A Note on Implementing Combining Networks[*]

Jörg Keller Thomas Walle
FB 14 Informatik, Universität des Saarlandes
Postfach 15 11 50, 66041 Saarbrücken, Germany

Abstract

In shared-memory multiprocessors, combining networks serve to eliminate hot spots due to concurrent access to the same memory location. Examples are the NYU Ultracomputer, the IBM RP3 and the Fluent Machine. We present a problem that occurs when one tries to implement the Fluent Machine's network nodes with network chips that do not know their position within the network. We formulate the problem mathematically and present two solutions. The first solution requires some additional hardware around nodes that can be put outside network chips. The second solution requires a minor modification of the routing algorithm, but one can prove that there is no performance loss.

Keywords: Computer Architecture, combining networks, butterfly networks

1 Introduction

In machines with emulated shared memory, combining networks serve two purposes. First, they route memory access requests between processors and memory modules. Second, they merge concurrent accesses of several processors to one memory cell into one request and thus reduce hot spots. This kind of access cannot be neglected because it will occur in system parts like synchronization and resource management. Also concurrent access is often used in parallel algorithms for the PRAM model. Hence, combining is crucial for implementing shared memory. Combining networks have been used in several architectures, e.g. the NYU Ultracomputer [3], the IBM RP3 [5], and the Fluent Machine [6].

The Fluent Machine differs from previous approaches in that it is guaranteed that requests for the same cell are merged into one request. However, it is not obvious how to implement

[*]This work was supported by the German Science Foundation (DFG) in SFB 124.

the Fluent Machine's network nodes with *universal network nodes*, i.e. nodes that do not know their position within the network. The use of universal network nodes is advantageous because only one type of node is necessary, which can even be used for designs of different sizes.

We will formulate the problem mathematically and present two solutions. The first does not change the routing algorithm used in the Fluent Machine but requires additional hardware around network nodes. The second solution requires a minor change of the routing algorithm. We prove that the algorithm is still correct and that performance is not affected by this modification.

The remainder of the article is organized as follows. In Section 2 we review Ranade's routing algorithm for the Fluent Machine. In Section 3 we work out the problem that occurs when implementing this algorithm in universal network chips. In Section 4 we present two solutions to that problem. Section 5 contains a discussion.

2 Ranade's Routing Algorithm

Ranade's routing algorithm uses six phases, i.e. six traversals of Butterfly networks to route and combine requests from processors to memory modules and to route and re-duplicate answers back to processors. However, routing only occurs in phases 2 and 5, the other phases can be implemented by dedicated hardware [1]. In Ranade's scheme, each butterfly node contains a processor and a memory module, however this can be changed such that processors (together with dedicated hardware for phases 1 and 6) are only placed at the inputs of phase 2. Memory modules with multiple banks are only placed at the outputs of phase 2. One physical processor simulates a number of Ranade's processors. We call the execution of one instruction of each simulated processor a *processor round*. For details of the processor architecture see [1, 4].

We will focus on phase 2 because combining happens here. Phase 2 is implemented on a butterfly network as given by Def. 1.

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

[1] In the sequel we will use binary representations instead of the numbers itself.

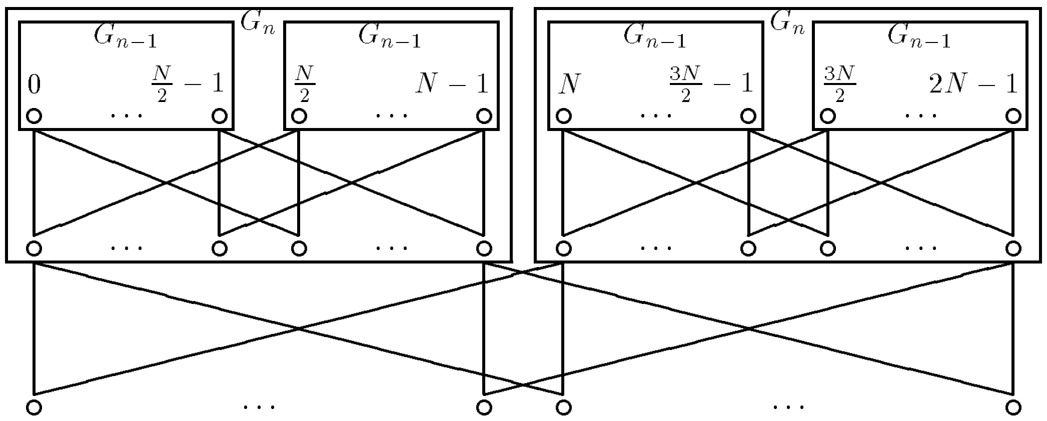

Figure 1: Construction of G_{n+1}

Requests by processors are put into packets, injected at level 0, and delivered to memory modules in level n. Packets consist of a *mode* (READ or WRITE), an *address* and one *data word*. Although a data word is not needed for READ packets, a dummy value is sent to get a unique packet length. Packets of one processor round are injected sorted by their addresses; at the end of the round a packet with special mode *End of Round* (EOR) and address ∞ is injected.

Each network node selects from the two input buffers the packet with the smaller address and thus maintains the sorted order of packets, which can be easily proven by induction. If two packets with identical addresses and modes meet, one is selected, the other is deleted and information is stored to guarantee re-duplication on the way back. The sorting guarantees that all packets of one round with identical addresses meet and get combined.

The packet selected by a node is transmitted to the next level of the network via the appropriate output link of the node (for path selection see Section 3). Only EOR packets are transmitted via both outputs to ensure separation of rounds (address ∞ ensures that an EOR is only selected if both input buffers contain EOR packets).

An empty input buffer prevents a node from sending a packet waiting at the other input buffer. If it would be sent, the sorting could be destroyed by a packet with smaller address arriving later at the empty input buffer. To avoid unnecessary waiting GHOST packets are introduced. If a selected packet is transmitted via an output link a, where $a \in \{0, 1\}$, then a GHOST packet carrying the same address is sent via output link $1 - a$. Hence, GHOSTs serve as lower bounds of future packet addresses along this link. GHOSTs that must wait because they are not selected or blocked by full buffers are destroyed because a new GHOST or a packet will follow the next cycle, so no information is lost.

3 Implementation

The n most significant bits of a packet's address specify the destination module of this packet. Path selection is given by the following Lemma 1

Lemma 1 *A packet with destination module $i_{n-1} \ldots i_0$, that is injected at level 0 of a butterfly network G_n, must be transmitted in level j, where $0 \leq j \leq n-1$, along output i_{n-1-j}.*

Proof (by Induction on n): The case $n = 0$ is obvious. To prove the claim for a butterfly network G_n, where $n \geq 1$, we consider the recursive construction from networks G_{n-1} as given in Fig. 1. The packet will be routed to node $x = i_{n-1} \ldots i_1$ in level $n-1$ in one of the networks G_{n-1}. By the definition of G_n, it will reach node $i_{n-1} \ldots i_0$ in level n from both positions by taking output i_0. ∎

Note that normally destination bits are not taken in reverse order. However, in our case this reversal only leads to a permutation of memory modules which does not affect the correctness but simplifies the implementation as we will see later.

In a direct implementation of the path selection scheme from Lemma 1 each network node must know its level number. To implement the algorithm with universal network nodes, this must be avoided. A solution would be to have the desired routing bit always at the same position in every level. This is possible by the following Lemma 2.

Lemma 2 *If two packets meet in a network node in level i, where $0 \leq i \leq n$, then the i most significant bits of both addresses are identical. We will call these bits* address prefix.

Proof: Consider the subgraph of G_n that contains the two nodes where the packets were injected and the node where they meet. The subgraph is a butterfly network G_i. We apply Lemma 1 with $n = i$, then the two packets are destined for the same output node of a butterfly network G_i and hence their i most significant address bits are identical. ∎

Lemma 2 seems to induce the following implementation: Because the prefixes of two meeting packets are identical, only the *remaining addresses*, which consist of address bits $n-i-1$ to 0, are needed to compare addresses in level i. If the address is shifted left by one position after each level, then the desired routing bit is always bit $n-1$ which allows to use universal network nodes. The address part of a packet contains a shifted version of the remaining address which guarantees correct packet selection within network nodes.

However, this implementation leads to errors as the following Lemma 3 will show.

Lemma 3 *If a GHOST and a packet meet in a node $j_{n-1} \ldots j_0$ in level i, then their address prefixes are different, i.e. comparison of the remaining addresses is not sufficient.*

Proof: For a packet, the prefix is the sequence of routing decisions so far. However, when a GHOST is generated, it is not transmitted via the output that the address would force (see end of Section 2). Hence, the GHOST's address prefix differs in that position from the sequence of routing decisions. If a GHOST and a packet meet, their sequences of routing decisions are identical, and hence their address prefixes must be different. In this case it can happen that the packet is selected before the GHOST, because the packet's remaining address is smaller, although the packet's address is larger than the GHOST's address. ■

4 Two Solutions

4.1 Minor Hardware Modification

One can avoid the error by providing complete addresses to comparator units (see Fig. 2). This however has to be done in such a way that the desired routing bit still has position $n-1$. Both demands together can be fulfilled by inserting the following circuit before the routing and address shifting unit in level i. The address is shifted right by one position, then bit $n-i-2$, i.e. the desired routing bit, is copied to position $n-1$.

Now the desired routing bit always is in position $n-1$ and after the regular left shift we have the complete address. We only have to ensure that that we have one spare bit in the address part of the packets so that no address information is lost during the right shift. This should normally be possible as address parts typically have fixed sizes (32 or 64 bit) and real address spaces are smaller.

The copying of an address bit is an implicit encoding of the level number. Hence we have to take care that this copying is done outside network chips. Then we can use universal network chips, only the network boards differ for different levels. This is considered a minor problem because there are fewer boards than chips and because boards are less expensive than ASICs.

To see how the copying unit can be placed outside a chip we consider the design of a network node as shown in Fig. 2. An obvious mapping of one network node to a chip would result in the copying within the chip. However, if we consider the dashed line in Fig. 2, the number of wires crossing it is not more than the number of wires in one output link. Hence we can use a mapping from [2] as shown in Fig. 3. The resulting chips do not use more pins than chips that implement one network node, and the copying can be put between two chips. One can prove that the network of chips obtained by this mapping still is a butterfly network [2]. Note that this mapping doubles the gate utilization in network chips. However, as network

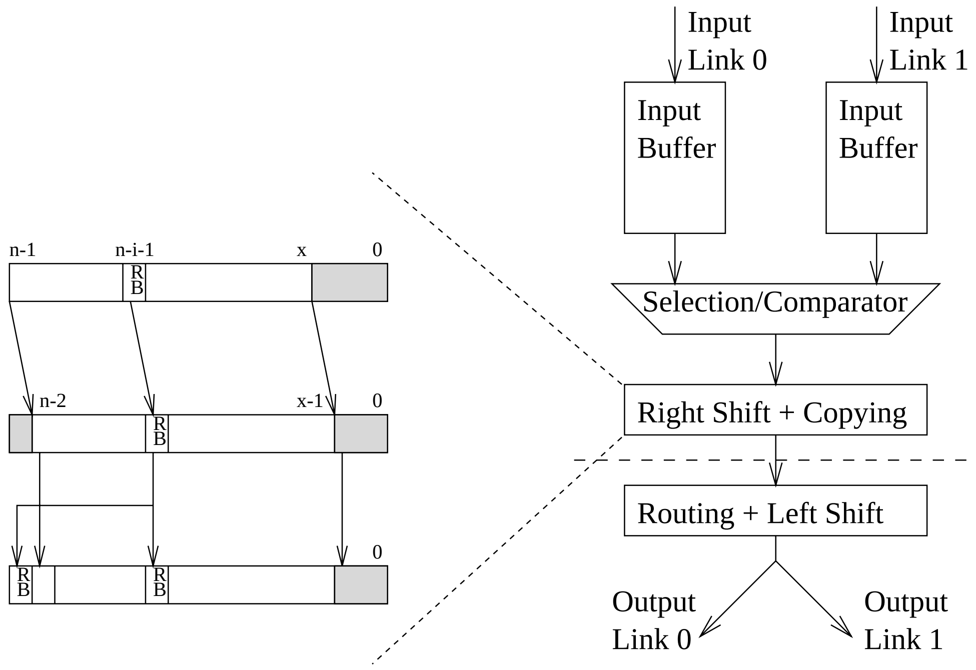

Figure 2: Schematic Design of a Network Node

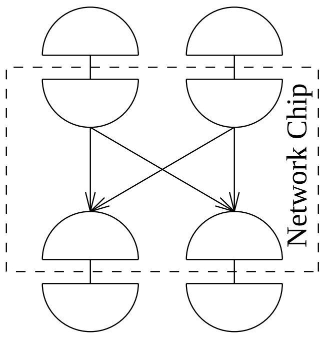

Figure 3: Mapping of Network Nodes to Chips

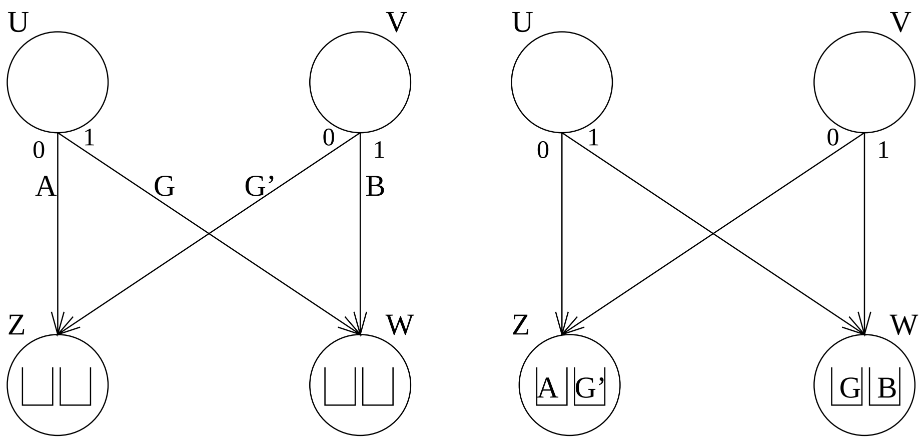

Figure 4: Generation of GHOSTs

chips are pinlimited, this does not impose a problem and even reduces the number of chips by a factor of two [2].

4.2 Minor Algorithm Modification

Consider the situation shown in Fig. 4. Node U sends a packet A to node Z along output link 0. The packet's address then must have the form $a0b$, where $a0$ is the prefix and b is the remaining address. The packet generates a GHOST G with address $a0b$ that is sent along output link 1 to node W. A packet B that meets this GHOST in node W must have entered W along the other input and hence must have address $a1c$. It follows that GHOST G must be selected in node W. Since GHOSTs serve to avoid unnecessary waiting, GHOST G is of no use in node W, because the packet in W must wait no matter whether the GHOST was there or whether the buffer was empty.

Now consider the situation for packet B which is routed along output link 1 from V to W. Packet B generates a GHOST G' with address $a1c$ that is transmitted along output link 0 to node Z where it meets packet A with address $a0b$. It follows that in node Z packet A must always be selected without address comparison.

It follows that if one sends GHOSTs only along output link 0, then the comparison between a packet and a GHOST is independent of the GHOST's address, the packet will always win.

It is obvious that the modified algorithm is correct as long as an empty input buffer prevents nodes from sending a packet that is waiting in the other input buffer. It is also easy to see that performance will not change as GHOSTs that were generated along an output link 1 have a smaller address than the packets that they meet, even if these GHOSTs are further forwarded.

5 Discussion

We presented two solutions to an implementation problem of Ranade's routing algorithm. The first solution has the minor disadvantage that it only allows usage of universal network chips but requires different layouts of boards for different levels. Also if packets are transmitted between chips in several pieces called *flits*, the flits must be treated differently depending on whether they carry an address part or a data part of a packet. This requires additional hardware on boards and enlarges propagation delay between two network chips. The second solution allows to use universal network nodes and boards and requires no additional hardware between network chips.

The second solution suffers from the fact that it can only be applied if routing bits are taken in reversed order as remarked in Section 3. The first solution allows any order of routing bits. Although this does not affect correctness, simulations hint that performance is better if routing bits are taken in normal order. The reason for this is the sorted order of addresses. If bits are taken in reversed order, than in each processor round each network node will first send packets along output link 0, then along output link 1. If routing bits are taken in normal order, then routing decision and sorting are de-coupled, i.e. in one processor round a network node will first send one or several packets along one of the output links, then some packets along the other output link, then some packets along the first output link and so on. This better distribution leads to a better utilization of buffers and hence to a performance improvement. Improvements in simulations have been between 5 and 10 %.

Thus one has a kind of trade-off between performance and universality of design.

Acknowledgements

The authors would like to thank Andreas Paul for bringing up the problem of disturbed sorted order in the network.

References

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

[2] D. Cross, R. Drefenstedt and J. Keller, Reduction of network cost and wiring in Ranade's butterfly routing, *Inform. Process. Lett.* **45** (1993) 63–67.

[3] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph and M. Snir, The NYU ultracomputer — designing an MIMD shared memory parallel computer, *IEEE Trans. Comput.* **C–32(** (1983) 175–189.

[4] J. Keller, W. J. Paul and D. Scheerer, Realization of PRAMs: Processor design, in: *Proc. WDAG '94, 8th Internat. Workshop on Distributed Algorithms* (1994) 17–27.

[5] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton and J. Weiss, The IBM research parallel processor prototype (RP3): Introduction and architecture, in: *Proc. 1985 Internat. Conf. on Parallel Processing* (1985) 764–771.

[6] A. G. Ranade, How to emulate shared memory, *J. Comput. System Sci.* **42** (1991) 307–326.