Real-Time Emulations of Bounded-Degree Networks

Bruce M. Maggs

School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213

Eric J. Schwabe

Department of Electrical and Computer Engineering, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208

Keywords: bounded-degree processor networks, real-time and work-preserving emulations

1 Introduction

In this paper, we survey the state of the art in *real-time network emulations*. In particular, we consider emulation schemes whereby a *host* network of one type can mimic, in a step-by-step fashion, any computation that can be performed by a *guest* network of another type. An emulation is called real-time if sizes of the guest and the host are equal, to within a constant factor, and the time required by the host and the time used by the guest are also equal, to within a constant factor. We restrict our attention in this paper to *bounded-degree* networks — that is, networks where each node has a fixed number of neighbors, independent of the number of nodes in the network. The emulations use a variety of techniques, from simple embeddings of one network structure into another to complex emulations utilizing redundant computation. In addition to the survey, we present new results on the emulation of area- and volume-universal networks and trees of meshes on butterfly networks.

[1] Email address: bmm@cs.cmu.edu. 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 and Sun Microsystems.
[2] Email address: schwabe@ece.nwu.edu. Supported in part by National Science Foundation Grant CCR–9309111.

The study of the ability, or inability, of networks of one type to perform real-time emulations of networks another other type is helpful in deciding which network to use as the underlying communication medium in a parallel computer. In particular, if a host network can perform a real-time emulation of a guest network, then little generality is lost if the computer is based on the host network rather than the guest, because any computation that could be performed by the guest can also be performed by the host with only constant slowdown. If such an emulation scheme is known, then the host network might be preferred if it compares favorably with the guest in some other respect, such as the expense or difficulty of constructing the network. On the other hand, if no real-time emulation scheme is possible, then there may be computations that the host cannot perform as quickly as the guest. If these computations are important, then the guest may be prefered over the host, even if it is more expensive in some respect. A real-time emulation scheme also gives an automatic method for translating any program designed for a computer based on the guest network into one that runs with only constant slowdown on the host network.

Real-time emulations are an instance of a broader class of emulations called *work-preserving* emulations. An emulation is work-preserving if the total work (processor-time product) performed by the host is within a constant factor of that expended by the guest in performing its computation. More formally, suppose that the emulation of T_G steps on an N_G-node guest network requires T_H steps on an N_H-node host network. The emulation is work-preserving if $T_H \cdot N_H = O(T_G \cdot N_G)$. Real-time emulations can be seen as the special case in which $N_H = O(N_G)$. Although a work-preserving emulation may not guarantee that $T_H = O(T_G)$, it is optimal in the sense that it achieves optimal speedup over a sequential emulation. In particular, the time for a work-preserving emulation by an N_H-node host is $O((T_G \cdot N_G)/N_H)$, i.e., (ignoring constant factors) the time for a sequential emulation divided by the size of the host. Of course, the host network may take longer to solve the problem than the guest, for only in real-time work-preserving emulations is there a guarantee of constant slowdown.

1.1 Network emulation strategies

The simplest notion of network emulation comes from the explicit embedding of the guest network into the host network. An *embedding* of a guest network G into a host network H is a mapping that maps the nodes of G to nodes of H and routes a path in H between the images of the end points of each edge in G. The *load* of an embedding is the maximum number of nodes of G mapped to a single node of H. The *dilation* of an embedding is the length of the longest path in H to which an edge of G is mapped. The *congestion* of an embedding is the largest number of images of edges of G that cross any single edge of H.

An embedding of G into H with constant load, dilation, and congestion implies the existence of a real-time emulation of guest network G by host network H, as long as $N_H = O(N_G)$. The host network performs a step-by-step emulation of the guest network, as follows. First, each host node emulates the local computation of all of the guest nodes that have been mapped to it. Then for each message that the guest sends across an edge, the host routes a message along the corresponding path. Because the load, dilation, and congestion are all constant, both of these actions require only constant time.

Embedding-based emulations are *static* in the sense that the mapping from the guest to the host is fixed over all time steps, so that the emulation of a guest node cannot be migrated from one host node to another, and the communication between guest nodes must take place over the fixed paths specified by the embedding. These emulations are called *Type 1* in [22]. More generally, an emulation may be *dynamic* — that is, there is either no fixed mapping of nodes of the guest network to the host, or the scheduling of the guest computation and communication in the host involves more than simply scheduling the embedded paths. Examples of dynamic emulations include the real-time emulations of normal hypercube algorithms [17, Chapter 3.1.3] on the shuffle-exchange and deBruijn networks [31], cube-connected cycles network [26], and butterfly network [29].

An even broader class of network emulations allows *redundant* computation — that is, multiple copies of each guest node may be emulated by the host. These emulations were first studied by Meyer auf der Heide [22]. Redundant emulations may be either static or dynamic. In a static redundant emulation, each copy of a guest node is mapped to a fixed host node, and is connected by a fixed path in the host to some copy of each of its neighbors in the guest. These emulations are called *Type 2* in [22], and were also studied by Fellows [11]. Emulations that are both dynamic and redundant, *Type 3,* were first considered by Meyer auf der Heide [22,23], and later in [13,24].

In addition to the real-time emulation schemes, lower bounds have been proved on the slowdown of an emulation of one type of guest network on another type of host network. These lower bounds indicate that in some cases a real-time emulation is not possible. The lower bounds are typically proved by comparing the diameter, bisection width, or expansion properties of the guest and host networks. Lower bounds on the slowdown of static redundant emulations are shown in [9]. Interestingly, many of the lower bounds have been proved for the more general case of dynamic redundant emulations [13,14,23,24,27].

We note that the notion of real-time emulation is transitive — that is, if network A can perform a real-time emulation of network B, and network B can perform a real-time emulation of network C, then network A can perform a real-time emulation of network C. The transitivity holds as long as it is applied only a constant number of times, as the slowdowns of the composed emulations will multiply.

2 Bounded-degree networks

2.1 Arrays

A k-dimensional *array* with side lengths $n_1, n_2, \ldots, n_k$ has $N = n_1 n_2 \cdots n_k$ nodes. Each node in the array as a distinct label $(x_1, x_2, \ldots, x_k)$, where $0 \leq x_i \leq n_i - 1$, for $1 \leq i \leq k$. If the array has *wraparound*, then for each dimension i, node $(x_1, \ldots, x_i, \ldots, x_k)$ is connected to node $(x_1, \ldots, x_i - 1 \bmod n_i, \ldots, x_k)$ and to node $(x_1, \ldots, x_i + 1 \bmod n_i, \ldots, x_k)$. If the array does not have wraparound, then for each dimension i, node $(x_1, \ldots, x_i, \ldots, x_k)$ is connected to node $(x_1, \ldots, x_i - 1, \ldots, x_k)$, unless $x_i = 0$, and to node $(x_1, \ldots, x_i + 1, \ldots, x_k)$, unless $x_i = n_i - 1$. An array without wraparound is also called a *mesh*. With wraparound, it is called a *torus*. An array in which $n_1 = n_2 = \cdots = n_k$ is called *square*. Otherwise, the array is *rectangular*.

Two and three-dimensional arrays have often been used as the underlying networks in parallel computers. These networks are attractive because they can be decomposed into smaller arrays of the same type (and hence built up from smaller arrays), and because they are easily packaged in two and three-dimensional space, respectively, using wires with lengths that are fixed and independent of the array size.

A 1-dimensional array without wraparound is called a *linear array*; with wraparound it is called a *ring*. Sekanina [30] showed that any connected network, including all of the networks discussed in this paper, can perform a real-time emulation of a linear array.

2.2 Trees and meshes of trees

The $M = 2^m$*-leaf complete binary tree* has $N = 2^{m+1} - 1$ nodes, labeled with the binary strings of length at most m. For all labels $x_{i-1} \cdots x_0 \in \{0,1\}^i$, $0 \leq i < m$, there is an edge between node $x_{i-1} \cdots x_0$ and nodes $x_{i-1} \cdots x_0 0$ and $x_{i-1} \cdots x_0 1$. Those nodes labeled with strings of length i are said to be in *level* i of the tree (the root node has the empty string as its label), and the nodes in level m are the *leaves* of the complete binary tree.

In order to construct the $M \times M$ *mesh of trees*, we begin with a set of M *row trees* and a set of M *column trees*, each of which is an M-leaf complete binary tree, and label each set of M M-leaf complete binary trees from 0 to $M-1$. Letting $bin(i)$ be the m-bit binary representation of i, for each $R, C \in \{0, \ldots, M-1\}$ we combine node $bin(R)$ of column tree C and node $bin(C)$ of row tree R into a single node. The resulting network (with total number of nodes $N = 3M^2 - 2M$) is the $M \times M$ mesh of trees. The construction extends in the natural way to k-dimensional meshes of trees for larger values of k.

One of the motivations for the mesh of trees is that it can emulate each step of a parallel random-access machine (PRAM) with M-processors and M-memory locations in $O(\log M)$ steps [17, Chapter 2.1.3]. The idea is to attach a processor to the root of each row tree and a memory location to the root of each column tree. The three-dimensional, $M \times M \times M$ mesh of trees is interesting because it can compute the product of two $M \times M$ matrices in $O(\log M)$ steps [17, Chapter 2.4.2].

2.3 Hypercubes and bounded-degree hypercubic networks

Each node in an $N = 2^r$-node hypercube consists of a distinct r-bit string, and two nodes are connected by an edge if their strings differ in precisely one bit position. The hypercube is *not* a bounded-degree network, because each node has r neighbors, and since $r = \log N$, the degree grows as the size of the network grows. We introduce it here because the hypercube has been shown to be capable of emulating many of the bounded-degree networks examined in this paper. The remainder of this section describes several popular bounded-degree networks that are closely related in structure to the hypercube.

The N-node *butterfly network with wraparound* has nodes consisting of all ordered pairs $\langle l, C \rangle$, where the *level* l is taken from the set $\{0, \ldots, r-1\}$ and the *column* C is an r-bit string. Hence $N = r2^r$. Node $\langle l, c_{r-1} \cdots c_{r-1-l} \cdots c_0 \rangle$ is connected to node $\langle l+1 \bmod r, c_{r-1} \cdots c_{r-1-l} \cdots c_0 \rangle$ by a *straight* edge, and to node $\langle l+1 \bmod r, c_{r-1} \cdots \overline{c_{r-1-l}} \cdots c_0 \rangle$ by a *cross* edge, where $\overline{c_{r-1-l}}$ denotes the complement of bit c_{r-1-l}. The butterfly network *without wraparound* is defined similarly, but with $N = (r+1)2^r$ and the mod r removed from the edge definitions (so that there are no edges from level r to level 0). In the butterfly without wraparound, the nodes in level 0 are called the *inputs* and the nodes in level r are called the *outputs*. In the butterfly with wraparound, the inputs and the outputs are identified to form single level (level 0). Each of these networks can be embedded into the other with constant load, dilation, and congestion, so that each can perform a real-time emulation of the other. The butterfly without wraparound is isomorphic to several other networks, including the omega network, the flip network, the baseline network, and the reverse baseline network [17, Chapter 3.8.1]

The *cube-connected cycles (CCC)* network was proposed by Preparata and Vuillemin [26]. Its structure is very similar to that of the butterfly and it is not difficult to show that each network can perform a real-time emulation of the other. In fact, the cube-connected cycles was later shown by Feldmann and Unger to be a subgraph of the butterfly with wraparound [10].

The nodes of the N-node *shuffle-exchange network*, where $N = 2^n$, consist of all n-bit strings. Node $x_n x_{n-1} \cdots x_2 x_1$ is connected to nodes $x_{n-1} x_{n-2} \cdots x_1 x_n$ and $x_1 x_n \cdots x_3 x_2$ by *shuffle* edges, and to node $x_n \cdots \overline{x_1}$ by an *exchange* edge.

The *deBruijn network* [17, Chapter 3.1.1] is closely related to the shuffle-exchange network and it is not difficult to show that each can perform a real-time emulation of the other.

2.4 Area- and volume-universal networks

Fat-tree networks were introduced by Leiserson [20], who showed that a fat-tree of area N could emulate any other network of area N with slowdown at most polylogarithmic in N. His original construction was later improved by reducing the slowdown [12,19] and making the emulation both deterministic and on-line [6].

The m^2-leaf fat-tree is defined as follows. The "coarse" structure of the fat-tree is that of a complete 4-ary tree, with m^2 leaf nodes at height 0, and a single root node at height $\log m$. There are $m^2/2^{2i}$ nodes at height i for each i, $0 \leq i \leq \log m$. Nodes at different heights contain different numbers of switches; in particular, each node at height i contains 2^i switches, so that the root node contains m switches and each leaf node contains one. Each node is labeled with a string of $2(\log m - i)$ bits, and each switch is labeled with a string of i bits. Connections between nodes at height i and its neighboring heights are as a follows:

Switch $x_{i-1} \cdots x_0$ in node $n_{2(\log m - i)-1} \cdots n_0$ is mapped to two switches at height $i+1$ and four switches at height $i-1$. (Unless, of course $i = 2 \log m$ or $i = 0$.) At height $i+1$, it is connected to switches $0x_{i-1} \cdots x_0$ and $1x_{i-1} \cdots x_0$ in node $n_{2(\log m - i)-1} \cdots n_2$. At height $i-1$, it is connected to switch $x_{i-2} \cdots x_0$ in nodes $n_{2(\log m - i)-1} \cdots n_0 00$, $n_{2(\log m - i)-1} \cdots n_0 01$, $n_{2(\log m - i)-1} \cdots n_0 10$, and $n_{2(\log m - i)-1} \cdots n_0 11$. (Note that these definitions are complementary, and actually cover each edge in the network twice.)

An N-node area-universal network based on a fat-tree is constructed as follows: Begin with an $\Theta(N/\log^2 N)$-leaf fat-tree, so that its root contains $\Theta(\sqrt{N}/\log N)$ switches. Attach a $\log N \times \log N$ mesh at the end of a chain of $\log N$ nodes to each leaf node of the network. (This construction is very close to that of Leighton, Maggs, Rao and Ranade [19], but it does not include certain enhancements that facilitated on-line message routing.) A similar construction using a slightly different fat-tree and attached three-dimensional meshes yields an N-node volume-universal network. The N-node area-universal network has area $\Theta(N)$ and can emulate any other network that can be laid out in area $O(N)$ with slowdown $O(\log N)$, in a packet routing model. A corresponding statement holds for the volume-universal network.

2.5 Trees of meshes

The two-dimensional tree of meshes was introduced by Leighton [15], who used it to demonstrate that there are N-node planar networks that require $\Omega(N \log N)$ VLSI layout area. The network was later used by Bhatt and Leighton [8] as part of a framework for solving VLSI layout problems using the notion of graph separators. The network also helped to inspire Leiserson's work on the area-universal fat-tree networks [20].

The tree of meshes is constructed as follows. Each node of a complete binary tree is replaced with a mesh, and each edge is replaced with a number of edges connected one side of the parent mesh to one side of its (smaller) child mesh. In particular, for a two-dimensional tree of meshes, the root is an $n \times n$ mesh, each of its two children is an $n \times n/2$ mesh, each of their children is an $n/2 \times n/2$ mesh, and so on. A mesh is connected to its two children by attaching each node on one of its sides to the corresponding node on one of the sides of the child. Their are a total of n^2 nodes at each level of the tree, and the total number of nodes is $N = 2n^2 \log n$. Higher-dimensional trees of meshes are easily constructed by the replacement of nodes of higher-degree complete trees with higher-degree meshes.

2.6 Expander-based networks

The AKS network, discovered by Ajtai, Komlos, and Szemerédi [2], is the only known n-input sorting network with depth-$O(\log n)$. The network is too complicated to define here (the reader is referred to Paterson's description [25]), but it has a leveled structure and requires certain expansion properties to hold in the connections between levels. This network sorts n keys using $O(n \log n)$ nodes. Leighton used the AKS sorting network in conjunction with a new sorting algorithm called Columnsort to construct an N-node bounded-degree network that can sort N keys in $O(\log N)$ time [16]. The resulting network is essentially an AKS network combined with a butterfly network.

Multibutterfly networks are similar in structure to butterfly networks, but the sets of connections between levels of the butterfly are augmented with additional edges (obtained by permuting the original connections) to guarantee certain expansion properties between levels. This idea was first used by Bassalygo and Pinsker [5] to construct optimum-size nonblocking networks. Multibutterfly networks have been shown by Upfal [32] and Arora, Leighton, and Maggs [3] to be powerful networks for solving general permutation routing problems. They have also been shown to be highly fault tolerate [18].

3 The current state of the art

Figure 1 summarizes known results on real-time emulations of the bounded-degree networks considered. Each arrow is directed from a guest network to a host network that can emulate it in real time. Solid arrows indicate static nonredundant emulations based on embeddings, while dotted arrows indicate static emulations that use redundant computation. Arrows are labeled with either a reference for the work that first established the emulation, "F" for a result that is either "folklore" or immediate from network definitions, or "X" for a result proved in this paper. A label of "G" followed by a reference indicates that the arrow is a straightforward generalization of the result in the reference. An arrow is unlabeled if the guest network contains the host network simply by definition. Arrows that follow from transitivity have been omitted for clarity. Note that the multibutterfly can perform a real-time emulation of all of the other networks considered in the paper.

Finally, we note that all of the networks considered here, except for the multibutterfly and AKS network, can be emulated in real time by a hypercube. The hypercube, however, has unbounded degree, and thus is not as scalable as the bounded-degree networks considered here.

4 New emulation results

Theorem 1 *A butterfly can perform a real-time emulation of an area- or volume-universal fat-tree.*

Proof First, we demonstrate an embedding of an m^2 leaf fat-tree into an order-$2m$ omega network with constant load, dilation, and congestion. This

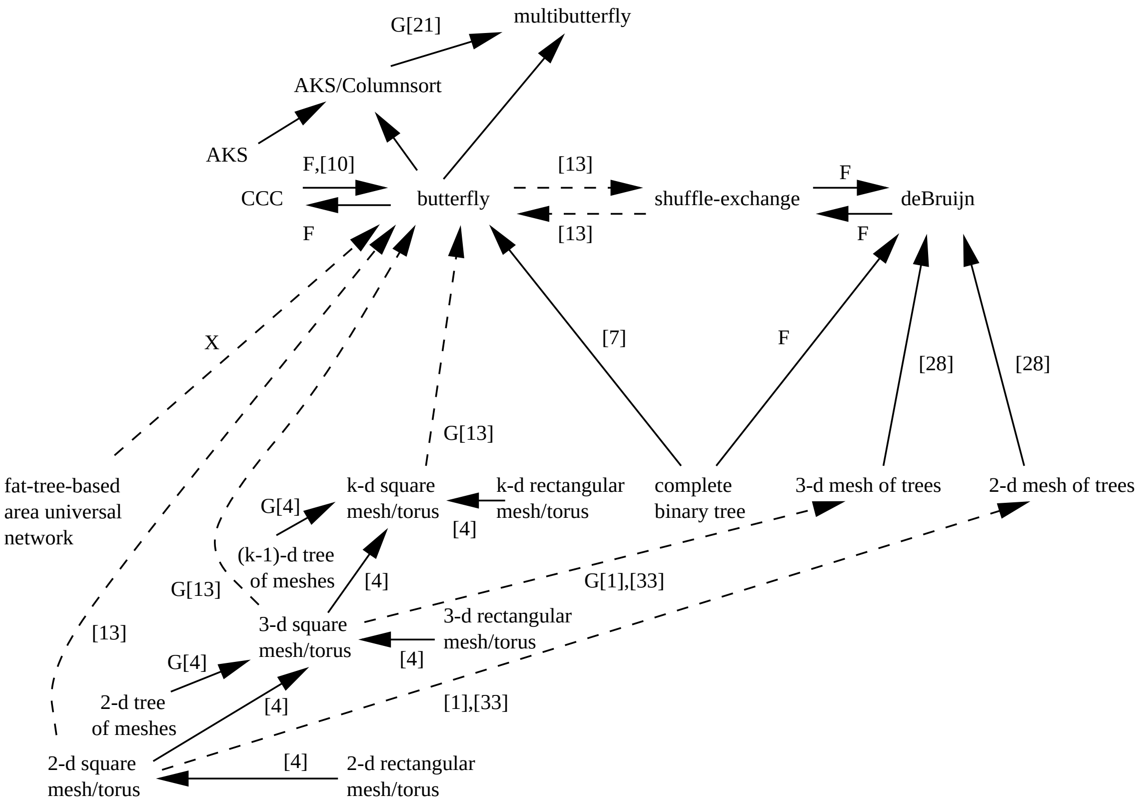

Fig. 1. Known positive results for real-time emulations.

result is then extended to give a real-time emulation of an area-universal fat-tree on a butterfly network. (The extension for volume-universal fat-trees is omitted.)

The order-$2\log m$ omega network consists of $2\log m+1$ levels of m^2 nodes each. The level of a node is a value between 0 and $2\log m$, inclusive, and each node within a level has a $2\log m$-bit string associated with it. Each node $x_{2\log m-1}\cdots x_0$ in level i (except for $i=0$) is connected to nodes $x_{2\log m-2}\cdots x_0 0$ and $x_{2\log m-2}\cdots x_0 1$ in level $i-1$. It is well-known that an order-$2\log m$ omega network is isomorphic to an $m^2\times(2\log m+1)$-node butterfly without wraparound [17, Chapter 3.8.1].

Lemma An m^2-leaf fat-tree can be embedded in an order-$2\log m$ omega network with load 1, dilation 2, and congestion 2.

Proof The desired embedding is as follows: A height-i switch $x_{i-1}\cdots x_0$ in node $n_{2(\log m-i)-1}\cdots n_0$ of the fat-tree is mapped to node $0x_{i-1}0x_{i-2}\cdots 0x_0 n_{2(\log m-i)-1}\cdots n_0$ in level $2i$ of the omega network. It is clear that each omega network node can have at most one switch mapped to it, since the level and the bit string identify at most one fat-tree switch that can be possibly mapped there.

Next, we describe the mapping of the edges of the fat-tree. Consider two arbitrary adjacent switches in the fat-tree. For some height i, bit strings $x_{i-1}\cdots x_0$ and $n_{2(\log m-i)-1}\cdots n_0$, and bits b_1 and b_0, these switches are $x_{i-1}\cdots x_0$ in node $n_{2(\log m-i)-1}\cdots n_0$ at height i and $x_{i-2}\cdots x_0$ in node $n_{2(\log m-i)-1}\cdots n_0 b_1 b_0$ at height $i-1$. The path taken from the embedded location of the former to the embedded location of the latter is

$$\left\langle 2i, 0x_{i-1}0x_{i-2}\cdots 0x_0 n_{2(\log m-i)-1}\cdots n_0\right\rangle \to \left\langle 2i-1, x_{i-1}0x_{i-2}\cdots 0x_0 n_{2(\log m-i)-1}\cdots n_0 b_1\right\rangle$$
$$\to \left\langle 2i-2, 0x_{i-2}\cdots 0x_0 n_{2(\log m-i)-1}\cdots n_0 b_1 b_0\right\rangle .$$

Clearly, the embedding has dilation 2. It is straightforward to verify that the congestion of this embedding is also 2, and therefore that the described embedding has the desired properties. This completes the proof of the lemma.

Now, consider an N-node area-universal fat-tree. We have shown that we can embed the 4-ary fat-tree (without the attached chains and meshes) into an $O(N)$-node butterfly network with constant load, dilation, and congestion. The resulting embedding maps all $\Theta(N/\log^2 N)$ leaves of the fat-tree to a single level of the butterfly. Using paths of length $\Theta(\log N)$ in the butterfly with constant total congestion, we can connect each leave to some node in a distinct subbutterfly with $\Theta(\log^2 N)$ nodes. Each of these subbutterflies can emulate a $\log N \times \log N$ mesh with constant slowdown. and the paths connecting each emulating subbutterfly to its corresponding leaf can perform an emulation of the chain of $\log N$ nodes with only constant slowdown. Putting these components together, the butterfly can thus emulate the area-universal fat-tree with only constant slowdown. The desired result for area-universal fat-trees follows.

(The proof of the volume-universal case is similar, with a slightly different embedding of the underlying fat-tree and emulation of three-dimensional meshes by the subbutterflies.) □

Theorem 2 *For fixed k, a $(k-1)$-dimensional tree of meshes can be embedded in a k-dimensional square mesh with constant load, congestion, and dilation.*

Proof: The basic idea is to first fold up the tree of meshes into a k-dimensional rectangular mesh (which contains it as a subgraph), and then embed the rectangular mesh into a square mesh using the technique of [4]. □

As a corollary, generalizing the technique of Koch et al. [13], the k-dimensional square mesh (and hence the $(k-1)$-dimensional tree of meshes) can then be emulated in real time by a butterfly.

5 Impossibility results

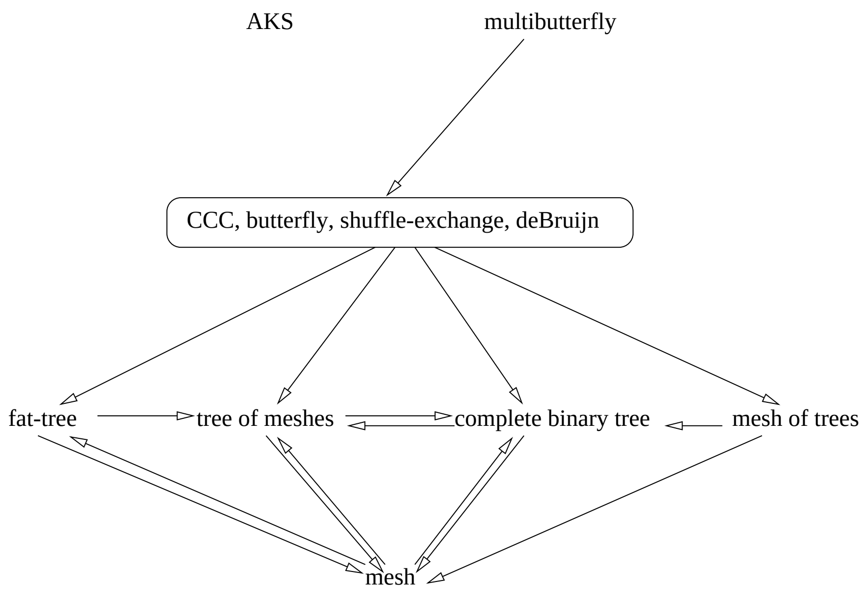

Fig. 2. Impossibility results for real-time emulations.

In addition to the positive results of the previous two sections, numerous negative results have also been established that show that it is impossible for a particular host to perform a real-time emulation a particular guest. Each arrow in Figure 2 points from a guest network to host network that has been proved to *not* be able to perform a real-time emulation of that guest. All of the "impossibility arrows" the figure follow from general lower bound results originally due to Koch et al. [13], except for the arrow from the multibutterfly to the butterfly and its equivalent networks, which follow from a result of Rappoport [27]. Because of their large bisection width, the AKS network and multibutterfly cannot be emulated in real time by any of the networks in the two lowest levels of the figure (these arrows have been omitted for clarity). It is also worth noting that there are bounded-degree networks that even a multibutterfly cannot emulate in real-time — for example, regular random networks and expander networks with bisection width $\Theta(N)$.

References

[1] A. Achilles. Optimal emulation of meshes on meshes of trees. In *Proceedings of EURO-PAR '95*, pages 193–204, Aug. 1995.

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

[3] 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.

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

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

[6] P. Bay and G. Bilardi. Deterministic on-line routing on area-universal networks. *Journal of the ACM*, 42(3):614–640, May 1995.

[7] 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.

[8] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, Apr. 1984.

[9] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. Reconfiguring arrays with faults part I: Worst-case faults. *SIAM Journal on Computing*, 26(6), Dec. 1997. To appear.

[10] 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.

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

[12] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In S. Micali, editor, *Randomness and Computation*. Volume 5 of *Advances in Computing Research*, pages 345–374. JAI Press, Greenwich, CT, 1989.

[13] 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.

[14] 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.

[15] F. T. Leighton. *Complexity Issues in VLSI*. MIT Press, Cambridge, MA, 1983.

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

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

[18] 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.

[19] 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.

[20] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, C–34(10):892–901, Oct. 1985.

[21] B. M. Maggs and B. Vöcking. Improved routing and sorting on multibutterflies. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 517–530, May 1997.

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

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

[24] 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.

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

[26] F. P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5):300–309, May 1981.

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

[28] E. J. Schwabe. Embedding meshes of trees into deBruijn graphs. *Information Processing Letters*, 43(5):237–240, 1992.

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

[30] M. Sekanina. On an ordering of the set of vertices of a connected graph. *Publications of the Faculty of Science, University of Brno*, 412:137–142, 1960.

[31] H. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C–20(2):153–161, 1971.

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

[33] L. Zhang. *Emulations and Embeddings of Meshes of Trees and Hypercubes of Cliques.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, Feb. 1995.