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

Reinhard Drefenstedt and Jörg Keller
Computer Science Department, Universität des Saarlandes, 6600 Saarbrücken, Germany

Abstract

We investigate implementations of butterfly networks. Obvious mappings of network nodes to chips lead to implementations with expensive wiring. We consider Ranade's butterfly routing algorithm. For this algorithm, we present a new mapping of network nodes to chips. This mapping only needs half the number of chips and links between chips. The chips' interconnections still form a butterfly network.

Keywords: computer architecture, packet routing algorithms, butterfly networks.

1 Introduction

Interconnection networks are important parts of parallel architectures, and therefore should provide high throughput with small delay. All network nodes should be identical. They should have constant degree, and the routing algorithm should only require constant length buffers per node. Then one kind of network node can be used to build networks of arbitrary size. A network that meets these requirements is the *butterfly network.*

Definition 1 *A* butterfly network *with $n = 2^u$ inputs and outputs is a graph G_n that consists of $u+1$ stages with n nodes per stage. G_1 consists of a single node, G_{2n} can be constructed by taking two copies of G_n and $2n$ additional nodes that form the last stage of G_{2n}. Node i, where $0 \leq i < n$, in the last stage of the smaller butterflies is connected to nodes i and $i+n$ in stage $u+1$. The construction is shown in fig. 1.*

Ranade was the first to develop a randomized packet routing algorithm for butterfly networks that meets the requirement of constant length buffers [6]. (Pippenger published a similar result in [5], but his algorithm could end in a deadlock.)

Ranade used his algorithm for the design of a very elegant emulation of a shared memory parallel machine on a processor network [7]. The emulation overhead is $c \log n$. A reengineered version of his emulation was shown to have an emulation overhead where c is very small [2], making the emulation interesting for practical use. Prior to [2], shared memory emulations were thought to be impractical because of large constant factors involved. Because shared memory parallel machines are easier to program than distributed memory machines, they could become a serious competitor to the latter, if there are prac-

*This work was partly funded by the German Science Foundation (DFG) in SFB 124, TP D4. D. Cross is currently with Mentor Graphics Corporation, 1001 Ritter Park Drive, San Jose 95131.

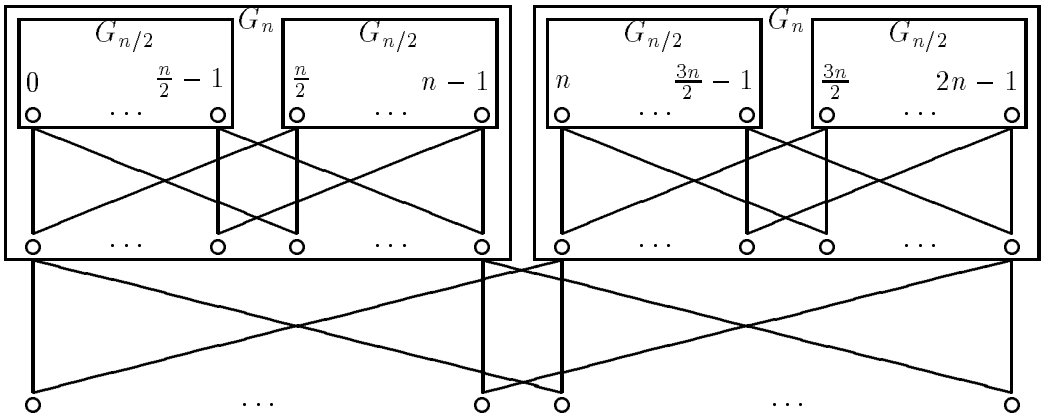


Figure 1: Construction of G_{2n}

tical emulations.

Consider a butterfly network G_n, run with Ranade's algorithm. We investigate implementations of G_n, i.e., a mapping M of network nodes to chips with a fixed number p of pins. We are interested in the number of chips and the number of links between chips that are necessary to implement G_n with M, given links with width w. We denote these numbers with $chip(M)$ and $link(M)$, respectively. If links get wide to improve throughput by parallel transmission ($w \geq p/4$), we run into the problem of pin restriction. Then, the most obvious mapping M consists of mapping one network node onto one chip. Our main result can be stated as follows:

Theorem 1 *There exists a mapping M' of network nodes to chips such that*
$chip(M') \leq chip(M)/2$ and
$link(M') \leq link(M)/2$.
The chips remain interconnected as a butterfly network.

If we implement G_n with mapping M' instead of M, then it needs less space and links get shorter. This reduces delays on wires and allows for an increase in speed. Engineering aspects such as cooling and power supply also simplify.

In section 2 we will sketch the design of a network node that implements Ranade's routing algorithm and will discuss the problem of pin count restriction that leads to mapping M. Section 3 shows how a slightly different mapping M' doubles gate utilization of the chips. This implies that we need only half the number of chips and links between chips. We finish with the proof that mapping M' preserves the interconnection structure of the 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 address, and the sorted order is kept during routing.

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

If one input buffer is empty, the packet in the other buffer has to wait until it can be sure that no packet with a smaller address will arrive at the empty input in the future. Otherwise, the order would be destroyed. To avoid unneces-

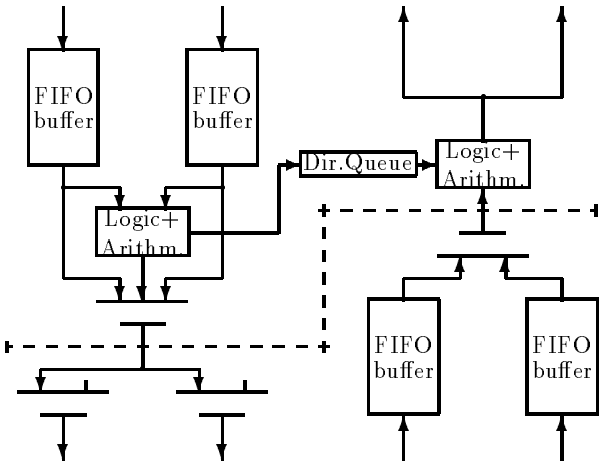


Figure 2: Datapaths of a Network Node

sary 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 has 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 return path 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 guarantees that answers will arrive in the same order as the requests passed, this information can be used to send back the answers [6].

A network node that implements these functions is shown in figure 2. We will ignore the dash line in this section. In practice, each address and data word will consist of 32 bits, and the control information will consist of 8 bits. In order to have a high speed transmission, each node–to–node link should be capable of transmitting one packet and one answer per clock tick. To achieve this, the link must 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. Each link then has a total width of 104 bits. Such a setup causes severe problems due to pin restrictions if one node is implemented on a single chip. As one node has four outgoing links, it needs $4 \cdot 104 = 416$ pins to obtain full speed. Custom chips can be obtained with up to $p = 240$ pins at reasonable prices. This forces sending packets in two pieces, yielding $416/2 = 208$ pins. Compared to pin count however, gate utilization is very low. Thus, most of the silicon on the network chips is wasted.

To increase the gate/pin ratio of the network chips one can implement a subnetwork with $n' > 2$ inputs and outputs instead of one node on one chip. The number of gates grows proportionally to $(n'/2)\log n'$, the number of nodes in the subnetwork (see Def. 1). The number of pins only grows linearly in n'. Therefore, this improves the gate/pin ratio by a factor of $\Theta(\log n')$. The situation for $n' = 4$ and links of width w is shown in table 1. 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 pieces as before.

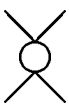
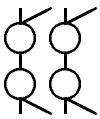
type	t1	t2
outline		
nodes per chip	1	4
number of pins	$4w$	$8w$
gates per chip	g	$4g$
gate/pin ratio	$\frac{g}{4w}$	$\frac{g}{2w}$

Table 1: Mapping nodes on chips

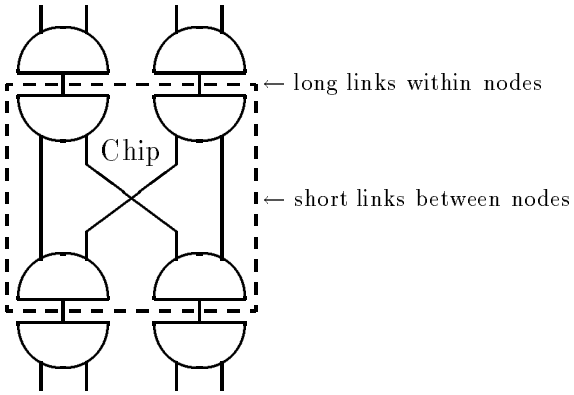


Figure 3: Partitioning of Network Nodes

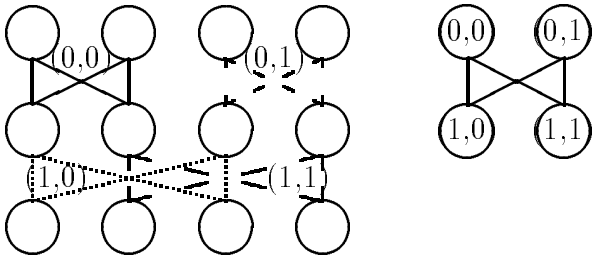


Figure 4: Construction of G'_4 from G_4

3 Different Mapping

Although the node has two inputs and two outputs, it can be cut into two halves such that only one link crosses the cut. This is due to the fact that only one packet is transmitted at a time. The cut is shown as a dash line in figure 2. We now implement a 2×2–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 3 shows this alternative mapping M'. The fact that network nodes can be split into two halves was also observed by Cross [3].

Lemma 1 *If we choose mapping M' as defined above, then*
$chip(M') \leq chip(M)/2$ *and*
$link(M') \leq link(M)/2$.

Proof: 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 per stage in comparison to mapping M. Furthermore, note 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 $chip(M') = (n/2) \cdot \log n$ instead of $chip(M) = n \cdot (\log n + 1)$ chips. Because the new chips have two inputs and outputs just as the original ones, $link(M') = n(\log n - 1)$ compared to $link(M) = 2n(\log n)$. □

We note that we now might have long links within network nodes (between upper and lower part). However, it can be shown that this does not increase node cycle time and thus cannot decrease throughput (see [4]).

Lemma 1 proves the first part of theorem 1. To complete the proof of theorem 1, we show that the chips obtained by mapping M' are interconnected as a butterfly. The mapping M' consists of shrinking each subgraph G_2 of G_n to a node in a graph G'_n. We connect two nodes in G'_n by an edge if the corresponding subgraphs of G_n share a node.

Lemma 2 *If we choose G'_n as defined above, then $G'_n = G_{n/2}$.*

Proof: (by induction on n)
Base: It is obvious that $G'_2 = G_1$ and 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 only have to prove that the ith and the $(i + n/4)$th nodes in this last stage are connected to the ith 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 at how G_n is constructed from $G_{n/4}$ subnets. The ith 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.

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] D. Cross, *VLSI Implementation of the Fluent Routing Chip*, Master's thesis, University of California, Berkeley, 1992.

[4] 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) 202–209.

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

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

[7] 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.