, 41(5):578–587, May 1992.

[30] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, July 1994.

[31] T. Leighton, D. Lisinski, and B. Maggs. Empirical evaluation of randomly-wired multistage networks. In *Proceedings of the 1990 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 380–385. IEEE Computer Society Press, September 1990.

[32] Y. Ma. An $O(n \log n)$-size fault-tolerant sorting network. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, pages 266–275, May 1996.

[33] B. M. Maggs and E. J. Schwabe. Real-time emulations of bounded-degree networks. *Information Processing Letters*, to appear.

[34] G. A. Margulis. Explicit constructions of concentrators. *Problemy Peredachi Informatsii*, 9:325–332, 1973. In Russian.

[35] G. A. Margulis. Explicit constructions of concentrators. *Problems of Information Transmission*, 9:71–80, 1975.

[36] W. F. McColl. BSP programming. In G. E. Blelloch, K. M. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms.* Volume 18 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 25–35. American Mathematical Society, Providence, RI, May 1994.

[37] F. Meyer auf der Heide. Efficiency of universal parallel computers. *Acta Informatica*, 19:269–296, 1983.

[38] F. Meyer auf der Heide. Efficient simulations among several models of parallel computers. *SIAM Journal on Computing*, 15(1):106–119, Feb. 1986.

[39] F. Meyer auf der Heide and R. Wanka. Time-optimal simulations of networks by universal parallel computers. In *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science.* Volume 349 of *Lecture Notes in Computer Science*, pages 120–131. Springer–Verlag, Heidelberg, Germany, Feb. 1989.

[40] F. Meyer auf der Heide, M. Storch, and R. Wanka. Optimal trade-offs between size and slowdown for universal parallel networks. *Theory of Computing Systems*, 30:627–644, 1997.

[41] R. Miller and J. L. Reed. The Oxford BSP library: User's guide. Version 1.0. Oxford parallel technical report, Oxford University Computing Laboratory, Oxford, England, 1994.

[42] M. S. Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5:75–92, 1990.

[43] A. Pietracaprina. Deterministic routing of h-relations on the multibutterfly. In *Proc. of the First Merged IPPS/SPDP Symposium: 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing*, pages 375–379, Orlando, Florida, March-April 1998.

[44] M. Pinsker. On the complexity of a concentrator. In *7th International Teletraffic Congress*, pages 318/1–318/4, June 1973.

[45] N. Pippenger. Self-routing superconcentrators. *Journal of Computer and System Sciences*, 52(1):53–60, February 1996.

[46] C. G. Plaxton. On the network complexity of selection. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 396–401, October 1989.

[47] C. G. Plaxton. Tight bounds for a distributed selection game with applications to fixed-connection machines. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 114–122, October 1995.

[48] K. J. Rappoport. On the slowdown of efficient simulations of multibutterflies. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 176–182, June 1996.

[49] J. H. Reif and L. G. Valiant. A logarithmic time sort for linear size networks. *Journal of the ACM*, 34(1):60–76, January 1987.

[50] E. J. Schwabe. Constant-slowdown simulations of normal hypercube algorithms on the butterfly network. *Information Processing Letters*, 45(2):295–301, Apr. 1993.

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

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

[53] U. Vishkin. An optimal parallel algorithm for selection. In *Parallel and Distributed Computing.* Volume 4 of *Advances in Computing Research*, pages 79–86. JAI Press, Greenwich, CT, 1987.