Empirical Evaluation of Randomly-Wired Multistage Networks

(Extended Abstract)

Tom Leighton[1,2] *Derek Lisinski*[2] *Bruce Maggs*[2]

[1]Mathematics Department and
[2]Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

In this paper, we present experimental data indicating that multistage interconnection networks with randomly positioned wires are likely to be substantially better for message routing applications than traditional multistage networks such as the butterfly. Data is presented for a variety of routing models, including store-and-forward routing, cut-through routing, and circuit switching, as well as for scenarios in which a potentially large number of switches are faulty. In most every situation, we find that randomly-wired networks outperform traditional networks with the same amount of hardware. In some cases, the differences are dramatic, particularly when several switches in the network are faulty. Overall, the data provides excellent empirical confirmation of recent theoretical work.

1 Background

Butterflies and similar hypercube-related graphs form the architectural basis of most parallel computers, including machines such as the BBN Butterfly, the Connection Machine, the IBM RP3 and GF11, the iPSC, and the NCUBE. The butterfly, in particular, is quite popular, and has been demonstrated to perform reasonably well in practice. An example of an N-input butterfly ($N = 8$) with depth $\log N = 3$ is shown in Figure 1. The nodes in this graph represent switches, and the edges represent wires. Messages are typically sent from the switches on level 0, called the *inputs*, to those on level $\log N$, called the *outputs*.

The message routing algorithm for a butterfly is quite simple. Each message simply follows the unique greedy path of length $\log N$ from its source input to its destination output. One problem with this algorithm (and hence the network) is that there is only one such path connecting any input to any output. Hence if some switch or edge along the path from input i to output j (say) becomes congested or fails, then communication between input i and output j will be disrupted. For example, if the edge from row 001 to row 011 connecting levels 1 and 2 of the 8-input butterfly shown in Figure 1 is congested (say by a message being routed from input 001 to output 011), then we can't send a message from input 101 to 010. (See Figure 2.)

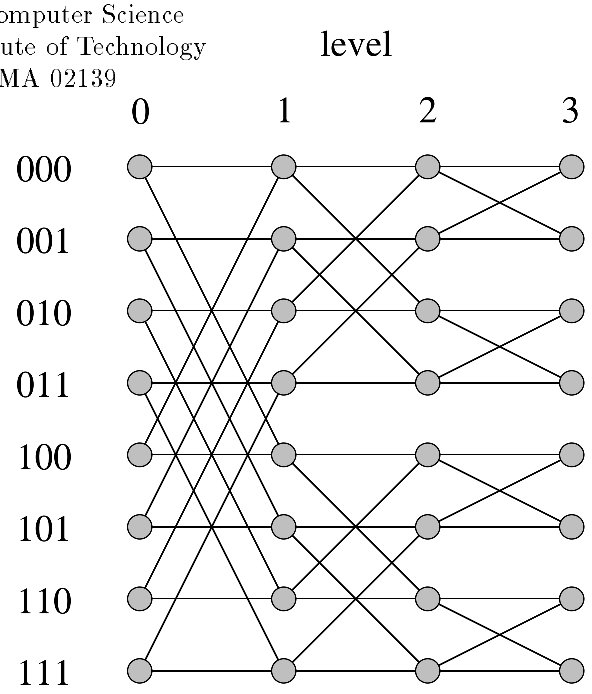

Figure 1: An 8-input butterfly network.

Because message traffic congestion is a common occurrence in real networks, the wires in butterfly networks are typically *dilated*, so that each wire is replaced by a *channel* consisting of 2 or more wires. For example, a 2-dilated butterfly is shown in Figure 3. Because it is harder to congest a channel than it is to congest a single wire in a butterfly, dilated butterflies are much better routing networks than simple butterflies [9, 10, 15].

A d-dilated butterfly can be thought of as d butterflies that are merged together. In general, however, there are many ways to merge d butterflies. In a recent paper, Upfal [16] proposed a more general way of merging the butterflies to produce a class of networks called *multibutterflies*. A multibutterfly is formed by gluing together butterflies in a somewhat unusual way. In particular, given 2 N-input butterflies G_1 and G_2 and a collection of permutations $\Pi = \langle \pi_0, \pi_1, \ldots, \pi_{\log N} \rangle$ where $\pi_l : [0, \frac{N}{2^l} - 1] \rightarrow [0, \frac{N}{2^l} - 1]$, a 2-butterfly is formed by merging the switch in row $\frac{iN}{2^l} + i$ of level

This research was supported by the Defense Advanced Research Projects Agency under Contracts N00014–87–K–825 and N00014–89–J–1988, the Air Force under Contract AFOSR–89–0271, and the Army under Contract DAAL–03–86–K–0171.

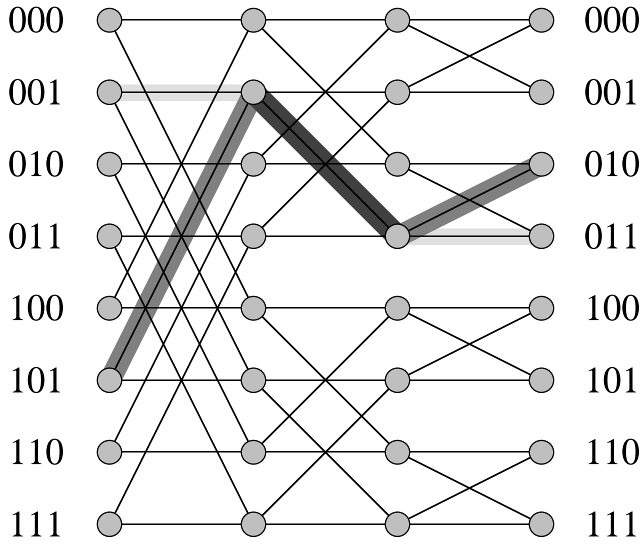

Figure 2: Path congestion.

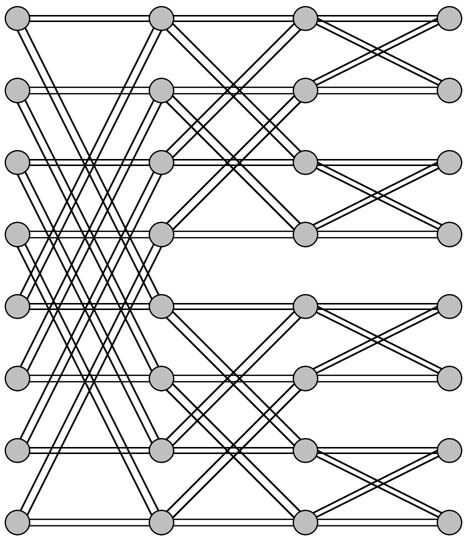

Figure 3: A 2-dilated butterfly.

l of G_1 with the switch in row $\frac{jN}{2^l} + \pi_l(i)$ of level l of G_2 for all $0 \le i \le \frac{N}{2^l} - 1$, all $0 \le j \le 2^l - 1$, and all $0 \le l \le \log N$. The result is an N-input graph with depth $\log N$ in which each switch has 4 inputs and 4 outputs. Of the 4 output edges at a switch, two are *up* outputs and two are *down* outputs (with one up edge and one down edge coming from each butterfly). Multibutterflies (i.e., d-butterflies) are composed from d butterflies in a similar fashion using $d-1$ sets of permutations, $\Pi^{(1)}, \ldots, \Pi^{(d-1)}$, resulting in a depth $\log N$ network with $2d \times 2d$ switches. For example, see Figure 4.

Butterflies and dilated butterflies are special cases of multibutterflies, and multibutterflies are contained in the much larger class of *delta* networks [11]. Delta networks are characterized by the up/down property of the edges that is used for message routing. The notion of up and down edges can be formalized in terms of splitters. More precisely, the edges from level l to level $l+1$ in rows $\frac{jN}{2^l}$ to $\frac{(j+1)N}{2^l} - 1$ in a multibutterfly form a *splitter* for all $0 \le l < \log N$ and $0 \le j \le 2^l - 1$. Each of the 2^l splitters starting at level l has $\frac{N}{2^l}$ inputs and outputs. The outputs on level $l+1$ are naturally divided into $\frac{N}{2^{(l+1)}}$ *up* outputs and $\frac{N}{2^{(l+1)}}$ *down* outputs. By definition, all splitters on the same level l are isomorphic, and each input is connected to d up outputs and d down outputs according to the butterfly and the permutations $\pi_l^{(1)}, \ldots, \pi_l^{(d-1)}$. Hence, any input and output of the multibutterfly (or delta network, in general) are connected by a single logical (up-down) path through the network, but each step of the logical path can be taken on any one of d edges. For example, see Figure 5.

Despite the fact that delta networks include a great many networks, dilated butterflies have remained the network of choice for most parallel machines. Recent work, however, suggests that this may be about to change. In fact, it now appears as though random delta networks (i.e., bounded-degree delta networks with randomly positioned wires in each splitter) are substantially superior to dilated butterflies for virtually any message routing application.

Randomly-wired multibutterfly networks have been discovered and rediscovered several times. In 1974, Bassalygo and Pinsker used multibutterfly and multi-Benes networks to construct the first nonblocking network of size $O(N \log N)$ and depth $O(\log N)$ [3]. (A multi-Benes network consists of back-to-back multibutterflies, in the same fashion that a Benes network consists of back-to-back butterflies.) In 1980, Fahlman proposed a related randomly-wired network called the Hashnet [6]. More recently, Upfal coined the term multibutterfly, and provided a simple deterministic algorithm for routing any permutation of N messages in $O(\log N)$ steps on an N-input multibutterfly network [16]. Leighton and Maggs simplified the analysis of Upfal's algorithm, and proved that a randomly-wired multibutterfly is an efficient routing network even when many of the switches are faulty [12]. Later, Arora, Leighton, and Maggs developed an a circuit switching algorithm for the multi-Benes network [2]. On an N-input network, the algorithm can establish a connection from any free input to any free output in $O(\log N)$ steps, even if many connections must be made simultaneously, and requests for connections are made asynchronously. Most recently, DeHon, Knight, and Minsky [5] have designed a 64-processor switching network using a randomly-wired delta network for processor to memory communications.

The proofs that randomly wired delta networks are

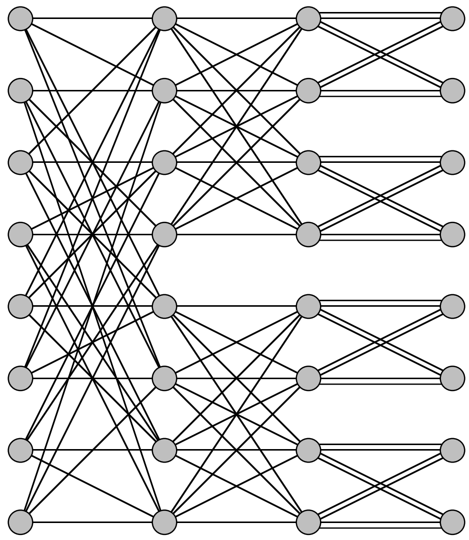

Figure 4: An 8-input 2-butterfly network.

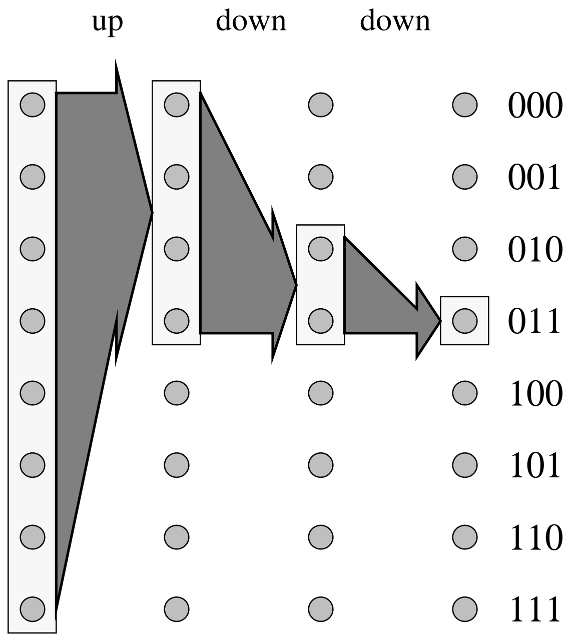

Figure 5: The logical path from any input to output 011.

worthwhile for message routing all make use of the fact that random interconnection patterns have a property known as expansion. In particular, an N-input splitter is said to have (α, β)-*expansion* if any set of $x \leq \alpha N$ inputs is connected to at least βx up outputs and βx down outputs, where $\beta > 1$. Splitters with expansion are good for routing since one must block βx outputs in order to block x inputs. In classical networks such as the butterfly, the reverse is true: it is possible to block $2x$ inputs by blocking only x outputs. When this effect is compounded over several levels, the effect is dramatic. In classical networks such as the butterfly, a single fault can block 2^l switches l levels back, whereas in a multibutterfly, it takes β^l faults to block a single switch l levels back. Splitters with expansion can be constructed deterministically, but randomized wirings typically provide the best possible expansion. Random wirings can be tested for expansion using the methods in [1].

2 Summary of our results

Although the results in [2, 12, 16] provide dramatic theoretical evidence that randomized delta networks are superior to traditional butterfly-based networks, the results hold only asymptotically, and it remains to be seen if the randomized networks will be truly practical. In this paper, we provide experimental data that supports the theoretical evidence. In particular, we find that randomly-wired delta networks almost always outperform dilated butterflies with comparable hardware. In some cases the observed differences are dramatic.

Our most startling data reveals that randomly-wired delta networks with 2 up outputs and 2 down outputs can withstand a very large number of switch failures before showing any ill effects. For example, a 1000-input network with random wires can withstand 250 random switch faults (this corresponds to a failure rate of 2.5%) without suffering any loss of input/output connectability and without suffering a significant loss in routing speed. Even with 1000 faults (corresponding to a 10% failure rate), the degradation in performance is minor. This level of fault tolerance is substantial, especially when compared to traditional networks such as the butterfly which start showing a degradation in performance after only a single fault.

We also found that for certain ranges of traffic load and message length, messages routed on randomly-wired networks experience significantly less average delay than do messages routed on a dilated butterfly with comparable hardware. Similarly, we found a much lower incidence of switch congestion for some ranges of traffic load when the randomly-wired networks are used for circuit switching. For other ranges of loading, the differences in performance between the networks is less substantial, though the randomly-wired networks almost always seem to perform at least as well as the dilated butterfly.

3 The experiments and data

This section presents data generated by three programs that simulate multibutterfly networks. The results of the first program show that a randomly-wired multibutterfly is capable of routing efficiently even when many of its switches are faulty. The second program shows that when not too heavily loaded, a multi-Benes network is nearly 100% nonblocking. The third shows that in the cut-through routing model, messages generally experience less delay on multibutterflies than on dilated butterflies. The three programs were written in PASCAL. The random numbers used in the simulations were generated using the minimal standard linear congruential generator from [14]. In all three programs, the randomly-wired networks were "cleaned up" after their creation to remove parallel edges between the same two switches wherever possible.

3.1 Fault tolerance

The first program tested the ability of a randomly-wired multibutterfly to tolerate faults. Four networks were compared: a normal butterfly, a 2-dilated butterfly, a randomly-wired 2-butterfly, and a randomly-wired 2-butterfly modified to improve its fault tolerance. Each network had $N = 1024$ inputs. The number of faults in the modified 2-butterfly varied from 0 to 1000.

The program primarily simulated store-and-forward routing. In the *store-and-forward* model, a message is treated as an atomic object. At each time step, a switch may send at most one message along each of its output edges. If necessary, messages may be queued at the switches. In this model, we measured the average number of steps required to route all of the messages to their destinations.

Messages were routed according to a simple greedy algorithm. At each time step, a switch was allowed to send a message along an edge provided that at the end of the previous step the number of messages in the switch at the end of the edge was at most 4.

We made two modifications to the 2-butterfly in order to make it more fault tolerant. First, we removed the last 2 levels and replaced each 4-input subbutterfly on these levels with a 4×4 complete bipartite graph. We then added a level -1 with 4 random matchings to level 0.

Faults were placed at random interior (i.e., not input or output) switches of the modified 2-butterfly. A non-faulty switch was declared faulty if either both of its up edges or both of its down edges led to faulty switches. Thus, faults could propagate from the last interior level back to the inputs. Because the faults were placed at random, the modified 2-butterfly could tolerate about $(N \log N)^{3/4}$ faults without any faults propagating back to the inputs.

We first ran a set of trials to compare the performance of the 2-butterfly to the normal butterfly and the 2-dilated butterfly when routing random and worst-case problems. We ran 4 types of trials. The first type consisted of routing a message from each input to a random destination, for a total of $N = 1024$ messages. The second consisted of routing a collection of 10 such random routing problems. The third type was the transpose permutation, a worst-case problem for the butterfly and dilated butterfly. In this problem the destination of each message is formed by rotating the binary representation of the origin of the message $\frac{1}{2} \log N$ positions in a circular fashion. The last type consisted of 10 of these transpose permutations. For each of these types, we varied the number of faults in the modified 2-butterfly from 0 to 1000.

	1	10	1T	10T
0 (nor)	14.0	26.0	38	272
0 (dil)	11.8	18.8	17	160
0	11.0	16.4	12.0	21.7
0*	12.0	18.1	11.0	17.1
1*	12.1	18.1	11.0	17.4
10*	12.1	18.4	11.1	18.6
100*	12.2	20.8	11.6	21.3
250*	12.5	23.5	12.2	23.8
1000*	14.1	34.2	13.7	37.0

Table 1: Store-and-forward completion time

The data is presented in Table 1. A square holds the average number of steps needed to route the problem or collection of problems. Each square represents the average of 50 trials. The first three rows show the data for a normal $N = 1024$ input butterfly, a 2-dilated butterfly, and a randomly-wired 2-butterfly, all executing the same routing algorithm. The last 6 rows are for the modified 2-butterfly with from 0 to 1000 faults.

Surprisingly, a randomly-wired 2-butterfly with up to 100 faults outperforms the fault-free 2-dilated butterfly on random problems, even though both networks consist of the same amount of hardware.

It is also possible to use this program to simulate circuit-switching. In the *circuit-switching* model, the goal of a message is to establish a dedicated path from its source to its destination. This path must be disjoint from the paths of all other messages; an edge can appear on at most one message's path. This model is most appropriate when the messages being transmitted are too long to be considered atomic objects. For circuit-switching we ran the same algorithm but instead measured the percentage of messages that reached their destinations without ever being delayed. These messages can be viewed as having successfully locked down paths from their sources to their destinations. The data is presented in Table 2. Once again, we found that a modified 2-butterfly with 100 faults outperformed a fault-free 2-dilated butterfly!

3.2 Circuit-switching

The second set of experiments tested several heuristics for establishing disjoint paths in multibutterfly and multi-Benes networks. We examined three types of networks: 4-dilated butterflies, randomly-wired 4-butterflies, and randomly-wired 4-Benes networks. Each network had $N = 64$ inputs. Attached to each input were 8 sources of messages, called *processors*, for a total of 512 sources. Each output had a pair of 4-wire

	1	1T
0 (nor)	45.1	3.1
0 (dil)	87.0	12.5
0	94.2	86.0
0*	88.6	96.9
1*	88.7	96.8
10*	88.3	96.7
100*	86.4	94.5
250*	83.0	90.9
1000*	64.1	69.6

Table 2: Average throughput

channels, each leading to a different destination, called a *memory bank*, for a total of 128 destinations. The size and configuration of the networks were chosen to model the Transit network proposed by Knight [5, 8].

The program simulated random message traffic patterns. At each time step, a processor that did not already have an active message in the system independently chose to send a message with probability p. Once created, a destination was chosen for the message at random. The head of the message then worked its way through the network until reaching its destination. Upon reaching its destination, the message held down its dedicated path for a specified amount of time, and then retracted the path in the reverse of the order in which it was locked down. Once the path was entirely retracted, the source processor became free to send another message. If the head of a message was ever blocked in the network, it immediately began to retract its path from that point. Upon reaching its source processor, the message waited to be sent again. At each time step, the source processor chose to resend the message with probability r.

Several heuristics were used to improve the percentage of messages that locked down their paths without ever being blocked. The first of these we call *status propagation*. At each time step, a switch was labeled *blocked* if each of its 4 edges in either the up or down directions was either in use, or led to a switch at the next level that was blocked at the previous time step. When routing through the network, a message gave priority to edges that led to unblocked switches. The second heuristic used was to examine the up or down edges at each switch in pseudorandom order.

The data is shown in Table 3. Each network was simulated for 10000 steps. The probability, p, of generating a message was varied from 0.006 to 1, while the resend probability, r, was held at 0.01. Once establishing its path, a message held it down for 100 steps before retracting it. As we varied p from 0.006 to 1, the utilization of the wires into the memory banks varied from about 30% to 50%. A square in this table represents the percentage of messages that successfully locked down their paths. This percentage was calculated only for messages whose destination was not blocked at the time of creation, i.e., there were at most 3 other messages with the same destination already in the network.

p	0.006	0.01	1
4-dil	86.2	78.7	70.0
4-but	95.6	89.2	74.5
4-Benes	98.6	95.1	75.9

Table 3: Circuit-switching success rate

In order to bring the success rate nearer to 100% for the 4-Benes network, we implemented a third heuristic, which we call the *unique neighbors* protocol. At each time step, each message sent a *request* to all of its neighbors in either the up or down direction at the next level. The message then gave priority to those edges that led to switches that received no other requests. By using the unique neighbors protocol on the 4-Benes network, fewer than 4 out of every 1000 messages were ever blocked due to switch congestion for $p = 0.006$ and $r = 0.01$.

3.3 Cut-through routing

The final set of experiments measured the average delay experienced by messages routing in the cut-through model. The *cut-through* model [7] is intermediate between the store-and-forward and circuit-switching models. Messages are not viewed as atomic objects, nor is it necessary for a message to lock down a path from its source to its destination. Instead, a message is modeled as having both a head and a tail. The content of the message lies between the head and the tail. As the head works its way through the network, it locks down the edges that it uses. The content of the message is queued at the switches on the path between the head and the tail. At each step, each edge on the path transmits one bit of the message towards the head. Each time the tail advances, it unlocks the last edge on the path. There is no bound on the number of bits that can be queued at any switch. Thus, if the head waits long enough at a switch, eventually the entire content of the message, including the tail, will catch up to it. By making the message content empty, the program could also be used to simulate store-and-forward routing.

The networks examined in this set of experiments were the 4-dilated butterflies and randomly-wired 4-butterflies from the circuit-switching experiments. As before, each network had 64 inputs, 512 processors, and 128 memory banks.

As in the circuit-switching experiments, the messages were sent to random destinations. At each step, each free processor independently chose to send a message with probability p. In these experiments, however, a processor was allowed to have more than one message in the network simultaneously. As soon as a processor sent the tail of a message, it was free to send another.

l	1		10			100		
p	0.5	0.6	0.05	0.1	0.2	0.01	0.02	0.05
2-dil	.33	1.3	.58	2.4	8.7	21.8	65.1	186
2-but	.14	.81	.30	1.7	6.9	14.9	46.4	167

Table 4: Average cut-through delay

For simplicity, we ignored the issue of routing replies back from the memory banks to the processors.

The average delay incurred by each message is shown in Table 4. For messages of length $l = 1$ and $l = 10$, we simulated the networks for 2000 steps, and measured the average delay of messages that arrived at their destinations between steps 500 and 2000. For message of length $l = 100$, we simulated the networks for 4000 steps.

4 Conclusion

Overall, we found that randomly-wired multibutterfly networks provide improved routing performance for multistage interconnection networks, thereby substantiating the earlier theoretical work in [2, 12, 16]. Although the randomly-wired networks look more complicated than standard butterfly-based networks, they are no harder to fabricate or program. Indeed, the routing algorithms are virtually identical, and the same switches can be used successfully in either network – only the wire connections differ. Even the wire lengths and total wire area [4, 13] of the randomly-wired networks are the same as for the dilated butterfly. Hence, randomly-wired networks should be seriously considered as an alternative to traditional butterfly networks by designers of parallel machines and communication networks.

5 Acknowledgements

The authors would like to thank Jose Fernandez for producing the figures.

References

[1] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[2] S. Arora, T. Leighton, and B. Maggs. On-line algorithms for path selection in a non-blocking network. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 149–158, May 1990.

[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] 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, April 1984.

[5] 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.

[6] S. E. Fahlman. The hashnet interconnection scheme. Technical Report CMU–CS–80–125, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, PA, June 1980.

[7] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communications switching technique. *Computer Networks*, 3(4):267–286, 1979.

[8] T. F. Knight, Jr. Technologies for low latency interconnection switches. In *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pages 351–358, June 1989.

[9] R. R. Koch. Increasing the size of a network by a constant factor can increase performance by more than a constant factor. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 221–230. IEEE Computer Society Press, October 1988.

[10] C. P. Kruskal and M. Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C–32(12):1091–1098, December 1983.

[11] C. P. Kruskal and M. Snir. A unified theory of interconnection network structure. *Theoretical Computer Science*, 48:75–94, 1986.

[12] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer*

Science, pages 384–389. IEEE Computer Society Press, October 1989.

[13] C. E. Leiserson. *Area-Efficient VLSI Computation.* MIT Press, Cambridge, MA, 1983.

[14] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, October 1988.

[15] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson. The monarch parallel processor hardware design. *Computer*, 23(4):18–30, April 1990.

[16] 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.