# Reduction of network cost and wiring in Ranade's butterfly routing

D. Cross, R. Drefenstedt, J. Keller

Computer Science/Department of Algorithmics and Architecture

**CS-R9243 1992**

# Reduction of Network Cost and Wiring in Ranade's Butterfly Routing

David Cross
*Department of Electrical Engineering and Computer Science*
*University of California, Berkeley, CA 94720, USA*

Reinhard Drefenstedt
*Computer Science Department*
*Universität des Saarlandes, 6600 Saarbrücken, Germany*

Jörg Keller
*CWI*
*P.O. Box 4079, 1009 AB Amsterdam, The Netherlands*

**Abstract**

The class of $n$–input butterfly networks is very important for the design of scalable parallel machines because of constant node degree, depth $\log n$ and the existence of routing algorithms capable of delivering $n \log n$ packets, forming a $\log n$–relation ($\log n$ packets destined to each output), in time $O(\log n)$ with constant length buffers. Implementations however show ugly wiring. We present a method that, in comparison to the obvious implementation, reduces the number of chips and the number of links between chips in Ranade's butterfly routing by a factor of more than two. The chips remain connected as a butterfly network. This reduction in space simplifies cooling, power supply and allows for shorter links thus reducing wire delay.

*1991 Mathematics Subject Classification:* 68M07, 68M10, 94C15
*1991 CR Categories:* B.4.3, B.4.4, B.7.1
*Keywords and Phrases:* computer architecture, packet routing algorithms, butterfly networks
*Note:* Part of this work was done while the third author was working at the Univ. des Saarlandes, Saarbrücken, Germany. This work is partially supported by the German Science Foundation (DFG) under SFB 124 – D4 and by the Dutch Science Foundation (NWO) through NFI Project ALADDIN under Contract number NF 62–376. D. Cross is currently with Mentor Graphics Corporation, 1001 Ritter Park Drive, San Jose 95131. This article has been accepted for publication in *Information Processing Letters*.

## 1 Introduction

Interconnection networks and routing algorithms for them are important parts of parallel architectures and therefore should fulfil certain requirements.

First, an interconnection network should be scalable, i.e. each network node should have a constant degree and only contain constant length buffers. Then one type of network node can be used to build networks of arbitrary size $n$. In order to keep the design of nodes simple, the degree should be as small as possible. In addition, the network should consist of as few nodes as possible.

Let us assume a network with $n$ inputs and outputs. A routing algorithm for such a network should provide a maximum throughput of delivering $n$ packets in constant time (provided that these $n$ packets form a permutation). The latency to deliver one packet should be kept at a minimum. Therefore it should not exceed the diameter of the network by more than a constant factor.

The constant node degree implies that the depth of such a network and thus the time to deliver a packet is at least $\log n$. In order to fulfil the above throughput criterion of $n$ packets in constant time together with a minimum latency, the routing algorithm must be capable of delivering $n \log n$ packets in time $O(\log n)$ provided that the packets form a $\log n$–relation. The term $\log n$–relation means that $\log n$ packets are fed into each input and that exactly $\log n$ out of these $n \log n$ packets are destined for every output.

Not all networks fulfil all requirements. Binary trees, as an example, have a constant degree, depth $\log n$ and consist of $n$ nodes which is optimal. But there are no routing algorithms that fulfil the other requirements.

A butterfly network with $n = 2^u$ inputs and outputs is a network with nodes of degree 2 that has depth $u + 1$ and consists of $n \cdot (u + 1)$ nodes. Thus it fulfils the requirement of scalability.

Ranade was the first to develop a randomized packet routing algorithm for butterfly networks that delivers a $\log n$–relation in time $O(\log n)$ and that only needs constant length buffers [5]. (Pippenger published an algorithm with similar properties three years earlier [4], but that one could end in a deadlock.) Ranade's algorithm fulfils the requirements of throughput and latency.

Because of the above properties, butterfly networks are very important in the design of scalable parallel architectures.

Ranade used his algorithm for the design of a very elegant emulation of a shared memory parallel machine on a processor network [6]. A reengineered version of his emulation was shown to have an emulation overhead of $O(\log n)$ where the constant involved is very small [2]. This makes the emulation interesting for practical use. Before, shared memory emulations were thought to have very large constant factors involved. Therefore they only seemed to be of academical interest. Because shared memory parallel machines are easier to program than distributed memory machines, they could become a serious competitor to the latter.

However, network design for a machine that implements Ranade's emulation is a heavy part, especially when links get wide to improve throughput by parallel transmission. We will present a method that in a butterfly network run with Ranade's algorithm reduces the number of network chips and the number of links between chips by a factor of more than two, compared to obvious implementations. The functionality and the performance of the network are unchanged, the network chips remain interconnected in a butterfly pattern. Because the network needs less space, links get shorter. This reduces delays on wires and allows to increase speed. Engineering aspects such as cooling or power supply get simpler, too.

In section 2 we will sketch the design of a network node that implements Ranade's routing algorithm. We will discuss difficulties in mapping nodes to chips, mainly low gate utilization in chips and pin count restrictions.

Section 3 shows how a slightly different mapping of nodes to chips doubles gate utilization. This implies that we need only half the number of chips and furthermore half the number of links between chips.

## 2 Original network nodes

We assume that packets consist of an address specifying the output, one word of data and control information. At the beginning $\log n$ packets are fed into each input. These packets are sorted by their addresses. The sorted order is kept during routing.

A node behaves as follows: If two packets are waiting in the input buffers of the node, the one with the smaller address is transmitted. The address also specifies the output along which the packet has to be sent.

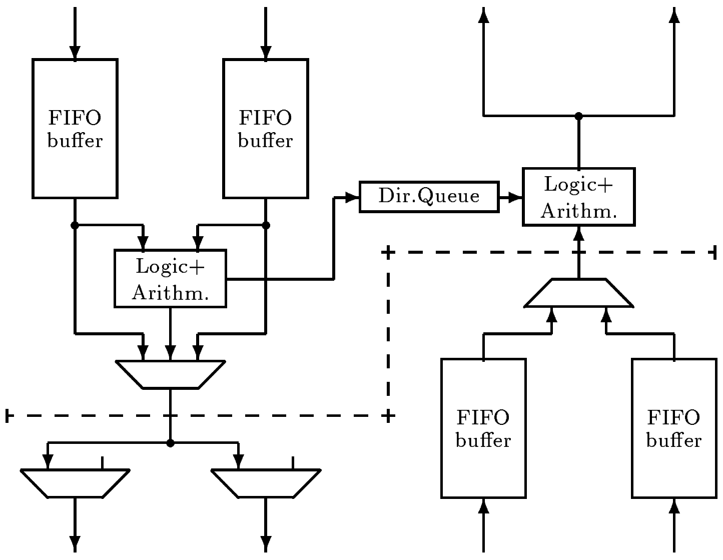


FIGURE 1. Datapaths of a Network Node

If only one input buffer contains a packet, this one has to wait. If the packet would be sent, a packet with a smaller address could arrive at the other input at a later time and the order would be destroyed. To avoid unnecessary waiting in nodes, GHOST packets are introduced. Each time a packet is transmitted along one output, a GHOST packet is transmitted along the other output. The GHOST packet also carries the packet's address but different control information. A GHOST packet arriving in an input buffer guarantees that only packets with an address larger than the GHOST's address will arrive on that input in the future.

Some of the packets can contain requests for a device on an output of the network to send back an answer, e.g. requests to a memory module to read the contents of a cell. The answers only consist of data. Their way through the network is determined by keeping track of the requests' path. This is done by maintaining a direction queue in each node. For each request that passes the node, input and output are recorded. As the sorted order guarantuees that answers will arrive in the same order as the requests passed, this information can be used to send back the answers.

A network node that implements these functions is shown in figure 1. We will ignore the dash line in this section. In practice, each address and data word will consist of 32 bit, the control information will consist of 8 bit. In order to have a high speed transmission, each link from one node to another should have a width of 72 bit in one direction to transmit a whole packet and 32 bit in the opposite direction to transmit an answer. This causes severe problems if one node should be realized by one chip because of pin restrictions. Custom chips can be obtained with up to 240 pins at reasonable prices. This forces sending packets in two flits. But compared to pin count the gate utilization is very low. This means that most of the silicon on the network chips is wasted.

To increase the gate/pin ratio of the network chips one can implement a small subnetwork with $i > 2$ inputs and outputs instead of one node on one chip. The number of gates grows proportionally to $i \log i$, the number of nodes in the subnetwork. The number of pins only grows linearly in $i$. Therefore, this improves the gate/pin ratio. The situation for $i = 4$ and links of width $w$ is shown in table 1. But unfortunately this improvement can only be achieved by either having more pins or by making links smaller. The first proposal cannot be realized because of the pin restriction. The second proposal reduces throughput because packets now have to be transmitted in twice as many flits as before.

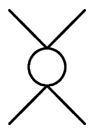
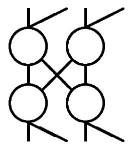
| type | t1 | t2 |
|---|---|---|
| outline |  |  |
| nodes per chip | 1 | 4 |
| number of pins | $4w$ | $8w$ |
| gates per chip | $g$ | $4g$ |
| gate/pin ratio | $\frac{g}{4w}$ | $\frac{g}{2w}$ |

TABLE 1. Mapping nodes on chips

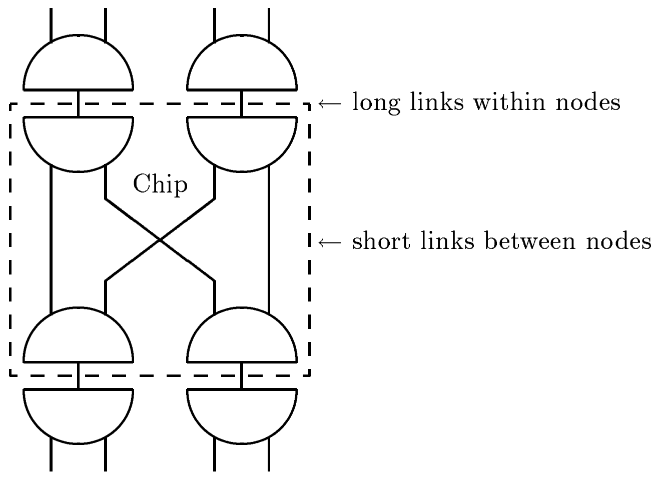


FIGURE 2. Partitioning of Network Nodes

## 3 Different Mapping

Let us again take a look at a network node. Although the node has two inputs and two outputs, it can be cut into two halves such that only one link crosses the cut. The cut is shown as a dash line in figure 1. We now implement a $2 \times 2$–butterfly in one chip but take only the lower part of the nodes of the first stage and the upper part of the nodes of the second stage. Figure 2 shows the partitioning. The resulting chip has as many pins as the original one but uses twice the number of gates. It follows that we need only half the number of chips in comparison to the original design.

We notice that the original algorithm uses only one input per node in the first network stage and only one output per node in the last stage. The second input is always filled with a ghost of lowest priority. Therefore the upper node parts in the first stage and the lower node parts in the last stage are not necessary. This means that we only need $(n/2) \cdot \log n$ instead of $n \cdot (\log n + 1)$ chips. Because these chips have two inputs and outputs just as the original ones, we also need less than half the number of links between chips.

We note, that although we now have long links within network nodes (between upper and lower part) this does not increase node cycle time and thus decrease throughput (see [3]).

It remains to be shown that the new chips are interconnected as a butterfly. This can be done by the following argument:

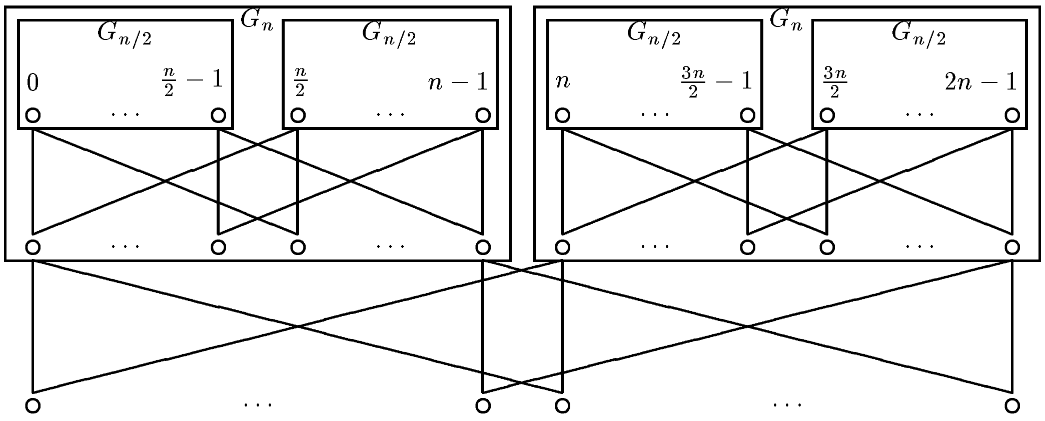


FIGURE 3. Construction of $G_{2n}$

A butterfly network with $n = 2^u$ inputs and outputs is a graph $G_n$, consisting of $u + 1$ stages of nodes with $n$ nodes per stage. $G_n$ is inductively defined as follows: $G_1$ consists of a single node, $G_n$ can be constructed by taking two copies of $G_{n/2}$ and $n$ additional nodes that form stage $u$. Nodes $i$, $0 \leq i < n/2$, in stage $u - 1$ of the two smaller butterflies are connected to nodes $i$ and $i + n/2$ in stage $u$. The nodes in one stage of the second copy of $G_{n/2}$ then are renumbered. The induction is illustrated in figure 3.

If we partition the nodes as done above we shrink each subgraph $G_2$ of $G_n$ to a new node and connect two nodes in this new graph $G'_n$ if the subgraphs share one node in $G_n$. We show by induction on $n$ that $G'_n = G_{n/2}$.

**Base:** It is obvious that $G'_4 = G_2$ (see also figure 4).

**Step:** Assume that the assumption holds for some $t = n/2$. We show that $G'_n = G_{n/2}$. To do that we recall the inductive definition of a butterfly network $G_n$. By the induction hypothesis we know that we can shrink both subnetworks $G_{n/2}$ to butterflies $G_{n/4}$. Shrinking the subnets $G_2$ of the last stage of $G_n$ results in $n/2$ nodes in the last stage of $G'_n$. We now know that $G'_n$ is constructed by taking two subnets $G_{n/4}$ and $n/2$ nodes as an additional stage. We just have to prove that the $i$th and the $(i + n/4)$th nodes in this last stage are connected to the $i$th nodes in the last stages of the two subnets $G_{n/4}$, $0 \leq i < n/4$. Then $G'_n = G_{n/2}$. But this is obvious from the inductive definition of $G_n$ if we look how $G_n$ is constructed from $G_{n/4}$ subnets. The $i$th subnets $G_2$, $0 \leq i < n/4$, in the last stages of both graphs $G_{n/2}$ are connected to subnets $G_2$ with numbers $i$ and $i + n/4$ in the last stage of $G_n$. ∎

ACKNOWLEDGEMENTS

We want to thank Wolfgang Paul for encouraging us to publish this work. We also want to thank Günter Hotz to whose honour an early version of this was published [1]. Last but not least we thank Dany Breslauer and John Tromp for discussions about butterfly networks.

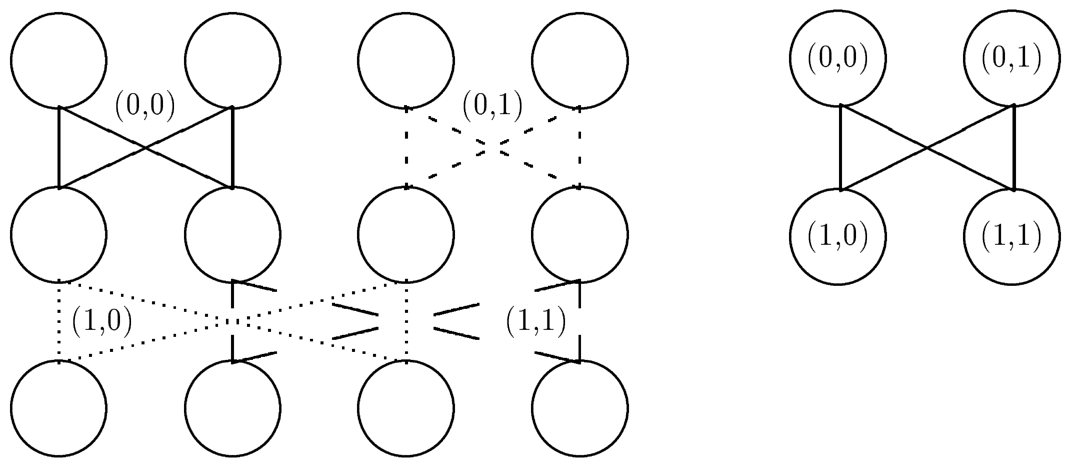


FIGURE 4. Construction of $G'_4$ from $G_4$

REFERENCES

[1] F. Abolhassan, R. Drefenstedt, J. Keller, W. J. Paul, D. Scheerer, On the physical design of PRAMs, In J. Buchmann, H. Ganzinger, W. J. Paul, (Eds.), *Festschrift zum 60. Geburtstag von Günter Hotz* (Teubner, 1992) 1–19.

[2] F. Abolhassan, J. Keller, W. J. Paul, On the cost–effectiveness of PRAMs, in: *Proc. 3rd Symposium on Parallel and Distributed Processing* (IEEE, 1991) 2–9.

[3] R. Drefenstedt, D. Schmidt, On the physical design of butterfly networks for PRAMs, in: *Proc. FRONTIERS '92, Symposium on the Frontiers of Massively Parallel Computation* (IEEE, 1992).

[4] N. Pippenger, Parallel Communication with limited Buffers, in: *Proc. 25th Symposium on Foundations of Computer Science* (IEEE, 1984) 127–136.

[5] A. G. Ranade, How to emulate shared memory, in: *Proc. 28th Symposium on Foundations of Computer Science* (IEEE, 1987) 185–194.

[6] A. G. Ranade, S. N. Bhatt, and S. L. Johnson, The Fluent Abstract Machine, in: *Proc. 5th MIT Conference on Advanced Research in VLSI* (1988) 71–93.