Improved Routing and Sorting on Multibutterflies

Bruce M. Maggs   Berthold Vöcking*

November 1996

CMU-CS-96-192

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

* Dept. of Mathematics and Computer Science, and
Heinz Nixdorf Institute
University of Paderborn
33095 Paderborn, Germany

Bruce Maggs is supported in part by the Air Force Materiel Command (AFMC) and ARPA under Contract F196828-93-C-0193, by ARPA Contracts F33615-93-1-1330 and N00014-95-1-1246, and by an NSF National Young Investigator Award, No. CCR-94-57766, with matching funds provided by NEC Research Institute. This research was conducted in part while he was visiting the Heinz Nixdorf Institute, with support provided by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen". Berthold Vöcking is supported in part by DFG-Sonderforschungsbereich 376, and by DFG Leibniz Grant Me872/6-1.

Keywords: Multibutterfly networks, AKS sorting network, routing algorithms, sorting algorithms

Abstract

This paper shows that an N-node AKS network (as described by Paterson) can be embedded in a $\frac{3N}{2}$-node degree-8 multibutterfly network 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 $n \log n$ keys on an n-input multibutterfly in $O(\log n)$ time, a work-efficient algorithm for finding the median of $n \log^2 n \log\log n$ keys 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 2-folded butterfly, whose individual splitters do not exhibit expansion, can emulate a bounded-degree multibutterfly with an (α, β) expansion property, for any $\alpha \cdot \beta < 1/4$.

1 Introduction

In 1983, Ajtai, Komlós, and Szemerédi devised a network for sorting n keys 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 [4]. Indeed, the difficulty of improving on Batcher's construction led Knuth to conjecture that there was no sorting network with depth $O(\log n)$ [18, p. 243].

The AKS sorting network differed from previous constructions in one crucial aspect: it incorporated *expansion* into its structure. Expansion is a graph-theoretic notion. An $l \times r$ bipartite graph is said to be an (α, β)-*expander* if every set of k nodes on the left side has at least βk neighbors on the right side, provided that $k \leq \alpha l$, where α and β are constants, $\alpha < 1$, and $\beta > 1$. As it happens, a random graph is likely to be an expander [29]. There are also explicit constructions of expanders. These constructions were first discovered by Margulis [24, 25], 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 [17].

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 $(\log^2 n)/2 + (\log n)/2$ [11, p. 650]. Some progress has been made in simplifying the AKS network and in improving the constant factors in its depth [28], 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 [19], and Ma's fault-tolerant sorting network [23]. Leighton shows how to construct an N-node degree-3 network capable of sorting N keys 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 shows 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 [3], 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; they probably did not consider it important.

The term "multibutterfly" was introduced by Upfal [35]. 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 [20, 22]. Hence, a case can be made for the practicality of multibutterflies, and several studies have explored their implementation [9, 10, 13, 14].

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 [20]. 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. 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 [15] and [30], 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 degree-8 multibutterfly with load 1, congestion 1, and dilation 2. As a consequence an N-node multibutterfly can emulate an N-node AKS network with constant slowdown.

The embedding has several immediate implications. The emulation of the AKS network by the multibutterfly, along with Leighton's columnsort algorithm [19], yields the first deterministic $O(\log N)$-step algorithm for sorting N elements on an N-node multibutterfly. The sorting algorithm can then be used to construct the first deterministic $O(\log N)$-step algorithms for finding the median of N elements and for routing with combining on multibutterflies. It also yields a work-efficient deterministic algorithm for finding the median of $N \log N \log\log N$ elements in $O(\log N \log\log N)$ time on an N-node multibutterfly. Because the embedding of the AKS network into the multibutterfly has constant 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 [6, 7], but the three-dimensional layout is new.

We also present two deterministic algorithms for solving h-to-one routing problems on an n-input butterfly in $O(h+\log n)$ time. One applies when h is known, and the other when it is not. Previous routing algorithms could solve h one-to-one problems in a pipelined fashion [20, 35], 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 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 [36], for which there are growing libraries of parallel programs [16, 27, 26].

Finally, we show that a 2-folded butterfly (i.e., a degree-8 multibutterfly), 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 keys on an N-node multibutterfly was the Sharesort algorithm of Cypher and Plaxton [12]. This algorithm was designed to run on the butterfly network, or on any other hypercubic network. 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 [21, 34].

Prior to this work, the fastest deterministic selection algorithm for multibutterflies was the algorithm of Berthomé, Ferreira, Maggs, Perennes, and Plaxton [5]. This algorithm selects the kth largest element from among N elements 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 [8], this algorithm, which performs $N \log N \log^* N$ work, is not work efficient. Furthermore, Plaxton [31] 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 elements. 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. 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 [32]. 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 elements on an N-node network, the AKS sorting network combined with columnsort can be used to sort the elements (and hence solve the selection problem) in $O(\log N)$ time [19]. This algorithm is optimal because selection on any bounded-degree N-node network requires $\Omega(\log N)$ time. The kth largest of M elements, $M \geq N$, 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 [37] that invokes the AKS sorting network and columnsort as subroutines [31]. 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 [33] has recently proved an even larger separation, namely that the largest butterfly that can efficiently simulate a 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.

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 multibutterfly 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 2-folded butterfly can simulate a multibutterfly with an (α, β) expansion property. 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. Let (ℓ, j) denote the jth node on level ℓ. For each level $0 \leq \ell \leq d$, the nodes on level ℓ are partitioned into 2^ℓ sets $A_{\ell,0}, \ldots, A_{\ell,2^\ell-1}$.

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

The nodes in $A_{\ell,i}$ are connected to some nodes in the sets $A_{\ell+1,2i}$ 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 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 $A_{\ell+1,2i+1}$. The nodes on level 0 of a d-dimensional multibutterfly are called *input nodes*, and the nodes on level d are called *output nodes*.

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 $B = A_{\ell+1,2i+1}$ or $B = A_{\ell+1,2i+1}$, for $0 \leq \ell \leq d-1$. The edges in a concentrator can be chosen in an arbitrarily 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, i.e., the butterfly network is a special variant 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 $|\Gamma(X) \cap B| \geq \beta|X|$. A multibutterfly is said to have (α, β)-expansion if all its concentrators have (α, β)-expansion. Upfal [35] shows that for any d, k, α, and β with $2\beta < k - 1$, and $\alpha > 1/(2\beta)(2\beta e^{1+2\beta})^{-1/(d-2\beta-1)}$, there exists a multibutterfly of degree $4k$ with (α, β)-expansion.

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

3 The AKS network

Our description of the AKS network is based on Paterson's description [28]. Ours is a little more general than Paterson's because we do not describe the building blocks, i.e., the separators, and the sorters in detail.

The AKS network is a sorting network that consists of $h \cdot T$ levels that are partitioned into T stages of width n and of constant height h. Let

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

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

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

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 of width m returns a partition of its m input elements into four parts, FL (far-left), CL (center-left), CR (center-right), FR (far-right). We do not describe the structure of the separators but we are interested in the sizes of the four partitions. The size of FL and FR is $\lambda \cdot m$ and the size of CL and CR is $(1 - \lambda)$, e.g, $\lambda = 1/8$. The sorters return the m input elements in sorted order. It is convenient to implement the sorters as Batcher's bitonic sorting network [4]. 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 $t-1$ are connected to which input partitions of the blocks in stage t, for $1 \leq t \leq T-1$. Our description is based on an oblivious sorting algorithm structured about a complete binary tree 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 works in T stages that are equivalent to the stages of the AKS network.

Consider a binary tree B with "bags" at each node. Initially, the set of n elements to be sorted is contained in the single bag at the root. Suppose each node of the tree partitions the elements that it gets from its parent into two halves and sends the smaller half to the left child and the larger half to the right child. Then the elements will arrive in sorted order at the leaves of the tree. Unfortunately, it is not possible to split the elements exactly into the two halves at each node in constant time. The strategy of the AKS algorithm is to make an approximate partition of elements, which can be done by the separators. The elements that are sent to the wrong child are then retransmitted in a later stages.

We will not describe the algorithm in detail. Instead, we consider the flow of the elements between the bags. The proof that the algorithm sorts can be found in Paterson's article [28]. Associated with each node of the tree is a *bag* that contains a number of elements. The *size of a bag* is the number of elements stored in the bag, and the *capacity of a bag* is the maximum number of elements that can be stored in that bag. During most stages, a bag is either empty or filled to its capacity. The capacity of each bag at level ℓ is $s \cdot A^{\ell}$ for some constant A, e.g. $A = 3$, and some value of s that decreasing with time.

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. Each stage T is implemented in stage T of the AKS network. At odd stages (some) odd levels are full and all the bags at the even levels are empty. The opposite holds at even stages. At each stage the elements 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.

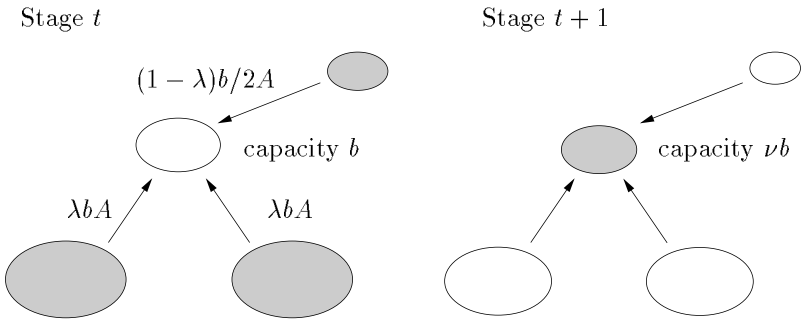

Figure 1: Reduction of bag capacities after each stage.

Consider a bag with capacity b that is empty at the beginning of some stage and which is filled to its new capacity νb at the end of the stage, as shown in Figure 1. Then

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

which gives

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

We assume that $\nu < 1$, e.g. $\nu = 43/48$. Thus, the capacities diminish at each stage and keys 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

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

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 $(1 - 1/(4A^2)) \cdot n$ keys. Since we would like the root to behave as if it were an ordinary node, we place above it a subset of the elements of size $1/(4A^2) \cdot n$. This subset we call the *cold storage*. The root exchanges keys with the cold storage as with a parent. The cold storage simulates half the root's parent, one-fourth the root's grandparent, and so on. The capacity of the cold storage in a step t is therefore

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

if t is even, and

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

if t is odd. The cold storage is the simplest structure in the AKS-network. It is implemented as a forward block.

During the course of the algorithm the elements 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 elements if it has sufficiently many. After this requirement is met, any remaining elements 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. $r = 160$, then the set of elements 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. After the first splitting step, a new splitting step will be required at regular bounded intervals, i.e., whenever the capacity of a bag becomes smaller than r, the separator is replaced by a sorter and the elements are split into two halves. The algorithm finishes after stage $T - 1$, in which the elements of the bags on some level are sorted and all bags below this level are empty.

The widths of the building blocks in the AKS network can be extracted from the above description. All sizes are specified as real numbers. 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. We denote the width of the AKS network by n, the number of stages by T, and the height of the stages by h. We assume that the widths of the building blocks are defined by the parameters λ, A, ν, and r as described in Section 3. We prove the following result.

Theorem 4.1 *An AKS network of size N can be embedded into a 2-folded butterfly of size $M \leq \kappa \cdot N + o(N)$ with 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 $h = 36$. Then κ is smaller than 1.5. In the following, we describe the embedding and prove the result on the relationship of the network sizes.

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, we can imagine that we have T trees $B_0, \ldots, B_{T-1}$ of fixed sized batches, such that the batches in the tth tree represent the building blocks of the tth multibutterfly stage. In particular, 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, we divide the blocks of the AKS network according to the tree-levels into partitions $P_0, \ldots, P_{\log n}$. That means that partition P_ℓ includes all $2^\ell \cdot T$ building blocks that are associated to a node on the ℓth tree-level in one of the T trees. In addition, we add the forward blocks of the cold storage 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, e.g., all bags below the partial level.

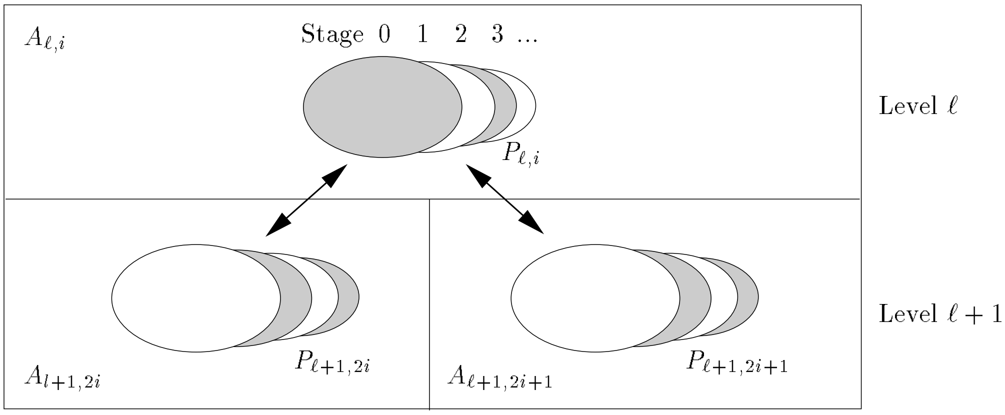

Figure 2: Rough embedding of the AKS network into the multibutterfly.

The MBF 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. Divide each partition P_ℓ into equal-sized subpartitions $P_{\ell,0},\dots,P_{\ell,2^\ell-1}$ 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 $P_{\ell+1,2i+1}$. Embed the AKS nodes of partition $P_{\ell,i}$ into the MBF-nodes of the set $A_{\ell,i}$. Of course, 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.

Now suppose we add all AKS edges to the MBF regardless of the multibutterfly structure, i.e., we 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 $P_{\ell+1,2i+1}$ is 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, are represented by edges inside the $A_{\ell,i}$ sets. Thus, we can restrict ourselves to give a description of the embedding inside the splitters.

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 $m:=|A|$. The edges between A, L, and R are defined by two butterfly networks BF_1 and BF_2 that are folded together to a multibutterfly. We assume that the embedding is done for the levels $\log n$ to to $\ell+1$. That means that the folding of BF_1 an BF_2 are fixed up to level $\ell+1$. We have to describe the mapping of the AKS nodes in subpartition $P:=P_{\ell,i}$ onto the MBF nodes in A.

First we embed the AKS nodes so that each compare-exchange edge of the AKS network can be simulated by two edges of BF_1. Suppose the nodes in A are labeled $(\ell,0),(\ell,1),\dots,(\ell,m-1)$, the nodes in L are labeled $(\ell+1,0),\dots,(\ell,m/2-1)$, and the nodes in R are labeled $(\ell+1,m/2),\dots,(\ell,m-1)$, so that each node $(\ell,v)\in A$ is connected by an BF_1 edge to node $(\ell+1,v)$ and $(\ell+1,v+m/2 \pmod m)$ from $L\cup R$. Then each node in A is connected by a *left edge* to a node in L and by a *right edge* to a node in R.

We embed each pair of AKS nodes u and v of P that are connected by a compare-exchange edge to two nodes (ℓ,u') and (ℓ,v') of A, respectively, so that $v'-u'=m/2$. Then the AKS edge between u and v can be simulated by a path of length 2. The path is

$$(\ell,u')\overset{BF_1}{\to}(\ell+1,u')\overset{BF_1}{\to}(\ell,v')\ .$$

Note that the path uses only left edges of BF_1.

Now we embed the forward edges inside the building blocks. Until now we have not used the freedom to determine the folding of the two butterflies, i.e., we have not fixed the edges in BF_2. Suppose we connect each node of the set R to two nodes in A such that each node in A is adjacent to one node in R. Then these edges are admissible choices for the edges in BF_2. We use this fact to realize the forward edges. Consider an AKS node $u \in P$. Suppose u is a node in row t of the AKS and u is connected by a forward edge to a node v in row $t+1$. Let u be embedded in node (ℓ, u') and v in node (ℓ, v') of A. We simulate the edge between u and v by a path of length 2 between (ℓ, u') and (ℓ, v'). This path consists of a right BF_1 edge and a right BF_2 edge. The path is

$$(\ell, u') \stackrel{BF_1}{\rightarrow} (\ell+1, u') \stackrel{BF_2}{\rightarrow} (\ell, v') \ .$$

Note that this plugs in an admissible BF_2 edge between $(\ell+1, u')$ and (ℓ, v').

Next we embed the forward edges between distinct building blocks, i.e., the edges between adjacent AKS nodes embedded in level ℓ and level $\ell+1$. Let $C_L \subseteq L$ and $C_R \subseteq R$ be two sets of nodes that are arranged *symmetrically*, i.e., $C_R = \{(\ell+1, v+m/2) \,|\, (\ell+1, v) \in C_L$. Define $m' := |C_L \cup C_R|$. Furthermore, let $C \subseteq A$ be a set of nodes on level ℓ of size m'. Suppose none of the BF_2 edges that we have plugged in until now is incident to a node in C_L, C_R, and C, and suppose we plug in an arbitrary matching of BF_2 edges between $C_L \cup C_R$ and C. Then these edges are admissible BF_2 edges. Now define C_L to be the set of nodes in L that should be connected to nodes in A, and define C_R to be the set of nodes in L that should be connected to nodes in A. We assume that C_L and C_R are arranged symmetrically. This can be done because the embedding into the two submultibutterflies below L and R can be assumed to be isomorphic. Unfortunately, we have already fixed the BF_2 edges incident to the nodes in C_R for embedding the forward edges inside the building blocks. Therefore, we have to modify the above embedding slightly. Define $\hat{C} \subseteq A$ to be the set of m' nodes above the nodes in C_L and C_R, i.e., $\hat{C} := \{(\ell, v) \,|\, (\ell+1, v) \in C_L \cup C_R\}$. We change the above embedding so that no AKS node is mapped onto the nodes in $\hat{C}$. This has a nice consequence: the BF_2 edges incident to C_L and C_R are not used for implementing the forward edges inside the building blocks. Finally, define $C \subseteq A$ to be the set of nodes that must be connected to the nodes in C_L and C_R. Then we can simulate these edges by an appropriate matching of BF_2 edges between $C_L \cup C_R$ and C that completes the description of the embedding from P into A. The load of our embedding is 1, the dilation is 2, and the congestion is 1 since no multibutterfly edge is used for simulating more than one AKS edge.

In order to implement the above embedding we have to assume that the size of A is not to small, or the other way round, that the size of partition P is not to big, i.e., the equation $h \cdot |P| + |C| \leq |A|$ must be satisfied. Note that $|C| \leq 2 \cdot |P|$, $|P| \leq |P_\ell|/2^\ell$, and $|A| \leq m/2^\ell$, with m denoting the number of nodes on a multibutterfly level. Thus, the above description can be implemented if

$$(h+2) \cdot |P_\ell| \leq m, \tag{1}$$

for every tree level ℓ of the AKS network. This defines a constraint on the relationship between the size of the AKS network and the multibutterfly. In order to investigate this constraint, we first calculate some properties of the AKS network.

Properties of the AKS network. 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. Then

$$C_\ell(t) = 2^\ell \cdot c_\ell(t) = \left(1 - \frac{1}{4A^2}\right) \cdot n \cdot \nu^t \cdot (2A)^\ell \ .$$

Note that the cold storage simulates a bag half the size of the root (the root's parent), one quarter the size of the root (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 $-\infty \leq \ell \leq \log n$ and $t \geq 0$.

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. Further, we say tree level ℓ is *above* tree level ℓ' if $\ell < \ell'$, and ℓ is *below* ℓ' if $\ell > \ell'$. 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 $A_\ell(t)$ to be the sum of the capacities of all tree levels above level ℓ and congruent to t. Then

$$A_\ell(t) \quad = \quad C_{\ell-2}(t) + C_{\ell-4}(t) + C_{\ell-6}(t) + \ldots$$

$$
\begin{aligned}
&= \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^t\cdot\sum_{i=-(\ell-2)}^{\infty}(2A)^{-2i}\\
&= \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^t\cdot(2A)^{\ell-2}\cdot\sum_{i=0}^{\infty}(4A)^{-i}\\
&= n\cdot\nu^t\cdot(2A)^{\ell-2}
\end{aligned}
$$

if $d \cong t$, and

$$
\begin{aligned}
A_\ell(t) &= C_{\ell-1}(t)+C_{\ell-3}(t)+C_{\ell-5}(t)+\ldots\\
&= \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^t\cdot\sum_{i=-(\ell-1)}^{\infty}(2A)^{-2i}\\
&= \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^t\cdot(2A)^{\ell-1}\cdot\sum_{i=0}^{\infty}(4A)^{-i}\\
&= n\cdot\nu^t\cdot(2A)^{\ell-1}
\end{aligned}
$$

if $d \not\cong t$. Suppose ℓ' is the partial tree level in stage t. Then the sum of the number of elements in the bags of the tree levels above ℓ is $A_\ell(t)$ if $\ell \le \ell'$. Of course, this sum can be at most n. As a consequence, if $A_\ell(t) \ge n$, then tree level ℓ is below the partial level. Define

$$t_0^\ell := (\ell-2)\cdot\log_\nu\left(\frac{1}{2A}\right) \quad .$$

Then tree level ℓ is below the partial level, and hence empty, in every stage $t \le t_0^\ell$, since

$$A_\ell(t) \ge n\cdot\nu^{t_0^\ell}\cdot(2A)^{-(\ell-2)} = n \quad .$$

Define

$$t_1^\ell := \ell\cdot\log_\nu\left(\frac{1}{2A}\right) \quad .$$

Then tree level ℓ is filled to its capacity in every congruent stage $t \ge t_1^\ell$, because

$$
\begin{aligned}
&A_\ell(t)+C_\ell(t)\\
&< \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^{t_1^\ell}\cdot(2A)^{\ell}+n\cdot\nu^{t_1^\ell}\cdot(2A)^{\ell-2}\\
&= n\cdot\nu^{t_1^\ell}\cdot(2A)^{\ell}\cdot\left(\left(1-\frac{1}{4A^2}\right)+(2A)^{-2}\right) = n \quad .
\end{aligned}
$$

which means that ℓ is above the partial level. Further, define

$$t_2^\ell := \log_\nu\left(\frac{r}{(1-\frac{1}{4A^2})\cdot n}\right)+\ell\cdot\log_\nu\left(\frac{1}{A}\right) \quad .$$

Then the splitting step of tree level ℓ is in the first congruent stage $t \ge t_2^\ell$. This is because

$$C_\ell(t_2^\ell) = \left(1-\frac{1}{4A^2}\right)\cdot n\cdot\nu^{t_2^\ell}\cdot(2A)^{\ell} = r \quad ,$$

which means that $C_\ell(t) \ge r$, for $t \le t_2^\ell$, and $C_\ell(t) \ge r$, for $t \le t_2^\ell$. Finally, we show that the AKS algorithm finishes with the splitting step in level

$$\ell^* = \left\lceil \log_2\left(\left(1-\frac{1}{4A^2}\right)\cdot\frac{n}{r}\right)\right\rceil$$

in stage T such that T is the smallest integer congruent to ℓ^* that satisfies

$$T \geq t_2^{\ell^*} = \left\lceil \ell^* \cdot \log_\nu \left(\frac{1}{2A}\right)\right\rceil = \log_2 n \cdot \log_\nu \left(\frac{1}{2A}\right) - \Theta(1) \ .$$

This is because $T = t_2^{\ell^*}$, and the number of elements stored in bags of levels below ℓ^* in stage T is 0 (since $C_\ell(T) + A_\ell(T) \geq n$), and because ℓ^* is the smallest level that satisfies these two conditions.

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 dimensional 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 1, i.e., $(h+2) \cdot |P_\ell| \leq m$, for every level ℓ of the AKS tree. We have to bound the size of each partition P_ℓ. We first assume ideal batch 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. This partition includes the root bags and the cold storage. The size of the root bag in an even stage t is $C_0(t)$, and in odd stages the size is 0. The size of the cold storage in a stage t is $A_0(t)$. Hence, we have

$$\begin{aligned} |P_0| &\leq \sum_{t=0}^{\infty} C_0(2t) + \sum_{t=0}^{\infty} A_0(t) \\ &= \left(\left(1 - \frac{1}{4A^2}\right) + \frac{1}{4A^2} + \frac{\nu}{2A}\right) \cdot n \cdot (2A)^\ell \cdot \sum_{t=0}^{\infty} \nu^{2t} \\ &= n \cdot \underbrace{\left(1 + \frac{\nu}{2A}\right)\left(\frac{1}{1-\nu^2}\right)}_{=: \kappa_1(\nu, A)} \ . \end{aligned}$$

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

- In each stage $t \cong d$ with $t_0^\ell < t \leq t_1^\ell$, the size of the tree level is at most $n - A_\ell(t)$.
- In each stage $t \cong d$ with $t > t_1^\ell$, the size of the tree level is at most $C_\ell(t)$.
- In all other stage the size is 0.

Thus, for $1 \leq \ell \leq \ell^*$, we have

$$\begin{aligned} |P_\ell| &\leq \sum_{t=0}^{\left\lceil \frac{t_1^\ell - t_0^\ell}{2} \right\rceil} (n - A_\ell(t_0^\ell + 2t)) + \sum_{t=0}^{\infty} C_\ell(t_1^\ell + 2t) \\ &= n \cdot \left\lceil \frac{t_1^\ell - t_0^\ell}{2} \right\rceil - n \cdot \nu^{t_0^\ell} \cdot (2A)^{\ell-2} \cdot \sum_{t=0}^{\left\lceil \frac{t_1^\ell - t_0^\ell}{2} \right\rceil} \nu^{2t} \\ &\quad + n \cdot \nu^{t_1^\ell} \cdot (2A)^\ell \cdot \sum_{t=0}^{\infty} \nu^{2t} \\ &= n \cdot \underbrace{\left(\log_\nu \left(\frac{1}{2A}\right) + 1 + \frac{1}{4A^2 \cdot (1-\nu^2)}\right)}_{=: \kappa_2(\nu, A)} \ . \end{aligned}$$

under the assumption that $r = 0$. Now we assume $r > 0$. That means that the size of tree level ℓ is increased by $A_\ell(t)$ in each stage t from the splitting stage of level ℓ to the splitting stage of level $\ell + 1$. Therefore the above bound is increased by at most

$$\begin{aligned}\sum_{t=\lceil t_2^\ell\rceil}^{\lceil t_2^{\ell+1}\rceil+2} A_\ell(t) &\le \left(\log_\nu\left(\frac{1}{A}\right)+2\right)\cdot n\cdot \nu^{t_2^\ell}\cdot(2A)^{\ell-2}\\ &\le \left(\log_\nu\left(\frac{1}{A}\right)+2\right)\cdot\frac{r\cdot 2^{\ell^*-1}}{\left(1-\frac{1}{4A^2}\right)A}\\ &\le n\cdot\underbrace{\frac{\log_\nu\left(\frac{1}{A}\right)+2}{A}}_{=:\,\kappa_3(\nu,A)}\ .\end{aligned}$$

Up to now we have assumed that all bags have ideal sizes as real numbers. But the integer sizes can be bigger than the ideal ones. Fortunately, both values differ only slightly, i.e., each bag of ideal size b has integer size at most $b+2$ [28]. Consider a level ℓ. The number of congruent stages between stage t_0^ℓ and t_1^ℓ is at most $\log_\nu(\nu/2A)/2$, and the number of batches on level ℓ in a stage is at most $2^\ell \ge 2^{\ell^*} \ge (1-1/4A^2)\cdot n/r$. In each congruent stage $t_1^\ell \le t < t_2^{\ell+1}+1$, all bags on this level have ideal size at least $\nu r/A$, which is an upper bound on the bag size in stage $t_2^{\ell+1}+1$. Further, all bag sizes in stages before t_0^ℓ and after $t_2^{\ell+1}+1$ are 0. Thus, the correct integer size of a level is at most an additive of

$$\begin{aligned}&2\cdot n\cdot\frac{\log_\nu\left(\frac{\nu}{2A}\right)}{2}\cdot\frac{1-\frac{1}{4A^2}}{r}+n\cdot\frac{\frac{\nu r}{A}+2}{\frac{\nu r}{A}}\\ \le\ & n\cdot\underbrace{\left(\frac{\log_\nu\left(\frac{\nu}{2A}\right)\cdot\left(1-\frac{1}{4A^2}\right)+\frac{2A}{\nu}}{r}+1\right)}_{=:\,\kappa_4(\nu,A,r)}\end{aligned}$$

bigger than the ideal size.

Define $\bar\kappa(\nu,A,r)) := \max\{\kappa_1,\kappa_2+\kappa_3\}+\kappa_4$. Then it holds $|P_\ell| \le \bar\kappa\cdot n$, for every tree level ℓ. An AKS network can be embedded into a multibutterfly network with m nodes per level if Equation 1, i.e., $(h+2)\cdot|P_\ell| \le m$, 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, $m\ge M/(\log_2 m+1)$ and $n=N/(h\cdot T)$. In addition, $\log_2 m=\log_2 n+\Theta(1)$ and $T\ge\log_2 n\cdot\log_\nu\left(\frac{1}{2A}\right)-\Theta(1)$. Thus, we have

$$N \ge \frac{M\cdot h\cdot T}{(h+2)\cdot\bar\kappa\cdot(\log_2 m+1)} \ge \frac{M\cdot h\cdot\left(\log_2 n\cdot\log_\nu\left(\frac{1}{2A}\right)-\Theta(1)\right)}{(h+2)\cdot\bar\kappa\cdot(\log_2 n+\Theta(1))} = M/\kappa - o(M)$$

for $\kappa(\nu,A,r,h) := (1+2/h)\cdot\bar\kappa/\log_\nu\left(\frac{1}{2A}\right)$. (which is at most $1.462\ldots$. for $\nu=43/48$, $A=3$, $r=160$, and $h=36$ as suggested in [28]).

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 $0\le\ell\le d$. 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 tuples 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 tuples in R. Each tuple $(v_0,v_d)\in R$ 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.

Each multibutterfly node can store only a constant number of packets. We assume that the multibutterfly has $h-1$ additional levels $-(h-1), \ldots, -1$ which model the initial storage for the at most $h \cdot n$ packets. Let V_ℓ denote the set of nodes on level ℓ, for $-(h-1) \leq \ell \leq -1$. We assume that 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 $\Gamma(X) \cap V_{\ell+1} \geq \beta |X|$.

Upfal [35] presents an algorithm for routing a permutation, or 1-relation, in $O(\log n)$ steps. Our algorithm uses Upfal's algorithm as a subroutine.

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

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.

We assume that each input node stores h packets that are partitioned into L batches $B(0), \ldots, B(L-1)$, such that no more than αm packets from each batch are routed through any splitter of size m (!). 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 adjacent to one node. The algorithm works in iterations. 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 one after the other according to the color order. Thus, in each step, only one edge adjacent to each processor is activated. When an edge from node (ℓ, u) to node $(\ell+1, v)$ is activated, if node (ℓ, u) stores in its buffer a packet with higher priority than the packet stored in the buffer of $(\ell+1, u)$, the two nodes exchange packets. (An empty buffer is considered a packet with the lowest priority.) We extract the following Lemma from Upfal's analysis [35].

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 has reached its destination in time $O(\log n + L)$.*

For permutations, it is easy to split 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 split 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.

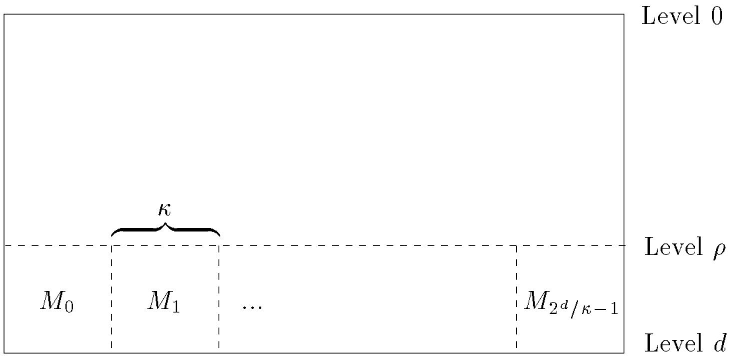

Figure 3: The submultibutterflies of size κ on the levels ρ to d.

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

$$\{(\ell, j) \mid \rho \leq \ell \leq d, \lfloor j/\kappa \rfloor = i\} \ .$$

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 3 illustrates the situation. 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 $\{(d,v) \mid v \pmod{L} = i\}$.
  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 $-(h-1) \leq \ell \leq d$, such that each node has to store at most a constant number of packets.

- *Phase 2:* Give each of the packets with the same destination a unique *rank*, i.e., for each M_i and each output node (d,v) of M_i, number the packets with destination (d,v) from 0 to $h-1$.

- *Phase 3:* Partition the packets into $L := \kappa \cdot \lceil 2/\alpha \rceil$ batches $B(i + j \cdot \kappa) := B(i,j)$ with $0 \leq i \leq \kappa - 1$ and $0 \leq j \leq \lceil 2/\alpha \rceil - 1$. $B(i,j)$ contains each packet p with rank i and a destination in $\{(d,v)|v \pmod{\lceil}) 2/\alpha\rceil = j\}$.
  Finally, complete the routing with Upfal's protocol according to the new batches.

Intuitively, we have split 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 destination.

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 Phase 1, the number of packets from a batch $B(j)$ passing through the i-th splitter on level ℓ of size m is at most

$$h \cdot |\{v \mid v \pmod{L} = j, \lfloor v/m \rfloor = i\}| \leq hm/L + h \leq \alpha m/2 + h\,.$$

Since the packets route only through the level 0 to $\rho - 1 = d - \log\kappa - 1$ in Phase 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 $\alpha m/2 + h \leq \alpha m$. In Phase 3, the number of packets passing through a splitter of size m is at most

$$m/\lceil 2/\alpha \rceil + 1 \leq \alpha m \ ,$$

for $m \geq 1/\alpha$. (We assume that splitters of size $m < 1/\alpha$ are completely connected.) As a consequence, Phase 1 and Phase 3 can be done in 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 $h + \log n$ 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. For each output node (d,u) in the submultibutterfly B, this can be done by a prefix computation. After this computation each node (ρ, v) on the input level of B knows the number of packets with destination (d,u) that are stored in the columns with smaller indices than v, i.e., the number of packets stored at nodes (ℓ, w) with $w \leq v$. Thus, node (d,u) can compute an disjoint range of ranks for the packets stored in its column.

κ prefix computations can be performed in time κ on each of the submultibutterflies. Thus, the ranks can be computed and distributed among the packets in the columns in time $O(h + \log n)$, which completes our proof. □

In the above algorithm, we have assumed that the value of κ is known in advance. In order to avoid this, the algorithm can double the value of κ beginning with some $\kappa \geq \log n$ and test for each κ if Phase 1 can be completed in the time stated above. Note that this increases the routing time by a factor of at most 2.

A more practical solution. When h is known in advance, and $h = O(\log n)$, 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 multibutterflies is $\Theta(n^2)$, replacing the submultibutterflies with the meshes of trees does not increase the asymptotic VLSI layout area.

6 Simulating expansion on a 2-folded butterfly

The set of edges in a concentrator of a 2-folded butterfly can be split into disjoint subsets such that the edges in each of these subsets forms a cycle. As a consequence, if we consider, e.g., the left concentrator on level 0, there exists an arbitrarily large subset $X\subseteq A_{0,0}$ with $|\Gamma(X)\cap A_{1,0}|\le|X|+1$. This means that a 2-folded butterfly has poor expansion properties. However, the following theorem shows that the effective expansion can be improved by simulating multibutterflies with higher degree.

Theorem 6.1 *For any $\beta<1/(4\alpha)$, there exists 2-folded butterfly A that can simulate with constant slowdown a multibutterfly B of the same size that has (α,β)-expansion.*

Proof: We describe a d-dimensional 2-folded butterfly A and an equal-sized multibutterfly B of degree $4k$ such that A can simulate B with constant slowdown. A and B will be constructed randomly, and we will prove that the probability that B has (α,β)-expansion is bigger than 0. This proves that there exists a multibutterfly B with appropriate expansion that can be simulated on a 2-folded butterfly A.

Consider the first k levels of the 2-folded butterfly A. We define these levels by describing the underlying butterfly networks BF_1 and BF_2, i.e., the two butterflies from which A can be 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 $1\le i\le k$ and $x\in\{0,1\}^k$, suppose $\phi_{k,x}$ is a permutation chosen randomly and uniformly from the set of permutations on $\{0,1\}^{d-k}$. Then each node (ℓ,v) with $v=(v_0,\ldots,v_{d-1})\in\{0,1\}^d$ is connected by a BF_2-edge to node

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

Intuitively, this edge flips randomly the last $d-k$ bits of the node labels. (The second BF_2-edge of the node which leads to level $\ell+1$ can be chosen arbitrarily.)

Next we define the first k levels of multibutterfly B with degree $2k$. Consider level ℓ of B with $0\le\ell\le k-1$. Suppose $\pi_{i,x}$ is a permutation chosen randomly and uniformly from the set of permutations on $\{0,1\}^{k-\ell}$, for $1\le i\le k$ and $x\in\{0,1\}^{d-(k-\ell)}$. Let (ℓ,v) be a node on level ℓ with $v=(v_0,\ldots,v_{d-1})\in\{0,1\}^d$. Define $x:=v_0,\ldots,v_{\ell-1}$, $y:=v_{\ell+1},\ldots,v_k$, and $z:=v_{k+1},\ldots,v_{d-1}$. Further, define $y'_i:=\pi_{i,(x,0,z)}(y)$, and $z'_i:=\phi^{-1}_{i,(x,0,y')}(z)$, for $1\le i\le k$. Intuitively, the π-permutations switch randomly the y-bits and the ϕ-permutations switch randomly 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 $1\le i\le k$. It is easy to check that all edges are inside the splitters and that each node on level $\ell+1$ is the endpoint of $2k$ edges. Thus, B is a multibutterfly with degree $4k$. Note that all edges in a concentrator on level ℓ are chosen independently (except that some of them are not allowed to end at nodes on level $\ell+1$ which have the same y- and z-bits).

The 2-folded butterfly A can simulate B with constant slowdown since there is a path in A of length at most $2k+1$ from (ℓ, v) to any adjacent node on level $\ell+1$. For $1 \leq i \leq k$ and $b, b' \in \{0,1\}$, this path can be constructed as follows

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

We now investigate the expansion of B. Consider one of the concentrators in the first k levels. It consists of a node set $A := A_{\ell,i}$ and a node set $B := A_{\ell+1,2i(+1)}$ with $0 \leq \ell \leq k-1$ and $0 \leq i \leq 2^\ell - 1$. Define $m := |A|$. The probability that all edges in these concentrator that are incident to nodes in a subset $X \subseteq A$ have their endpoints in a subset $Y \subseteq B$ is at most $(2|Y|/m)^{k \cdot |X|}$. As a consequence, the probability that the concentrator has no (α, β)-expansion is at most

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

We choose $k \geq (\beta \cdot \ln(1/(2\alpha\beta)) + \ln(4/\alpha) + 2)/\log(1/(4\alpha\beta))$. Then the above term that bounds the probability of a bad event in one concentrator by 2^{k-1}. Thus, the probability that all 2^{k+1} concentrators have (α, β)-expansion on the first k levels is greater than 0. Consequently, we can choose the edges of A so that A can simulate the first k levels of a Multibutterfly with $(\alpha, beta)$ expansion. The levels k to $d-1$ of A can be viewed as 2^k independent 2-folded Butterflies 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 an (α, β)-expansion property be embedded with constant load, congestion, and dilation, in an $O(N)$-node AKS network whose ϵ-halvers have an (α, β) (or better) expansion property?

2. What is the complexity of selecting the kth largest item from among M items on an N-node bounded-degree network for 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] L. A. Bassalygo and M. S. Pinsker. Complexity of an optimum nonblocking switching network without reconnections. *Problems of Information Transmission*, 9:64–66, 1974.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] R. E. Cypher and C. G. Plaxton. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 193–203, May 1990.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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

[26] 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.

[27] 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.

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

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] E. Upfal. An $O(\log N)$ deterministic packet routing scheme. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 241–250, May 1989.

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

[37] 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.