Replication and Reduction in Multistage Interconnection Networks*

Joseph D. Touch

November 10, 1989

Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389
touch@cis.upenn.edu

ABSTRACT: Multistage interconnection networks are conventionally composed of 2x2 switching elements which perform only permutation functions. A number of networks have been built which provide more powerful switch functions involving replication and reduction of information, including the NYU Ultracomputer and several copy networks. Here we investigate these machines as a group, to see how replication and reduction are interrelated, and what other issues they involve.

---

* This work supported by Bell Communications Research (Morristown, NJ), under the DAWN Project.

1. Introduction

Multistage interconnection networks are composed of series of stages of regularly connectioned switching elements (usually 2x2). In conventional architectures, these networks support only permutation functions in their switch elements, although several notable exceptions have capitalized on other switch functions. These include the reduction capability of the NYU Ultracomputer [GotGK83] and the replication capability of copy networks [Let88] [Tu88] [BuT89].

Here we investigate the two non-permutation functions of reduction and replication as a unified discipline, and consider the interrelationship between the two functions, as well as the common difficulties in their realization. For the purposes of this discussion we consider only hardware implementations, although some comparisons to their software analogues are noted.

The need for non-permutation functions in the switch elements of multistage networks arises from the solutions of several independent phenomena. Reduction networks arise from the need to alleviate 'hotspot' memory contention in partitioned distributed memory systems, while replication networks are being designed primarily to provide multicast capability in packet switched networks. In addition, distributed systems require multicast capabilities to support the distributed data objects which are used to manage the operating system and provide failure resilience [Wa82]. Further, the increased services of packet switched networks will include multicasts for wire services, many-way multicasts for teleconferencing, and support for video lectures, which includes both multicast and reduction requirements [Tu87b]. Although the communication in the networks appears distinct among these two networks, there are fundamental similarities in the methods of information collection and dissemination which unify the approaches.

2. Definitions.

Before discussing the particulars of the networks we consider, it is important to attempt first to unify the disparate terminology of the field, especially since many different areas - including distributed memory, packet networks, and parallel algorithms - are involved in the analysis. Here we consider only MIMD systems, even though many of the multistage interconnection networks were originally designed in SIMD systems [Fl66].

2.1. Architectures

Distributed processing systems are based on variations of Schwartz's **Paracomputer** model [Sc80]. This model assumes a central shared memory, with a number of processing elements (PE's) on its periphery. The central memory supports simultaneous requests by any number of PE's, where write-write conflicts are resolved by simply omitting all but one memory write.

Since Paracomputers are difficult to build, requiring an N-way multiport memory, a more restricted model is used which limits the memory to a fixed, finite number of simultaneous accesses. This is known as the **Ultracomputer**, also due to Schwartz [Sc80]. In order to support a variable number of processors, a set of these memories is used to simulate the single memory of the Paracomputer. This partitions the Paracomputer memory, and requires a communication network between the memory elements and the processors (Figure 1).

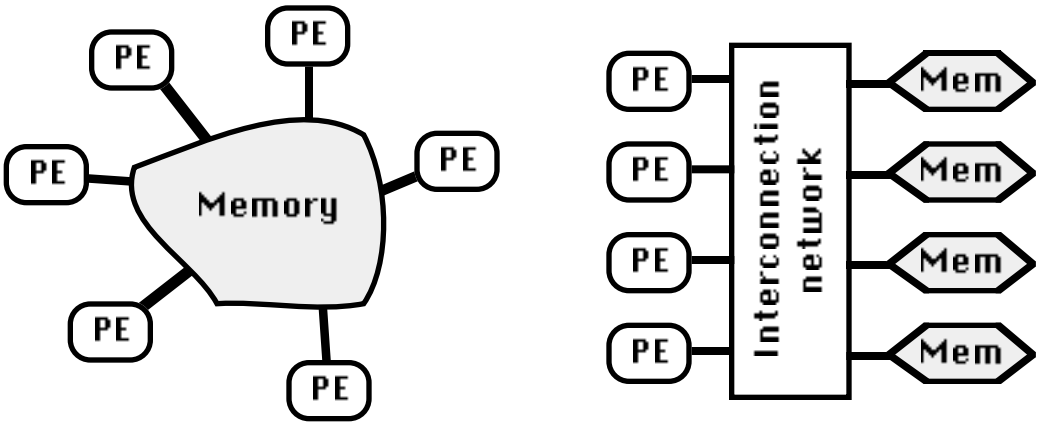


Figure 1: Paracomputer and Ultracomputer models

There are two common ways in which the memory modules, processors, and network can be organized. In **shared memory** systems, the network separates the processors from the memory, forming a bipartite graph. No direct communication between the processors is supported in this topology; all information is conveyed by shared memory locations. The alternate to shared memory is **message-passing**, where each processor is directly connected to a memory module, and the network interconnects all these pairs. Communication between processors is directly supported by the network, and shared data structures are maintained by processor control of access to the structures in its memory module (Figure 2). The number of

memory modules and processors need not be equal in either organization. Shared memory can be considered message passing between memory and processors, so the distinctions are often not obvious. Some message passing systems include the Starlite and its descendants [HuaK84] [LetBA88] [Tu88], the Cosmic Cube, and the BBN Butterfly [Th86] [Mel88], whereas the NYU Ultracomputer [GotGK83] and RP3 [PfBG85] are shared memory systems. The Starlite (and its descendants) are actually developed as packet switched networks, but the principles of packet switching are similar to message passing in MIMD distributed systems.

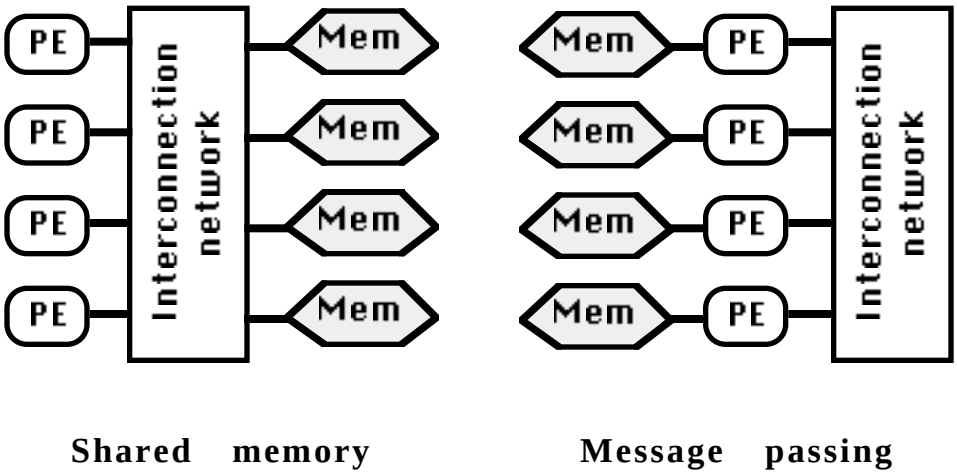


Figure 2: Message passing and shared memory architectures

2.2. Multistage interconnection networks

A **multistage interconnection network** (MIN) is a regularly connected series of columns of switching elements, each affixed to the next by a wired permutation (Figure 3). These switch elements are usually square 2-input, 2-output switch blocks with very simple functionality; the wiring permutations between the stages defines the overall function of the network. The design of these networks provides some of the functions of a completely-connected network (a crosspoint), with less overall switch elements (N log(N) for most MIN's, v.s. N^2 for a crosspoint) [DaT89] [De89] [WuF80].

There are a few canonical organizations of these networks, including recursively factored topologies [Cl53] and self-routing organizations [Be62]. Self-routing organizations include the Omega network, the Staran FLIP network, the binary Benes network, the hypercube, and the **banyan** network (named for an African tree, the intertwined branches of which its permutations resemble). All of these are topologically equivalent and have log(N) stages, each stage being composed of N/2 switch elements. Another important organization is the **Batcher**

sorting network [Ba68], which is composed of the sequential composition of increasingly longer prefixes of a banyan network, resulting in N $\log^2$N complexity.

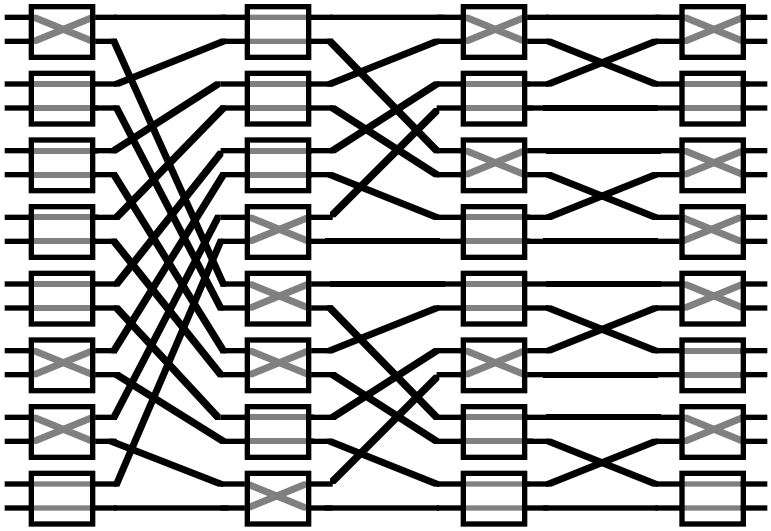


Figure 3: Multistage interconnection network

It is assumed here that these networks are used in a synchronous mode, where at each time-step a new set of packets is presented aligned at the inputs, and these packets proceed through the network in a phase that proceeds through each stage in synchrony. Pipelining of packets, where the next phase of inputs is presented before the previous phase exits the network, is permitted in most of the networks noted here as well.

These topologies have several distinguishing functional characteristics. A network is **internally non-blocking** if, given a permutation of addressed packets at its inputs, it can route all these packets through the network without contention or collision. The notion of blocking originates in telephony [SuB77], and is thus restricted to the case where each input is paired with an desired output, thus only permutations of addresses need be considered.

Contention is the event in a switch element where both input ports request routing to the same output port; this can be resolved by buffering one of the requests within the switch element, known as **link-based internal buffering** [Tu88]. Another contention resolution scheme is to route one of the two packets to the wrong (idle) output port, assuming it will be stored in a **shared internal buffer** at some later point, and **recirculated** through the network via a feedback path, as in the Starlite [HuaK84]. A **collision** occurs when no internal buffering is provided, and one of the two packets is simply lost, as is done in the BBN Butterfly [Mel88]. **Congestion** is the event where contention causes other traffic in a network to exit more slowly

than in the absence of such contention, regardless of whether that traffic participates directly in the contention.

It is important to consider the other effects of buffering, including an increase in the size of the switch elements (in link-based buffering) or the overall network (in shared buffering), and the potential for buffer overflow. Some designs [Tu88] use a reverse wiring of 'buffer-empty' signals to permit an automatic flow-control path, from the overflow back to the source, but most networks simply begin to lose packets when overflow occurs. Buffering also destroys the relative alignment of packets, which can destroy consistency requirements of some multicasts (seen later). This prevents ganging the inputs together, which providing a greater bandwidth via parallel use of input lines [HuaK84]. The **constant internal latency** afforded by unbuffered networks permits such grouping without requiring downstream re-alignment of a set of grouped packets.

Internally non-blocking networks are divided into two cases: completely non-blocking and rearrangeably non-blocking. **Strict non-blocking** implies that after a set of input/output port pairs are in use, a new pair can be routed through the network without breaking any existing connection. A more relaxed constraint requires only that the new pair be connectable, including rearrangement of existing connections; this is known as **rearrangeably non-blocking**. Since the MIN's we consider are packet or message based (as opposed to circuit-based telephony [SuB77]), only rearrangeably non-blocking fabrics need be considered, since new pairs are added to the only when the entire set is resupplied to the net as a result of the next phase of arrival.

Batcher sorting fabrics are rearrangeably non-blocking for any set of inputs (otherwise they would be of little use as sorting fabrics) and thus require no internal buffering to avoid internal collisions, but do not actually route the sorted packets to their destinations. Banyan self-routing fabrics are provably rearrangeably nonblocking only for a set of addresses which is monotonically increasing or decreasing and has no idle inputs interspersed. Unfortunately, no fabric whose switch elements have inputs and outputs of the same bandwidth can be completely collision-free, since the set of addresses provided at the input can have duplicates. These collisions are called **output port collisions** (as distinguished from **internal collisions**, which are avoidable), and must be considered regardless of the interconnection network used.

There are three primary methods for the design of a non-blocking switch fabric. A Batcher sorter can precede the banyan router, resulting in a **Batcher-banyan**, so that the Batcher maintains the monotonicity and gap-free constraints required for the banyan router to be non-

blocking, and the router delivers the sorted packets to their final destinations [Let88] [LetBA88]. In a **buffered banyan**, buffers are included in the switch elements, where packets are stored when contention occurs, and retransmitted when the desired output port is otherwise idle. To further reduce the potential for internal contention in a buffered banyan, a random buffered banyan can precede the router [Tu88], resulting in a **probabilistic buffered banyan**. This random network routes packets according to other deterministic or nondeterministic criteria, the only significance of which is the ignorance of the packet source and destination in the routing decisions. The probabilistic buffered banyan requires only 2logN stages (of components, and delay), whereas the Batcher-banyan requires $N\log^2 N$ stages, although the former requires a more complicated buffered switch element to reduce blocking, and does not guarantee non-blocking for arbitrary input sets.

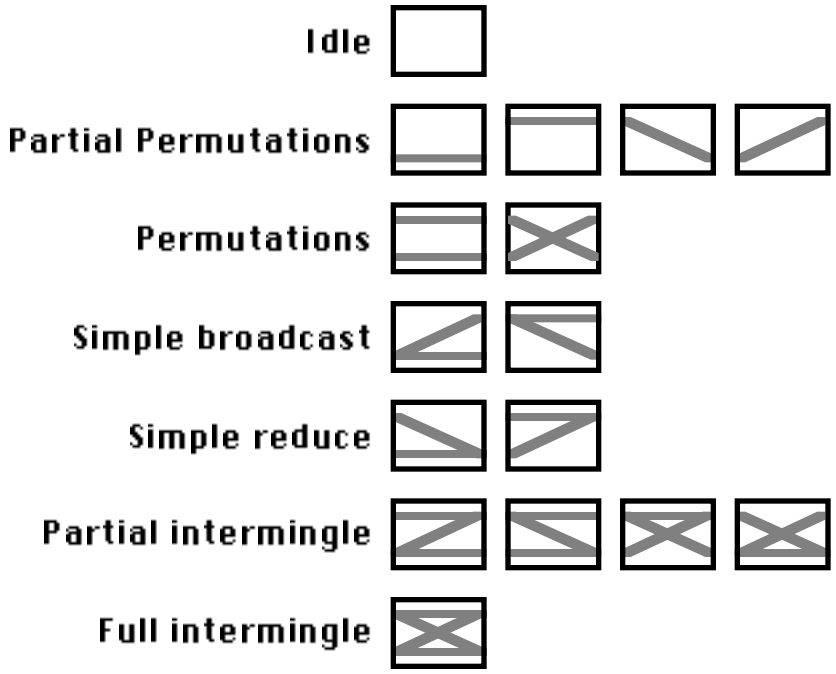


Figure 4: Switch element configurations

The **switch elements** are composed of two input ports and two output ports each, in the simplest case. Normally, only permutation operations are supported in these switches - straight-through wiring, and exchange wiring, of which there are 7 configurations (some have idle inputs) (Figure 4). There are other functions possible in these elements, the remainder of which include fan-out and fan-in of information. If the fanout is restricted to copying the data portion of the packet, it is called a **copy** (also called **broadcast**) [JiP85], of which there are two configurations, upper-copy and lower-copy. The fan-in requires the use of a mapping function to **reduce** (also called a **combine**) the amount of information (there are similarly

two configurations, upper- and lower-reduce), or alternately requires a doubling of the resulting packet length. Other more complicated patterns include the combination of a copy and a reduce [HuaK84], of which there are four configurations (the Starlite uses only the Z and inverse Z configurations [HuaK84]), and the two copies and two reduces resulting in a complete intermix of information in the packets. The lone latter configuration has never been observed in a switch design. Since there are 4 links in a switch element, there are 16 possible configurations. These latter configurations can be called 'interminglings'.

2.3. Multicast

Broadcasting and multicasting are the dissemination of information from a single source to multiple receivers, and are network functions made possible by copying at the switch level. In network **broadcasting**, a single source packet is replicated and copies are delivered to all outputs; at most one broadcast can be supported at a time, since all output ports are utilized by a single broadcast. **Multicasting** is a restricted form of network broadcasting, where only a subset of the outputs receive copies; broadcasting is just a special case of multicasting where the destination set covers the outputs. These two operations support distributed objects [Tu87d] and reduce the amount of memory accesses to retrieve global (common) information [HaMS86].

Note that in each case the MIN can be used to perform the desired operation, provided the replication occurs in a way which distinguishes the copies - completely identical copies would arrive at the same output port, where multicast delivers distinct copies to distinct addresses. So the copy function must (at least) **index** the replicates such that, after all stages of replication, no two replicates share the same index. The term **replicate** can be used to denote these differentiated copies, as distinguished from identical **duplicates**.

Multicasting is commonly implemented in MIN's via a two-phase network composed of a copy stage which replicates the packets and a conventional routing MIN [Tu88] [Let88] (Figure 5). Both overflow and collision is possible in a two phase network, where overflow is not defined for single-stage multicast networks. Also, overflow loss does not necessarily reduce collision loss, as seen here.

Multicasting can emanate from a single source to a set of destinations, or among members of a set to each other, called **m-way multicasting**. The latter can be achieved as a set of simple one-way multicasts, so only those will be considered here. Further, in fault tolerant contexts it is useful to consider the **atomicity** of the multicast, i.e. whether a multicast is completely

fulfilled or completely lost, contrasting a partially fulfilled request. In the serial composition of a copy stage and a routing stage, there are two locations where packet loss can occur, destroying atomicity. Copy requests may **overflow** the replication capacity of the copy stage, and multicast sets may overlap, resulting in conventional output port **collision** in the routing stage (also Figure 5). Note that output port collisions occur due to the nature of a multicast request, and not due to the factorization of the copy process, whereas overflow is a function of the existence of a dedicated copy phase.

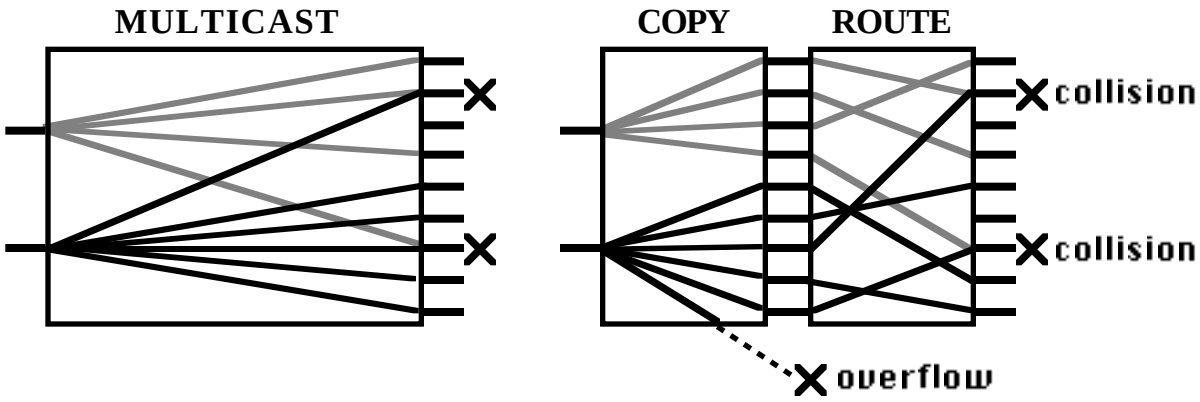


Figure 5: Multicast as serial composition of copy and route

Assuming a multicast is atomic and one-way, there are still other considerations. A multicast may be **serial** or **parallel**, depending on whether the set of copies is made entirely in one phase in the network, or multiple phases are required to complete the set. While most MIN's accommodate parallel multicasting, there are circumstances where the entire set cannot be completed in a single phase; in these cases, it is often useful to **split** the multicast into two or more equivalent subsets, and satisfy the subsets in sequence (Figure 6). It is important to notice that multicasts always satisfy the collision-free property [To87] - the set of destinations is composed of unique addresses; this property can be called **self-consistency**. The self-consistency of a multicast set means that splitting forms two smaller self-consistent sets. For the purposes of optimal switch utility, it is useful to maintain the largest self-consistent set possible, to reduce the probability of collisions.

Multicasts are often accompanied by reduction operations, in order to facilitate multiway group interaction [Tu87b], collect acknowledgements from the multicast receivers [Kat87], or support multicast flow control [Tu88]. These reduction operations can be reduction versions of Chang's echo algorithm [Ch82], or a serial collection of replies [GopJ84], as well as a hardware reduction as defined later.

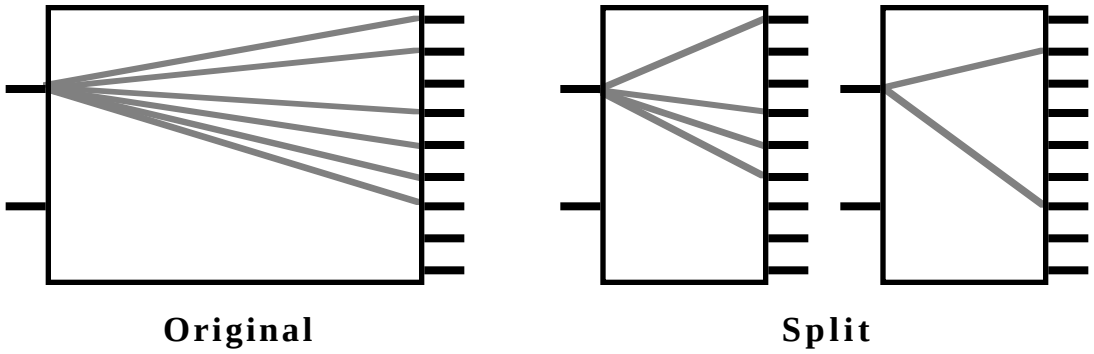


Figure 6: Splitting a multicast request

In addition, some multicast methods provide for easy addition and deletion of members of a multicast set, called dynamic multicasting [Tu88]. Other schemes provide such updating at a much higher cost, or not at all (static multicast).

2.4. Reduction

Reduction is the collection of information from multiple sources for delivery to a single destination, and is supported in a network by the combining operations at the switch elements. Reduction, also known as network combining, is used for reducing hotspot memory contention in the NYU Ultracomputer [GotGK83], distributed management of locks in the RP3 [PfN85], and eliminating serial access to distributed data structures. The distributed processing of central data can also benefit from the use of reduction, in a way which translates serial code into a parallelizable equivalent [Got84]. In this way, limitations in algorithm speedup proscribed by Amdahl's Law* can be circumnavigated.

In order to perform a reduction, a composition function must be defined at the switch level. In the simplest case, the function is a selection - thus omitting one of the two packets, and presenting the other as output. This function is the method of reduction in some networks which approximate the effects of contention resolution via omission [HoE89], and also used to explain the dropping of packets which occurs in unbuffered blocking networks, like the BBN Butterfly [Th86].

---

* Amdahl's Law states that if k% of an algorithm is serial (not parallelizeable), then the optimal acceleration would reduce all parallel code to one time-step, increasing the algorithm speed by a factor of 1/(k%).

Any function which reduces the product of two packets to one output packet can be used in the combining, but reduction is made simpler if the contents of the packets are known (and restricted). If we consider the shared memory organizations, packets normally contain only read and write operations, which are more easily combined. For example, two write operations to the same memory location can be combined by omitting one of the write requests, such that the memory behaves as if the two writes actually occurred in sequence, with no intervening operations. Note that the combining can only occur if the packets affect the same memory location; other pairs of packets, destined for the same output port but with distinct addresses, cannot be combined in this way.

Of the set of restricted operations which facilitate combining, there are two commonly used in shared memory systems. Replace-add operations are requests which replace a memory location with a sum, and return the new value (Figure 7). Fetch-add operations are similar, returning the previous memory value instead. While 'add' is the function describing these operations, any associative operation (op) can be specified [GotK81]. The equivalent result of a replace-op can be computed from the result of a fetch-op, but the reverse (computing fetch-op from the result of a replace-op) is true only for functions with an inverse [GotK81]. For example, replace-min can be computed from fetch-min, but fetch-min cannot be computed from replace-min, since min has no inverse. There are also equivalences for test-and-set primitives and simple load and store operations in terms of either fetch-op or replace-op functions. Some common operations supported in fetch-op systems are *add*, *and*, *or*, *min*, *max*, *store*, *swap*, and *store if zero* [BrMW85] [Mel88]. Combining via these primitives is especially useful in maintaining locks in distributed systems [PfN85]. This also assumes that the memory modules in the shared memory system support these operations with ALUs at each module.

The number of reductions which occur at a single switch element has also been studied. If a switch is unbuffered, only two packets can be combined, and the probability of two packets colliding whose address matches is slim. As a result, combining is usually considered in buffered networks, where the possibility of matching a packet in a storage buffer is high. This same buffering can retard the passage of the combined packet, providing the possibility of combining the result of a combine with another incoming packet, known as n-way combining. It has been shown that for reasonable systems, 3-way combining is sufficient [HoE89], even though many systems support only 2-way combining, such as the NYU and RP3 [GotGK83] [Tz89] [LegKK86].

$$\text{Fetch-op(address,value)} = \text{Replace-op(address,value)}\ \ op^{-1}\ \ \text{value}$$

$$\text{Replace-op(address,value)} = \text{Fetch-op(address,value)}\ \ op\ \ \text{value}$$

$$\text{Store(address,value)} \equiv\ * = \text{Fetch-}\Pi_2\text{(address,value)}$$

$$\text{Load(address)} \equiv\ \text{value} = \text{Fetch-}\Pi_1\text{(address,*)}$$

$$\text{where } \Pi_1(x,y)=x \text{ and } \Pi_2(x,y)=y$$

Figure 7: Definition of fetch-op and replace-op equivalences

Some reductions require no further processing, such as the combination of two stores, where one operation can be omitted. Another simple combine results from a store and a fetch-op. The store continues on to the memory element with a new value ('storevalue op fetchvalue'), and the fetch-op is immediately replied with the value of the store (storevalue), in a 'short-circuit' operation (Figure 8). Each of these two combinations maintains the serializability of memory operations, and reduces the number of packets reaching the memory module.

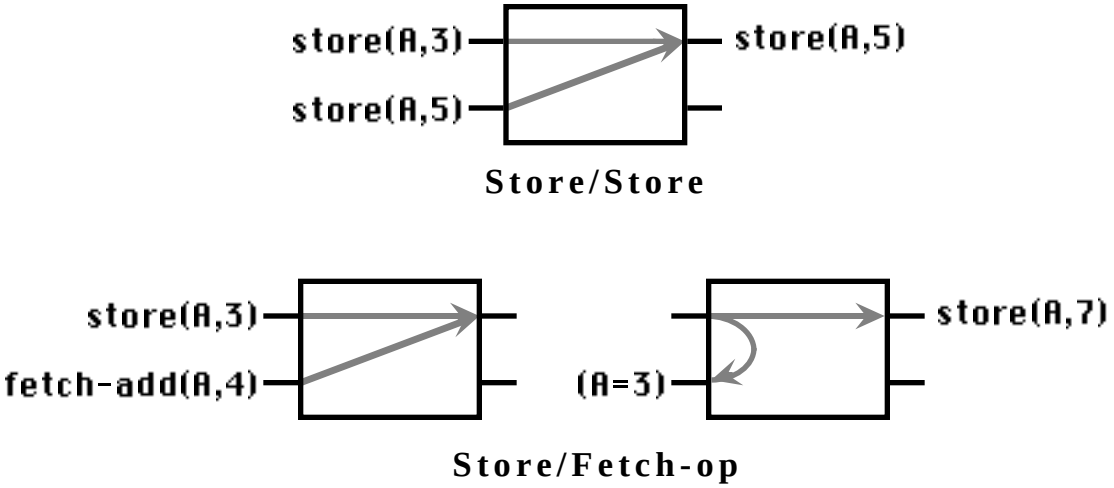
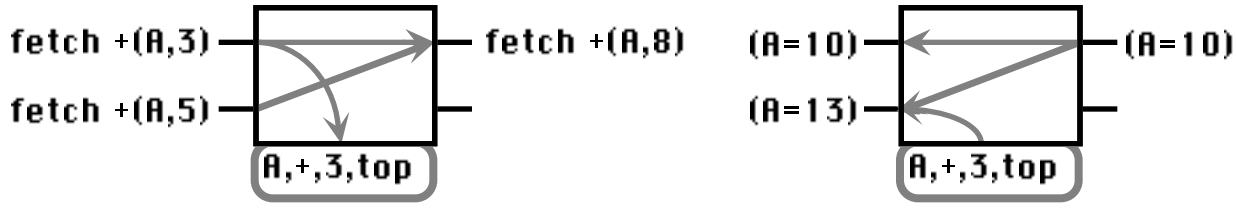


Figure 8: Short circuit reductions

Other operations require intermediate storage in the switch element where the reduction occurs. These include load/load and other combinations of fetch-op/fetch-op pairs (Figure 9). Only one request continues to memory, satisfying the reduction requirement, but a local buffer holds the destination and intermediate information of one of the source packets. When a reply arrives, a reverse multicast satisfies both requests in parallel. In this way, reduction networks often incorporate multicast capabilities in the reverse network paths [PfN85].



forward reduction **reverse broadcast**

Figure 9: Stored-intermediate reductions - 'A' begins as 10, ends as 18

The use of packets which provide memory functions, and are thus easily combinable, results from the use of reduction networks for alleviating memory 'hotspot' contention problems. The contention for a memory cell is noticed by the contention of packets at switch elements in the network, and the reduction alleviates the contention before it reaches the memory. While other packet contents may be reducible in other contexts, none except for the reduction functions equivalent to omission has been suggested in the literature for contents which extend beyond fetch-ops.

In some systems, once a hotspot has been sent a packet, no further requests to any hotspot may occur. This is known as limited access [YewTL87], but also called a 'blocking request'. Blocking requests is necessary in buffered combining networks, since otherwise requests could combine and pass each other in a way which would not be ultimately serializable [EdGK85].

2.5. Consistency

Central to the notion of the ability to reduce or multicast packets in a network is the definition of consistency. Since reduction networks focus on combinations of memory requests, consistency is defined as serializability of the set of reduced operations, as if they would have occurred in some serial order on a single-access memory. Note that this temporal access need never actually have occurred; it is sufficient that the results obtained are equivalent to some serial interleaving of the requesting streams from the source processors.

In multicasting, consistency is related to the possible misordering of a set of multicast packets. Two multicasts should arrive at all destinations in the same order, although there may or may not be permitted some variation in the delay between the two arrivals. A fixed delay is required for many fault tolerant voting schemes, which use time-out to assume the effective loss of communication.

3. Canonical reduction and multicast networks

There have been several implementations of reduction and multicast networks; the NYU Ultracomputer [GotGK83] is the original reduction network, and only two recent networks, by Bell Communications Research (Bellcore) [Let88] and the University of Washington at St. Louis (WashU) [Tu88] incorporate multicast facilities.

3.1. The NYU Ultracomputer reduction network

The NYU Ultracomputer [GotGK83] [Sc80] [Kr82] [GotLR83] [Got84] [EdGK85] [LegKK86] is a shared memory link-buffered network, supporting fetch-op memory access. The original design is based on an implementation of Schwartz's Ultracomputer model [Sc80].

The network combines requests of the fetch-op type. Preliminary Ultracomputer documents support the superiority of fetch-op to replace-op [GotK81], since replace-op is always computable from fetch-op results, while the converse is true only for invertible 'op' functions, but subsequent discussions assume use of replace-op functions [Kr82] [GotLR83], up until the final implementation described in [GotGK83].

The switch elements are composed of a buffer on each output port, and on each input port as well, to service reverse-flow multicast replies. In addition, an internal waiting buffer is provided, to record the combination of fetch-op/fetch-op requests in a way which facilitates reverse multicasting of the reply information (Figures 9,10). Other combinations, such as store/store and fetch-op/store requests are combined without use of the intermediate storage, as previously described. While various simulations study the optimum input and output queue lengths, no study of the wait queue lengths was described.

The output port buffers are implemented as systolic matching queues. Requests are queued as received, and dequeued each cycle to the output port. In addition, as packets snake through the buffer, they are compared against each other, and combined as indicated, where possible (Figure 11). When a combine is specified, the packet in the left column moves into the combine column on the right; the column shifts out with the middle column. If the combine column is not empty, a combine occurs when the two packets empty. Note that the actual combination does not occur until *both* packets exit the buffer, in this design. The combination thus alleviates traffic in subsequent stages, but the queue stores both packets as if no combining occurred. The design of this queue indicates a worst case where the queue is full and combining requests arrive

immediately adjacent in the queue; in this case, combining does not occur until the packets have traversed half the queue (Figure 12).

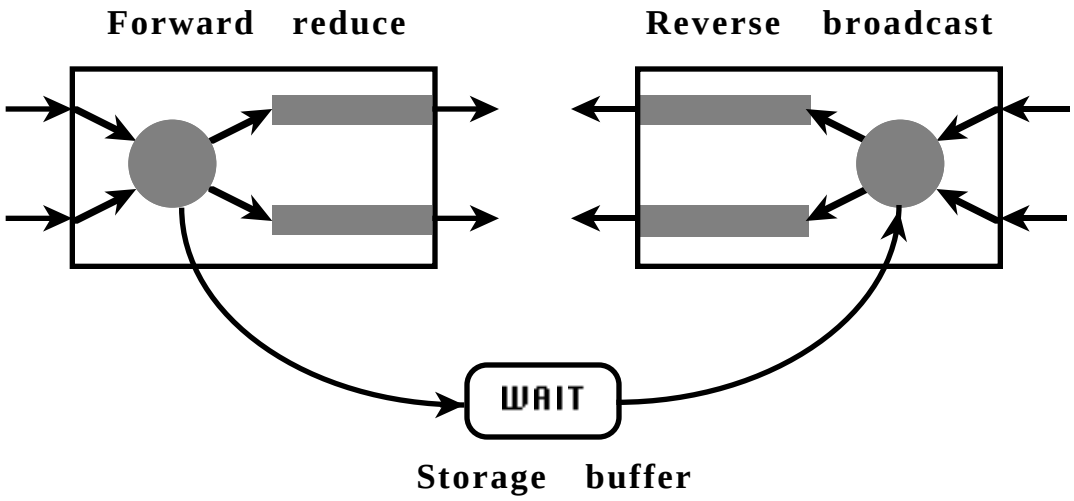


Figure 10: NYU switch element design

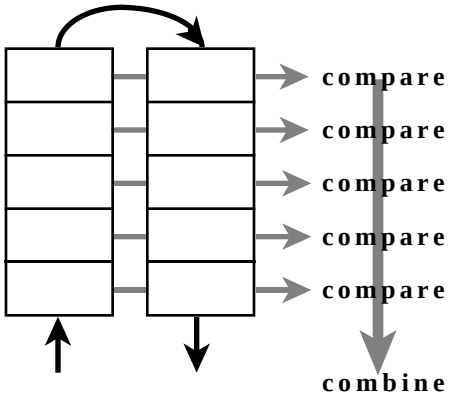


Figure 11: Systolic queue

This design promotes queue growth when packets arrive aligned and the network is loaded, indicating a deficiency which amplifies undesirable behavior. The design suffices when output packets are pipelined on each port, but does not provide the matching required for the wait queue; the organization of the wait queue was never described. This design supports only 2-way combination - once a combine occurs, it blocks subsequent combines (Figure 13).

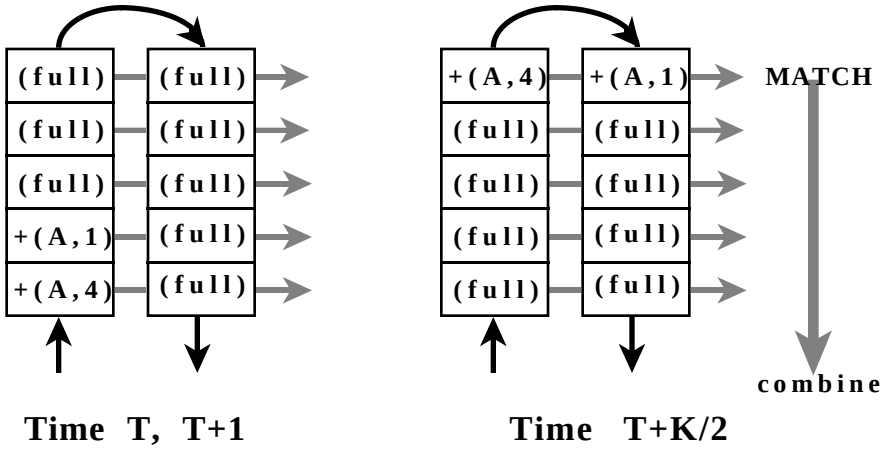


Figure 12: Worst case combining

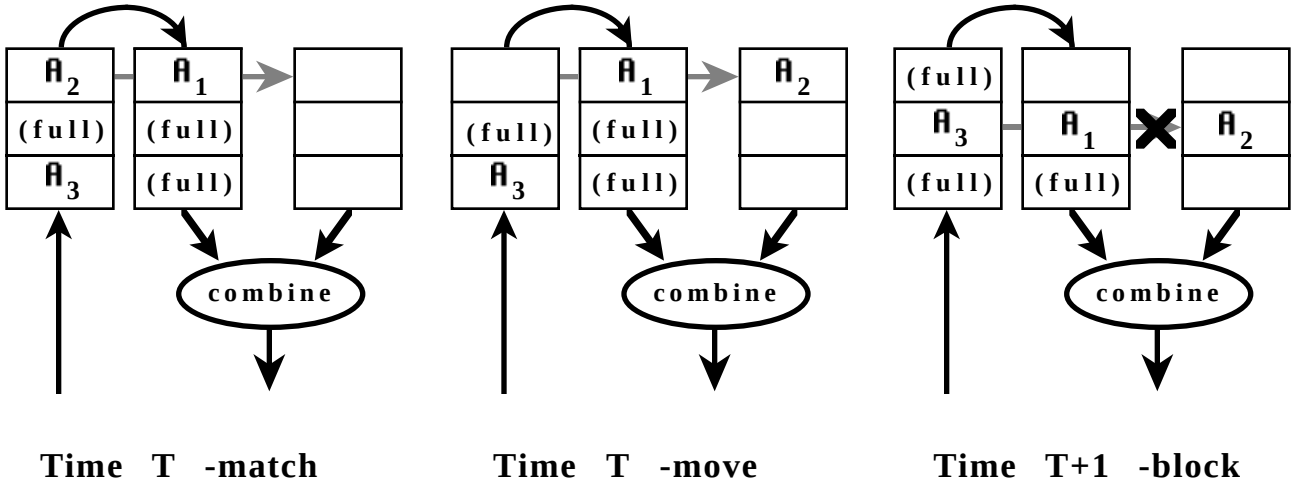


Figure 13: 2-way combine blocking a third attempt

The switch relies on interfaces at the processors and memory modules that support the reduction. In addition to packet assembly/disassembly operations, these interfaces block subsequent requests to a memory location if there is an outstanding request to that location, which is a less restrictive version of the blocking access described before (only accesses to the unsatisfied memory request location are blocked, other memory accesses are uninhibited). The memory interface also provides the arithmetic facilities for supporting the fetch-op primitives.

Later analyses of the design attempt to reduce the complexity of the switch element and provide extended network functions [EdGK85]. Some of the simplifications involve supporting only

combinations that do not require intermediate storage, such as store/store and store/fetch-op combines. Implementing combination only at later stages in the network and restricting combinations to pairs only (2-way combine) were also considered. Finally, a combination of two parallel networks was described, one which combined and one which did not. This design was implemented, for cost and simplicity, in the RP3. In addition, the PE caches were suggested to be lockup-free - non-blocking on requests, such that a cache miss and subsequent memory fetch would not block subsequent PE access to that cache at other locations. There were also analyses which note that 2-way combining is not sufficient for sample problems [LegKK86], and note that 3-way combining would suffice.

Two extensions to the design were proposed [EdGK85]: reflection and refraction (Figure 14). Reflection uses a virtual address in a memory module and an active memory interface to redirect a request from a PE to that module to another PE, supporting automatic forwarding of a request. This implements message passing among PE's as a direct operation, and IPC (interprocess communication) supported by the memory controller. Refraction is a similar redirection to a peripheral processor connected directly to the memory module. With the addition of these two primitives, the distinction between the original shared memory organization and message passing blurs substantially. These automatic forwarding of messages are also known as mailbox operations.

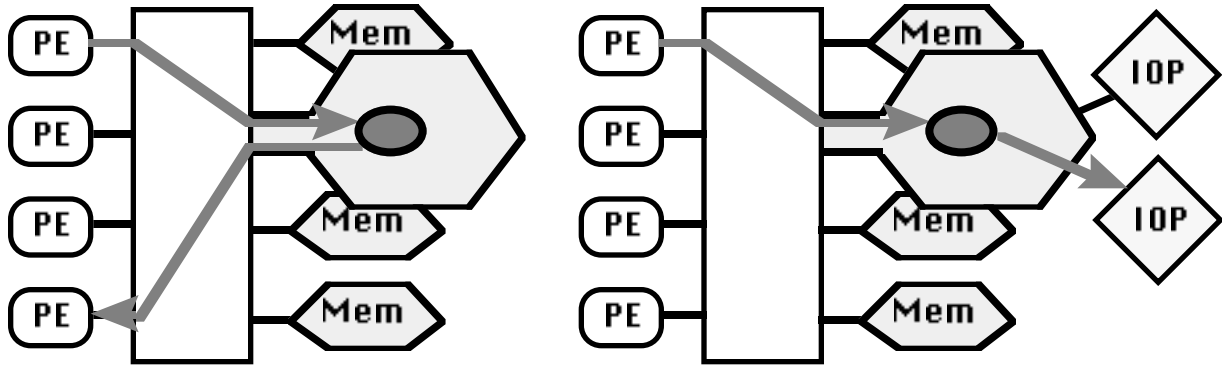


Figure 14: Reflection and refraction

3.2. The Washington University multicast network

The Washington University at St. Louis (WashU) has a multicast network under development [Tu88] [Tu87a,b,c,d] [KhT87] [Ro87] [BuT89] (Figure15). The network supports packet

message deliveries, and is not intended for direct MIMD multiprocessor use, but the principles of the network apply to a message passing paradigm similarly. The network is a two-phase design, using a link-buffered banyan network for copying, and a probabilistic link-buffered banyan network for routing. Between the two networks, the copy indices are used in conjunction with the packet's original channel number to determine a particular destination for that copy, using a translator table (called the BGT, or broadcast and group translator). The multicast sets support dynamic addition and deletion of members, but only by a central call processor, which broadcasts updated set table entries to the BGT's using the network itself, as is also required in the Bellcore switch [Let88]. Also, the use of buffering prohibits utilizing ganging the inputs to increase the link-based throughput of the switch, since the variable internal latency introduced by the buffers destroys alignment of the ganged packets (which it is not trivial to reestablish).

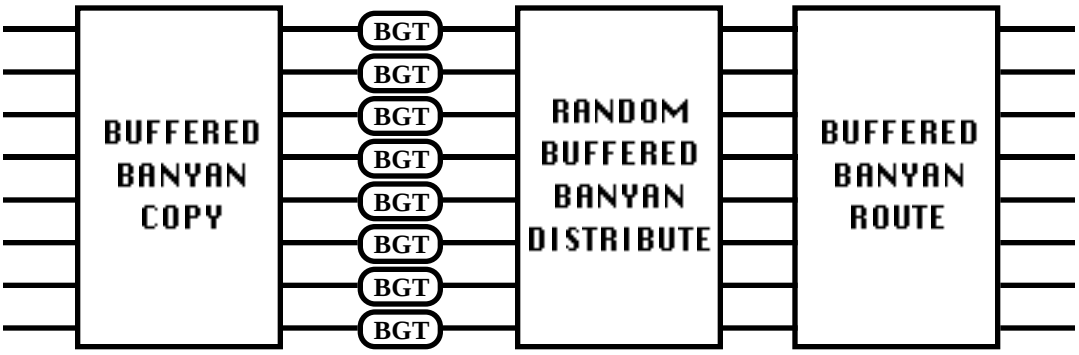


Figure 15: WashU copy network

The network is designed to handle both connection oriented (virtual circuit) and connectionless (datagram) services, but only connections can be multicast, since a central call processor must initiate the translation tables. Datagram traffic is not translated in the BGT's; it is assumed that they are hierarchically routed, and thus do not require redirection provided by these tables. Group translations are also handled by the BGT, permitting trunk grouping, where packets are routed to any one of a set of destinations (a trunk group), all of which are deemed equivalent. In this way, packet loading can be spread across a set of links dynamically, without involving call setup/teardown.

The WashU copy network replicates packets in a buffered banyan network, using the buffering to resolve copy contentions. The replication occurs in a way which delays the copy operations as long as possible in the traversing of the stages, and attempts to create copies in a balanced binary tree. It is observed that copying as late as possible reduces the potential for congestion.

Note that late copying is equivalent to early reduction in the reverse of the network, i.e. reducing as soon as possible. It is more manifest that early reduction reduces contention (as was its initial purpose), and since late copying is conversely equivalent, the same congestion avoidance should also be true. Unbalanced replication, where descendants of a multicast packet request unequal numbers of replicates, was later considered a way to reduce the (high) potential for contention in the later stages of the copy network (Figure 16).

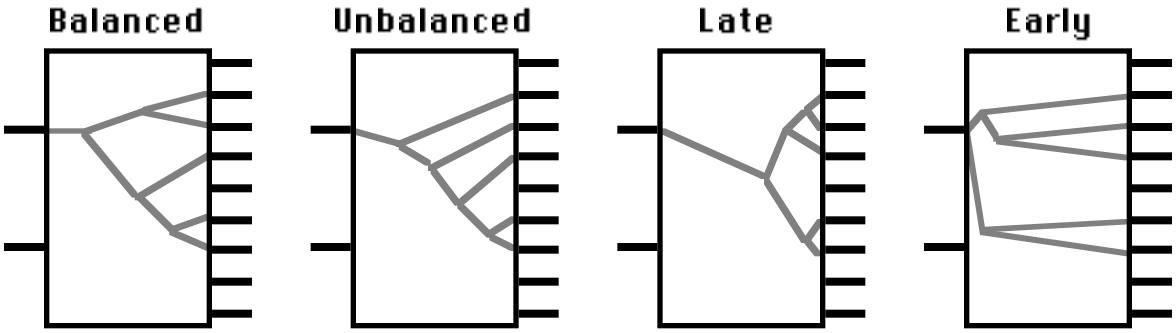


Figure 16: Multicast variations

The determinism of the copy algorithm causes replicates of a multicast set to be produced along a fixed path, determined by the number of copies requested (fanout), and the ID number of the group (which is used to select evenly distributed routing when no copy is indicated in a stage). As a result, the BGT's do not need to maintain individually an entire translation table for each multicast set; given a group ID, fanout, and output port (each of which has a unique BGT), the copy index is completely determined.

Computation of the copy index can be performed at the BGT processors, based on the group ID number, the number of copies in the multicast set, and the number of the output port where the BGT is located (since this completely determines the copy index). The algorithm outlined which computes this index actually performs a software retracing of the multicast operation in the tree, costing logN steps to calculate. While this is expensive, the calculation is unnecessary, since the copy index would be replaced by the destination address for that index, which is determined from the ID, fanout, and output port anyway. Other networks that do not pin the copy destinations compute the index directly as copies are made [Let88].

The total number of BGT entries is the same as the total number of multicast set members, since each BGT has at most one translation entry for each group. This is in comparison to copy methods which do not determine output ports uniquely from a single packet, such as the Bellcore switch [Let88], where each BGT must maintain a copy of the entire translation table for each

multicast set. This same determinism which simplifies the BGT (and making it much smaller and independent of the size of a multicast group), causes multicasts to collide in the copy network [Ro87]. This contention is resolved by link-based buffering, as described before.

The WashU group suggests that the use of a probabilistic link-buffered network is superior to that of a Batcher-banyan unbuffered network, since the Batcher sort involves “higher topological complexity” [Tu87b]. The connectivity of a Batcher, however, is the same as that of a banyan, requiring more stages of the same topology [HuaK84]. The switch elements in an unbuffered network are very simple, but the WashU elements require an additional 240 Kilobits of storage per element, so the complexity gain by using buffering is not convincingly demonstrated. Further, the use of a probabilistic switch to reduce contention also can cause loss of packet sequence information. While [Tu87b] shows that priority fields in the packet headers can help reestablish order, this same order can cause the queue designs to become very complex, since retrieval or insertion would be a function of these priority values.

Analyses of the multicast network performed by the WashU group was not conclusive. While some results consider loading of the network when multicasts are broadcast only, there was no consideration of overflow in the copy network as distinct from output port collisions arising from overlap of the multicast sets [Tu87c]. Further, the simulations did not cascade the copy and routing stages, and so never measured the loss of atomicity and alignment caused by the internal buffering and loss [BuT89].

There were suggestions for extensions and alterations to the original design, which include unequal fanout splitting at a switch element (contraindicated by [Let88]), use of splitting to accommodate sequential copying (contraindicated by [To87]), and giving multicast packets higher priority than other traffic.

Other multicast management considerations are outlined, including authorization to add/remove group members, connection management, routing, security, and congestion [Tu87b]. The congestion control mechanisms rely on a several levels of management, including connection refusal, feedback, and a packet priority field. Feedback utilizes an upstream reduction based on the wiring of buffer overflow indicators to the output enable of the previous stage. While this performs a type of information reduction, no packets are actually reduced, although this does emphasize the need for such a capability in multicast networks.

3.3. The Bell Communications Research multicast network

The Bell Communications Research (Bellcore) multicast switch is composed of a two-phase unbuffered banyan copy network followed by an unbuffered Batcher-banyan routing network [Let88] [HuiA87] [LetBA88] (Figure 17). This switch, like the WashU switch, is a packet network, and is not intended for direct MIMD multiprocessor systems, although the principles are similar. Preceding the copy network is an organization which computes offsets such that no contention occurs in the copy stages., and between the two phases are translation tables (TNT, trunk number translators), similar to those in the WashU design.

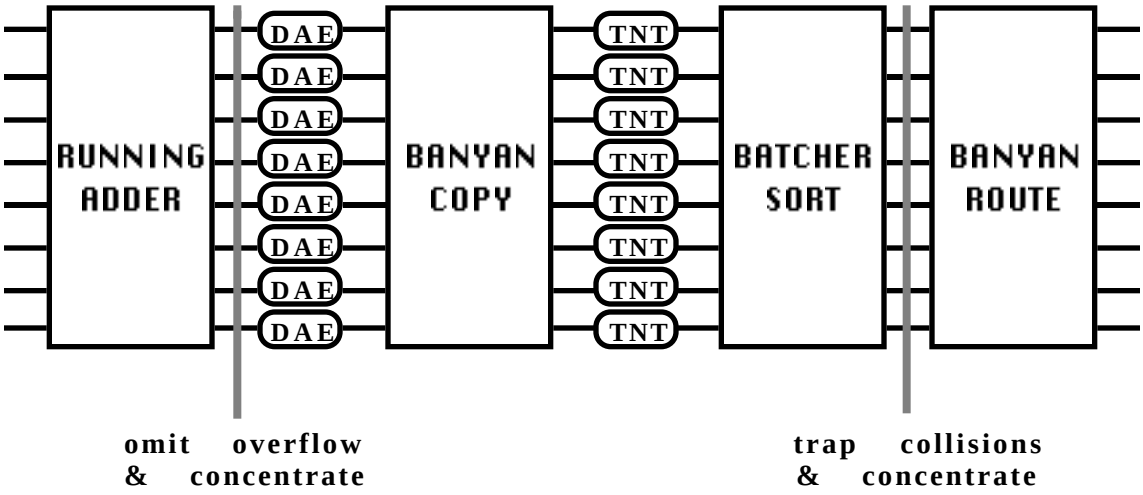


Figure 17: Bellcore switch design

The Batcher-banyan in the Bellcore switch is the basis for the routing operation, but a more complex set of operations ensures collision avoidance. The sequence sorts the packets by destination; if a packet has the same destination as the packet above it (a local computation), then that packet is tagged as a collision. These collisions can be filtered out by a trap network, and routed for recirculation, as in the Starlite [HuaK84]. The remaining packets (which thus cannot collide at the output ports) can be routed with a banyan network, also as in the Starlite. The output of the RAN can also be trapped, to prevent overflow in the copy network, by preempting packets which cause the overflow, via a similar trap mechanism.

In this switch design, the destination of each index of a copy set depends on the other requests to the copy network during that phase. If the first packet requests 5 copies, and the second packet requests 7, then the third packet of the second set appears at output #8, whereas if the first packet requested 2 copies, that same replicate would appear at output #5. As a result, each TNT

must maintain the entire table of index/destination entries for each multicast set, as opposed to the pinned copying performed by the WashU design [Tu88], which requires only one entry per BGT per multicast set (not the whole table). While the TNT tables are thus larger than in the WashU design, Bellcore's copy network is nonblocking and unbuffered; as a result, input port ganging [HuaK84] is possible in this network.

In order to perform the copying without contention, each multicast request is translated to a request for a contiguous span of output ports by the Running Adder Network (RAN) and Dummy Address Encoder. The RAN performs a partial sum on the list of fanouts, and the DAE translates this list into pairs of output port addresses corresponding to the span indicated (Figure 18). While the RAN proposed in [Let88] uses N-ary electrical fanout to compute the sums in logN time, a binary fanout cascaded sum network can compute the same partial sum in logN time as well [To87] [HocJ86]. The RAN in the diagram shows the cascaded sum.

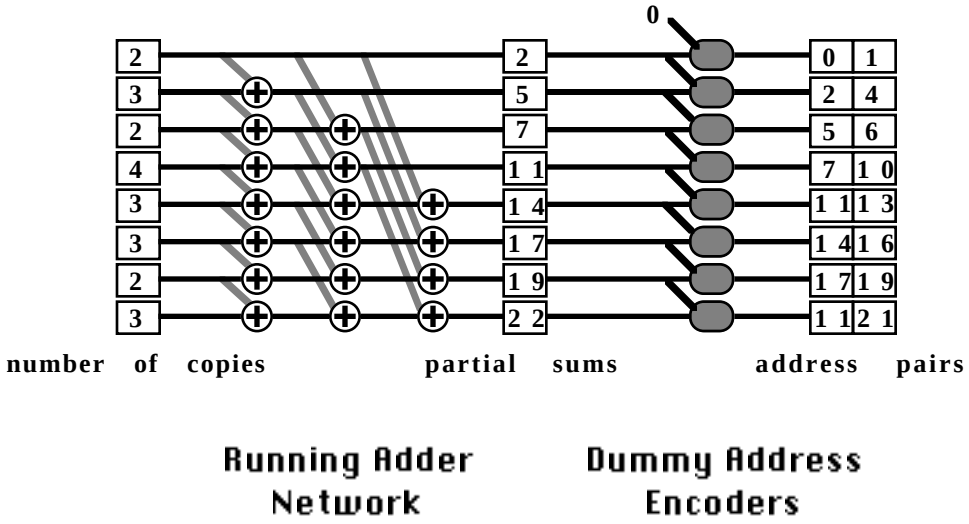


Figure 18: RAN & DAE

The address interval computed by the RAN/DAE sequence is used in the copy network to replicate packets and split multicast requests into halves as appropriate to avoid contention [Let88]. This algorithm is named the 'Boolean Interval Splitting Algorithm' therein, and was first described in SIMD multicasting systems [NaS81]. The complete sequence of operations in the Bellcore network is similar to that performed in software in this SIMD network, as will be described later.

The unfairness of a single, top-down RAN has also been noticed, since copy network overflow avoidance causes requests at the bottom of the network to be omitted, while top-end requests

always succeed (Figure 19). A proposed solution [Let88] was to use alternating top-down and bottom-up RANs, since the algorithm is topologically equivalent in either direction. It was assumed that this alternation would exactly compensate for the top-down unfairness of a single RAN; it was later shown that the combination is not fair [To87], since the unfairness is not linear, and thus combining the two does not exactly cancel the effect. In fact, the dual alternating RAN tends to omit packets just inside from the top and bottom of the network. A proposal which ensures fairness in all cases with only one, top-down RAN uses a random Batcher network to to scramble the order of the incoming packets [To87].

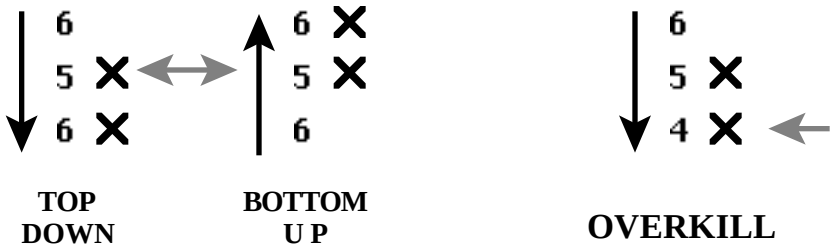


Figure 19: Unfairness in dual RAN, loss of optimality (limit = 10)

Another difficulty in the use of the RAN/DAE involves bit order. Self-routing networks (such as the Batcher-banyan and copy banyan) require the most significant bit of addresses to occur first, while the arithmetic operations (add/subtract) performed in the RAN/DAE require least-significant-bit-first ordering. This imposes an additional delay proportional to the size of the address, to permit addition and routing with the same addresses.

The Bellcore group has also considered partitioning the copy network, for a simpler design (Figure 20). This results in less complex copy networks, and reduces overflow interference from multicast requests outside the partition (a desirable isolation), but increases overflow interference within the partition (no more than fairness would impose in terms of loss, for non-pathological cases). Since the total set of copies are delivered by the same full-size routing network, no additional collisions are likely. Further, since a multicast request coming into a partition cannot cause copies to exit outside the partition, the TNT's need not have tables for multicasts not in their partition, thus simplifying their design.

Other extensions and modifications of the network have been proposed, which include the use of two distinct classes of traffic (reserved, connection / unreserved, connectionless), where there are separate copy networks for each traffic type [LetBA88]. Since connection-oriented traffic avoids copy network overflow at call setup time, its network can be simpler and faster, and can avoid overflow loss completely. Unreserved (connectionless) unicast traffic (point-to-point,

conventional traffic, which can be considered a ‘multicast’ to a single address) can circumnavigate the copy network, thus reducing its complexity as well. This two-level scheme was also investigated in the RP3, with a difference that the RP3 has one copy and one multicast network.

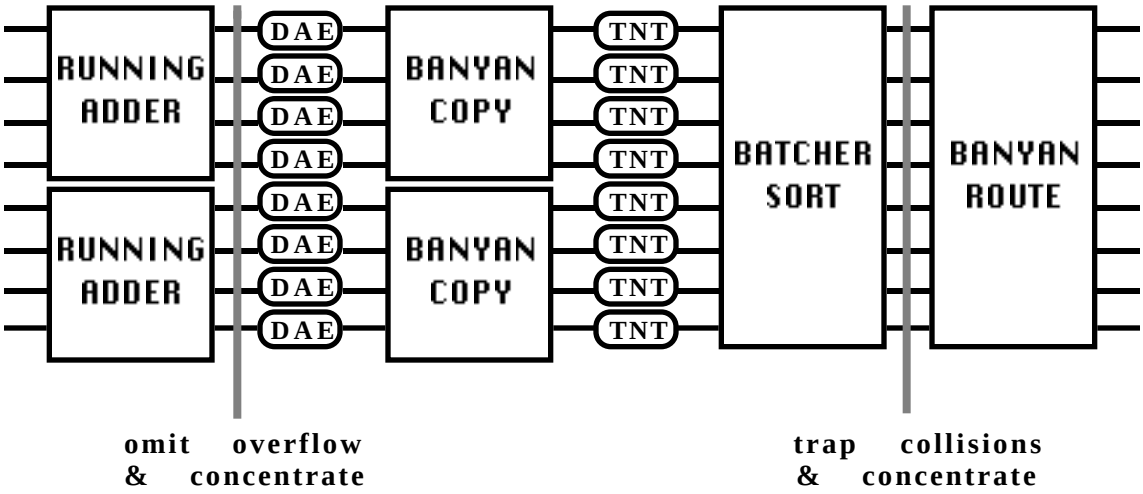


Figure 20: Partitioned copy networks

The WashU group noted that in the Bellcore switch, the overflow problem is similar to binpacking (with the same NP-complexity), since the copying is floating (not pinned to a location) [Tu87d]. This analysis considers only overflow, however; multicast scheduling is a more complex problem, since output port contention must also be considered. Some distributed resource allocation algorithms may indicate whether a method for scheduling multicasts without central call-processor scheduling is possible [To87], such as the Dining Philosopher’s Problem solution [ChM84].

4. Related areas

There are several areas of research whose results relate to the study of reduction and multicast networks presented here. In addition to the obvious examples of other implementations or reduction networks, most notably the RP3, other networks have been designed supporting copying and reduction without explicit multistage combine/broadcast functions, such as the Knockout switch, and the BBN Butterfly. Two other areas, of hotspots in shared memory systems, and software multicasting, also lend insight to the problems studied here.

4.1. Hotspots

Reduction networks originated in response to the hotspot memory contention problem. Once a Paracomputer memory is partitioned into modules, there is a likelihood that some modules will be accessed disproportionately. Memory locations used for distributed systems locks, buffer and network management, and interprocess communication are accessed in this way; such memory locations are known as **hotspots**.

Hotspots can be global or more focused, where global hotspots are the result of all PE's contributing to the hotspot, and focused hotspots are accessed only by a (small) subset of the PE's. Most simulation studies assume global hotspots [PfN85] [KuP86], but focused hotspots are more common in practice [Ler85].

Hotspots can be reduced by two common means; the primary involves use of reduction networks. Another scheme uses copies of the data which are moved out into the network, to substitute for the hotspot. This is used for synchronization, where counters are decremented by a fixed, static set of PE's. In this case, a central counter of 10 can be split into 5 counters of 2 each. When each count reaches zero,. it signals the central count, and decrements it by 2 [YewTL87].

Combining fails when contention is the result of a 'hot' or popular memory module, not arising from a single memory element in common [Ler85]. There are also other kinds of hotspots, such as single source hotspot arising from file transfers from one PE to another PE, and focused hotspots arising from locks shared among a small group of collaborating PE's. The simulations of reduction networks studied did not indicate consideration of these variations, or the implications thereof on the results obtained.

4.2. Software multicast

Software support of multicasting is directly related to the design of multicast MIN networks, yet the current implementations at WashU [Tu88] and Bellcore [Let88] did not include any references to this area. Broadcasting is a facility assumed in most distributed, fault tolerant operating system designs, and many such studies directly address the issue [Wa82]. Examples of multicasting algorithms have been developed for SIMD architectures as well [NaS81]. Some of these are related to Bellcore's switch, such as [NaS81], while most others use a variation of Chang's 'echo algorithm' to distribute replicates [Ch82] [Kat87]. Software reduction has also been studied, in its ability to move virtual copies of a memory location out into the network [YewTL87].

Most software multicast algorithms are based on the use of a spanning tree to send packet copies to distinct locations in a system, without overlap of effort [Wa82]. This spanning tree is effectively computed in the broadcast banyan networks used in both the WashU and Bellcore copy networks [Tu88] [Let88]. The analysis of the overlap of these spanning trees for simultaneous multicasts and broadcasts [Wa82] is similar to the Boolean Interval Splitting Algorithm of the Bellcore switch [Let88].

In the case of software SIMD multicasting algorithms, there is correlation between the algorithm [NaS81] and the current hardware implementation of the Bellcore copy network [Let88] (Figure 21). Even the Boolean Interval Splitting Algorithm is effectively described in [NaS81], as well as the sort-tag-trap-route sequence of the Starlite network, which is also included in the Bellcore switch. In fact, the observed complexity of the algorithm is identical to the hardware complexity of the Bellcore switch. This was also the earliest reference which notes that a banyan network performs a distribution routing on a set of sorted packets, an observation first implemented in the Starlite [HuaK84].

Some algorithms also use variations of Chang's echo algorithm to distribute packet replicates [Ch82]. The echo algorithm was originally proposed as a way to provide software checkpointing in a distributed system, where packets are broadcast in a breadth-first tree through the network. The parent is acknowledged when all children have returned acknowledgements, a kind of broadcast/reduction algorithm on acknowledgements, such that broadcast atomicity is ensured.

Other methods of multicasting are based on serial addressing and emission of packets at the source. Many assume multipoint connections, such as the Knockout [EnHY88], or an Ethernet or token ring global-read bus [GopJ84]. A serial collection of acknowledgements, akin to the echo of the echo algorithm above, provides atomicity of the broadcast.

Software algorithms for reduction have confirmed results from reduction network analyses. Among these, the need for at least 3-way combination at the switch element and the use of discarding as reduction have been studied in software [HoE89]. Some of these techniques do not implement a software multicast to distribute information from replies in reverse, as the hardware reduction schemes tend to. This research also verified the distinction between blocking hotspot requests (which inhibit subsequent hotspot requests), and nonblocking requests, calling them 'limited' and 'unlimited'.

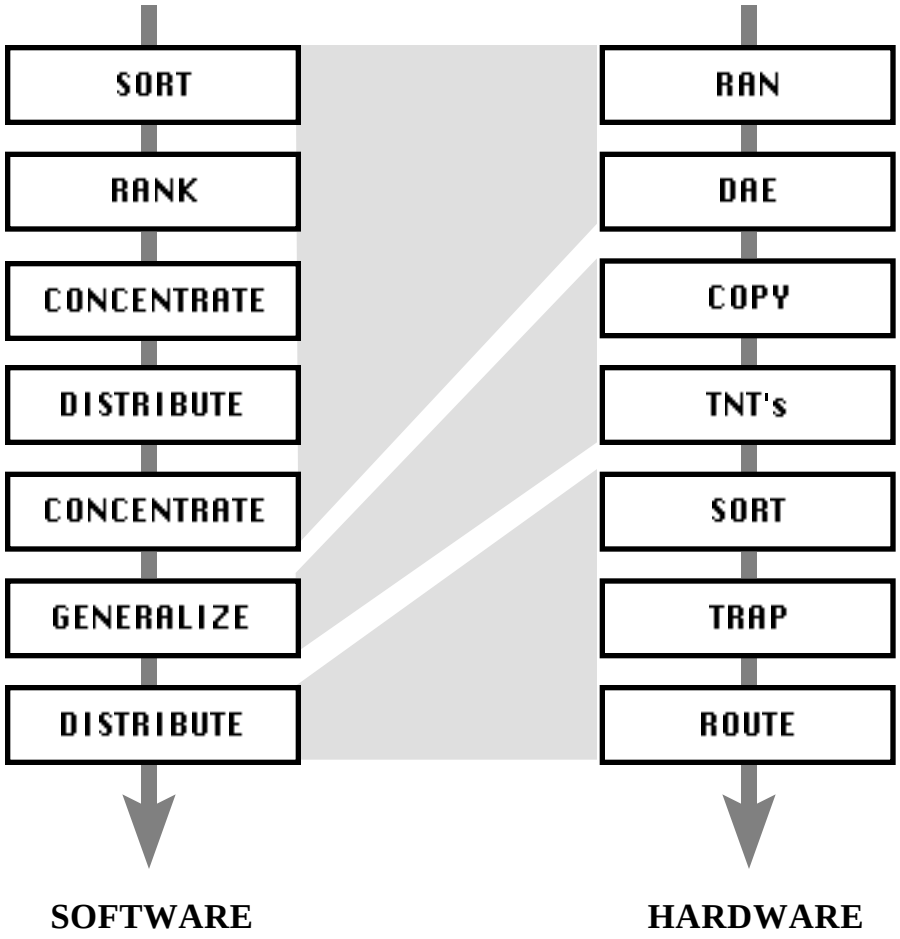


Figure 21: Software v.s. hardware multicast

One notable exception is [YewTL87], which notes that hotspots can be reduced in cases of synchronization variables by moving copies of the variable out into the network, and distributing the access through the virtual copies. The networks used therein were message passing, where PE's were considered processors at the nodes of a virtual shared memory system, facilitating the software support of the virtual copies of shared variables.

In the distributed programming of multiple processors, a countdown variable is often used to implement a join synchronization. Instead of counting from, i.e., 10 down to 0, we can have 5 copies of variables, each initially set to 2. Each process accesses a particular copy, such that no variable is accessed by more than two processors, reducing memory contention. When a variable reaches zero, a signal is sent to the main synchronization variable. This moves the data value out into the network in a tree fashion, but requires prior knowledge of the distribution of these values, and must indicate to each processor which variable copy to access. It is not clear that this partitioning is valid for any other type of data sharing than

synchronization of a fixed number of non-relocating processes, which is a severely restricted case. Note that the intermediate shared variables are in addition to the original shared variable, where each copy is shared by 2 processes, and the original value is shared by the 5 virtual values. This may increase overall hotspot contention in the network by increasing the number of hot locations, depending on the factoring of the value into 'virtuals'.

4.3. Machines

There are several other machines which analyze reduction and replication in processor networks. These include the BBN Butterfly, and the IBM RP3 reduction network, the Starlite which supports copying [HuaK84], and the Knockout [YehHA87] [EnHY88].

4.3.1. The BBN Butterfly

The BBN Butterfly is an unbuffered shared memory network [Th86] [HoE89] [Mel88]. While there is an extension of the Butterfly, known as the Monarch, which is suspected of having multicast capabilities, information on that system was not available in time for this report. The Butterfly has been used to verify studies which analyze reduction networks [Th86], and its internal omission of contending packets has been studied in terms of reduction operations.

One study disproves the phenomenon of tree saturation [PfN85]. Tree saturation was identified as a problem in blocking networks, and shown not to occur in the Butterfly [Th86]. Tree saturation, however, occurs only in link-buffered blocking networks, and the Butterfly is unbuffered, so it is not surprising that the effect was not observed.

The Chrysalis operating system of the Butterfly [Mel88] supports and assumes atomic fetch-op primitives, so extending the network to support reduction operations is also possible.

The omission which occurs due to the unbuffered blocking network design was also studied in terms of its equivalence to reduction. Two requests arrive at a switch, and they contend for an output port. In conventional shared memory reduction, these requests are combined only if they address the same memory location, even thought they contend whenever they are destined for the same memory module (even if to different addresses in that module). In the studies which address reduction as omission, packets are lost only when the specific memory locations coincide; in the Butterfly network, packets are dropped if addressed to the same memory module, regardless of location. So the comparison of these techniques is not valid, although several attempts have been made [Th86] [HoE89] [DiK89].

4.3.2. The IBM RP3 reduction network

The IBM RP3 is a shared memory reduction network, similar to the NYU Ultracomputer in its switch element design, but with special emphasis on the use of caches [PfN85] [PfBG85] [BrMW85] [DiK89].

The RP3 attempts to justify the use of combining in an unique way, with respect to hotspots. They claim that previous studies have assumed hotspots, and attempted to reduce their effect on the processors contributing to the hotspot contention. Here they notice a separate effect, called 'tree saturation', where the presence of hotspot contention in a network severely degrades the performance of the network for all traffic, not just hotspot access. It is for this reason, they argue, that hotspots must be avoided; in the case where a processor is involved directly in the hotspot traffic, that processor 'gets what it deserves'.

Tree saturation is (incorrectly) noticed as an effect of hotspot memory contention which requires a blocking multistage network with distributed routing, and [hotspots]" [PfN85]. This definition spurred the study of counter-examples [Th86], which noticed that the effect occurs only in internally blocking link-buffered networks. It is caused by a successive backup of queues to a hot memory module (Figure 22). It should be noted that tree saturation is an effect of head-of-line queue blocking and a linear composition of queues, in any topology. Once the last queue in a line is blocked and saturated, all queues feeding into that queue (and nowhere else) must also block and saturate, ultimately back to the primary sources of the traffic.

The study which defines tree saturation makes other assumptions which were later called into question. These include an analysis of hotspot traffic percentages as if all processors contribute equally to the hotspot. In real traffic patterns, more local grouping of shared hotspots is evident [Ler85]. They do notice that combining is efficient for lock management, since reduction functions on lock access operations are easily defined [PfN85].

Another discussion on tree saturation claims that buffer management techniques can effectively eliminate the saturation effect [DiK89]. There are two possible solutions - combine, or reject one of the requests. This notices that tree saturation is fundamentally a head-of-line queueing problem, and may be avoided by software queueing restrictions. When a buffer nears its limit, incoming requests to a hot location are omitted, to prevent buffer backup.

The RP3 is limited to a single combine per switch element; it is not known whether this refers to the NYU switch elements ability to combine only one pair per cycle, or the more evident

definition that multiple combines are prohibited by the hardware design, regardless of buffer length.

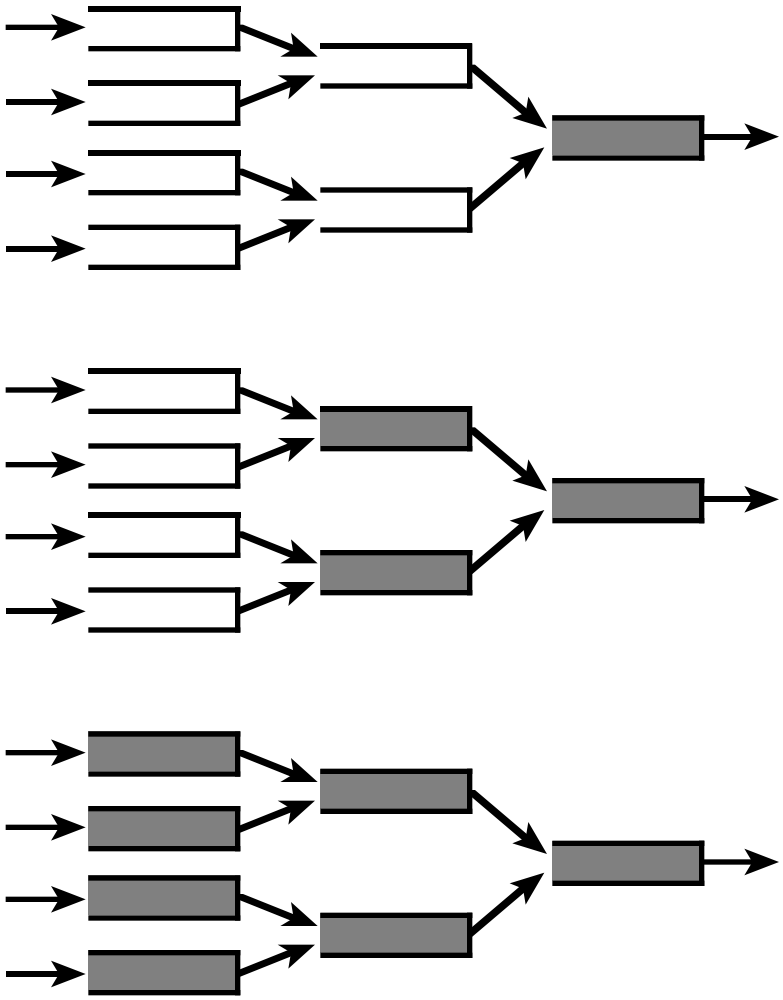


Figure 22: Tree saturation

The RP3 also emphasizes and studies the use of caches in the PE's. They notices a miss rate of 1% in read-only data access (typically code), while read-write accesses miss the cache 20% of the time. Use of the caches can thus hinder the operation of the combining network; since non-combined accesses stay local to the PE, the percentage of the network load which accesses hotspots is increased. In order to cache data, it is assumed that the status of a variable (hot/cool) is known at load time [PfBG85]. The cache uses store-through to maintain consistency; this increases overall network traffic, but reduces the burstiness of that traffic, compared to copy-back schemes [BrMW85].

The processor cache uses prefetch, and has immediate internal acknowledgement of store requests - imitating the short circuit operation of a combining network in the presence of multiple stores [BrMW85]. Normally, only one outstanding prefetch on shared data is permitted (i.e. blocking requests), but the limit is enforced by a software-controllable fence register, permitting multiple outstanding requests. The number of outstanding requests is limited to ensure serializability, but in many cases the limit of a single pending access is conservative. The fence permits adjusting the limit in cases where serializability is known to survive more than one pending request. In addition, local cache stores do not cause immediate network access; only after a subsequent miss is a write-through performed. This reduces traffic when only local writes are performed.

Other techniques were considered to optimize the network design. The use of a software combine was suggested, specifically to reduce the hotspots caused by global summation operations. The algorithm suggested is essentially a partial sum operation [YewTL87]. The design implemented used a two-level network, where one level performs combines (slow) and the other only routes (fast) (Figure 23). This assumes both that a processor knows the status of an access (hot/cool), in order to route the request. The partitioning of requests prevents some combines from occurring, and also there may be cases when it is better to use a faster non-combining network, rather than a slow network which might combine, especially in a lightly loaded system.

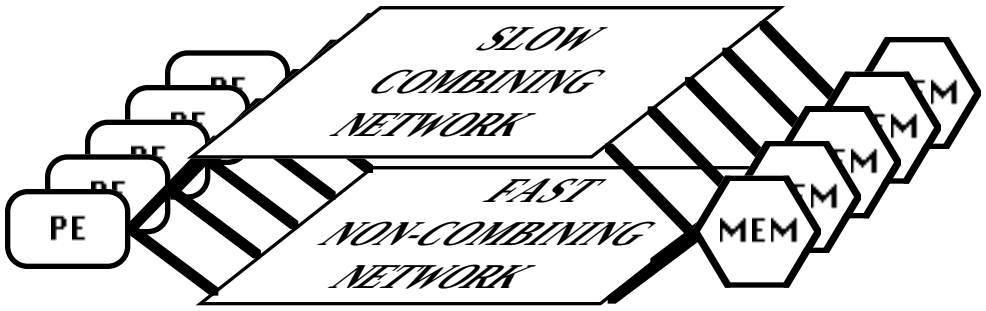


Figure 23: Two-level RP3 design

4.3.3. The AT&T Starlite multicast network

The AT&T Starlite network is a packet network, similar to a message passing network. It is composed of a centrally-buffered recirculation network, the main path of which is an unbuffered Batcher-banyan network, as was seen in the related Bellcore network [HuaK84] (Figure 24). The use of a shared internal buffer precludes the ability to gang the inputs in the network if recirculation is permitted, but this is one of the first papers to state the ganging

potential of switch fabrics that have constant internal latency (no internal link-based buffering).

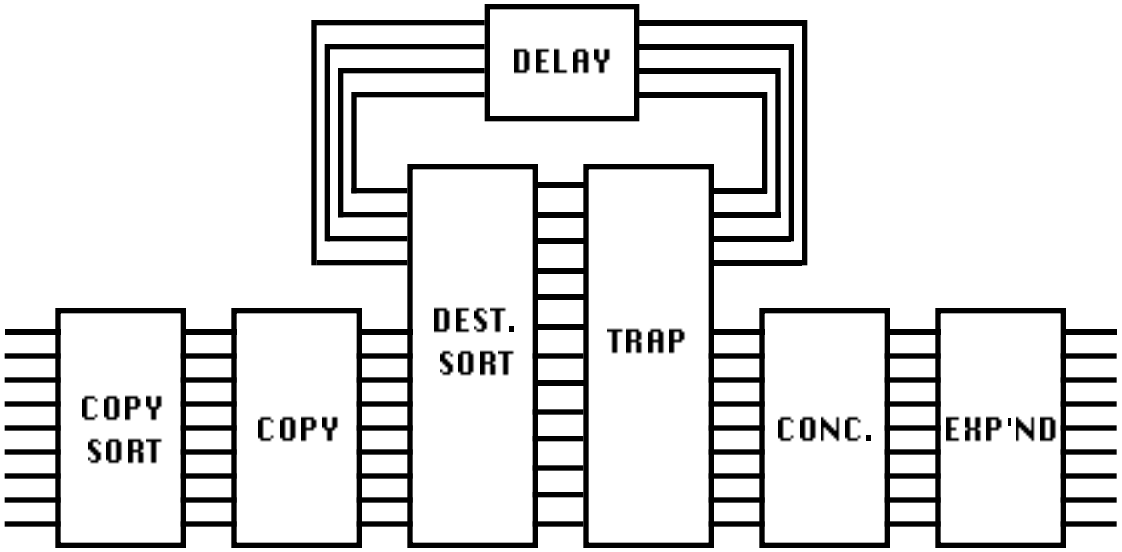


Figure 24: Starlite network

The Starlite performs a sequence of operations as a multicast request passes through the network (Figure 25). First, packets are not explicitly copied - no single packet actually generates additional packets in the network. Instead, extra empty packets are generated, either by the sender of a multicast request or by the receivers of that multicast. The sets of source and empty packets are first sorted (sort-to-copy), where source packets head each group of empty packets requesting copies from that source. In the copy stage, the contents of the source packet are copied into each empty packet below it, until another source packet is encountered (note that this implies that a serial copy occurs, even though faster, logarithmic copy schemes may be possible). The replicates need not be distinguished, as is required in other copy networks [Let88] [Tu88], since each destination packet is pre-addressed when it is generated. Since the switch elements performing a copy actually input two packets (source and empty), the switch element internal function is a special case of a reduction/broadcast function in the switch element (Figure 26).

The resulting packets proceed through a Batcher-banyan sequence, called here ‘sort-to-destination’ and ‘expander’, including the trap and concentrator intermediate phases, which remove packets which would cause output port collisions.

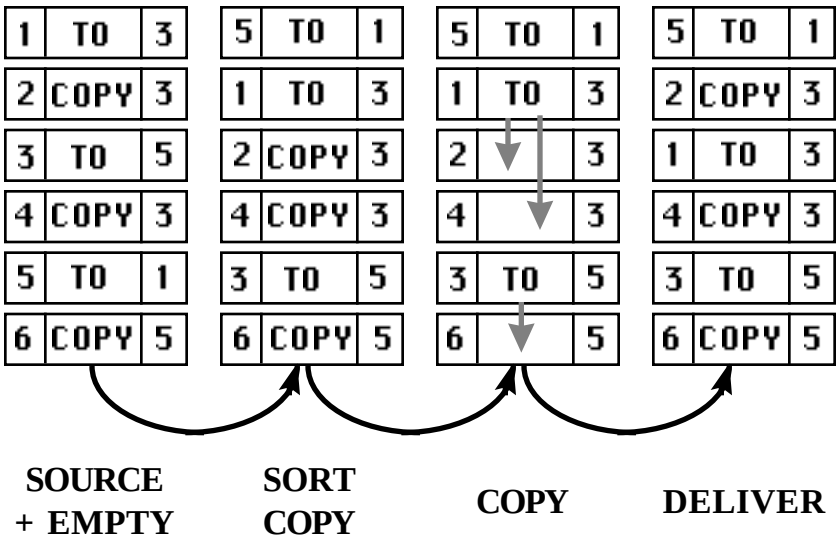


Figure 25: Starlite multicast sequence

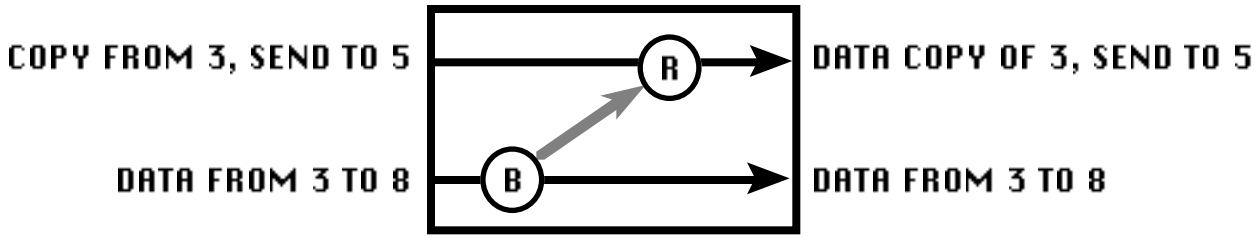


Figure 26: Starlite switch element function - broadcast/reduce

In the design outlined, empty copy packets are generated either serially by the sender of a multicast or by receivers of that multicast. In implementation, neither would be simple to design, since the former implies a serial generation, where the source might just have well performed a serial copy of the packet data as well. The latter requires that source and empty packets of a multicast set arrive at a switch in the same cycle, requiring the alignment or synchronization of communication in a multicast set. A simpler scheme would relegate the task of empty packet generation to a separate set of processors (such as the BGT's of [Tu88] or TNT's of [Let88]) that are programmed at call setup time. At a specified time, in synchronization with the sender, they would generate a set of empty packets in parallel.

The use of copy receptors in the Starlite network is similar to the use of dummy messages in a network proposed at Stanford [Mo79] (this was not noted in the Starlite paper). The goal of this network is to eliminate the output port contention problem; this is done by inputting a set of dummy packets, one for each output (these dummies are automatically generated, or

hardwired as input). These dummy packets are sorted with input packets via a Batcher sorting network. At this point each set of destinations is led by a dummy packet, followed by the set of input packets vying for the output port. The top two packets are effectively exchanged, in a way which delivers that top input packet to the output port desired, and the dummy packet is routed back to the source that sent the input packet which was successfully delivered, as a positive acknowledgement. Other packets are delivered back to their input ports, for retries, as negative acknowledgements.

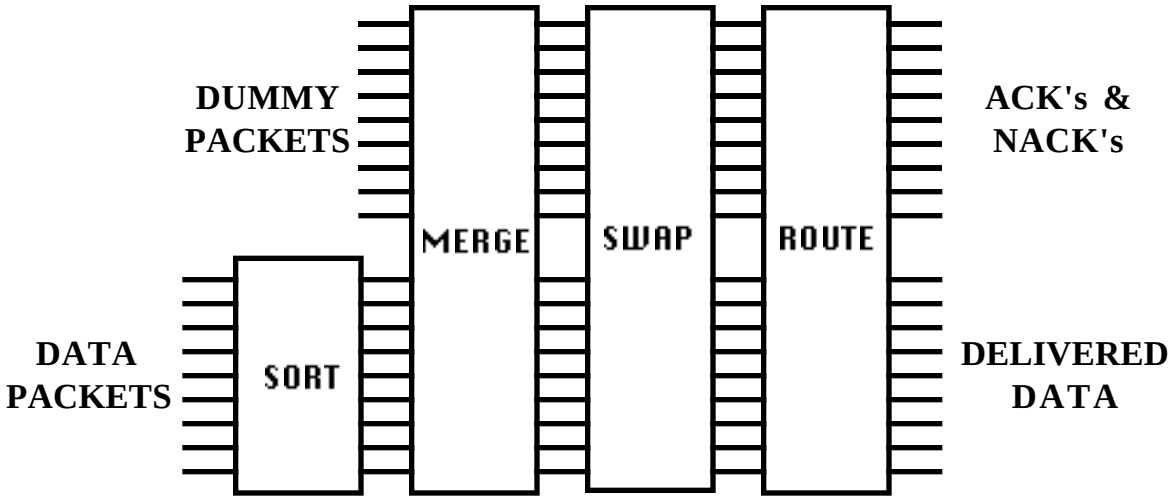


Figure 27: Use of dummy packets

4.3.4. The AT&T Bell Labs Knockout multicast-capable network

The Knockout network is not a MIN network, but its design does incorporate multicast capability, and is useful for comparison [YehHA87] [EnHY88]. The network is fully connected, in a multiple bus configuration, and supports packet switched communication (Figure 28).

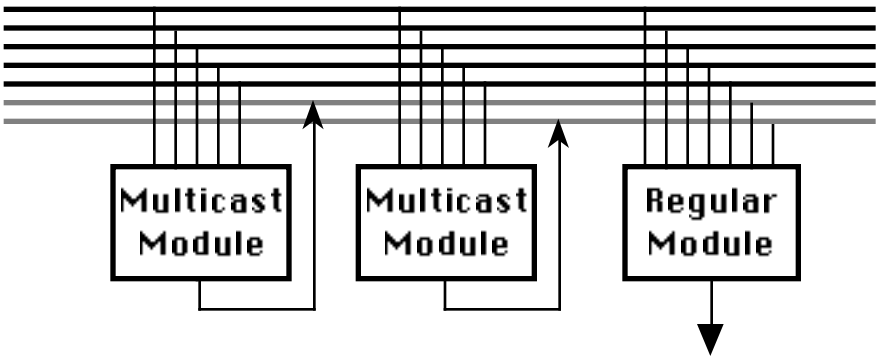


Figure 28: Knockout switch multicast modification

The switch is composed of a set of busses, one from each PE, and a bus-collection device, a Knockout concentrator, one into each PE. The concentrator looks at packets on all busses, and collects those packets for its PE's address. Since more than one packet can be collected in a cycle, the concentrator must buffer or resolve contentions before the packets arrive at the PE; this is performed by a 'knockout' elimination tournament in the concentrator.

There has been an extension recently suggested for the Knockout network which would support multicast requests [EnHY88]. It is composed of multicast modules on the Knockout bus, much like concentrators with internal processors, but without host PE's (also Figure 28). Broadcast traffic is transmitted on the conventional busses, where the destination is a 'universal donor' (borrowing from blood-typing terminology), so all concentrators recognize that particular address as compatible with their own hardwired address.

For multicasts, a virtual multicast set identifier replaces the usual destination address. The multicast modules recognize this identification, and intercept the packet. At this point, one of two resulting operations is possible.

In the serial multicast design, called the packet replicator, the multicast module has the entire multicast group table in memory. When it receives a multicast request, it reads it off the sender's bus, and holds the request in memory. In the next cycles, the multicast module replicates the packet for each destination in the multicast set table, and places the resulting packets on its output bus. The other concentrators behave normally, reading the multicast bus as any other bus, and receiving the multicast replicates for their address as usual. Note the fundamental operation of this design: serial multicast, where the entire multicast table is in the multicast module, and the concentrators are unchanged.

In the parallel multicast design, called the 'fast packet filter', multicasts are recognized and placed on the multicast bus, unaltered (so there is really no reason for a multicast module at all in this design). Here the concentrators are modified, where they include a table of virtual multicast set identifiers and fast comparison hardware. Packets addressed for the concentrator, or whose virtual identifications are in the internal table, are received at the concentrator. In this scheme, the multicast module is nonexistent functionally, but the concentrators require fast lookup hardware to monitor the multicast bus, and each store some virtual addresses. No explicit replication occurs; all concentrators read simultaneously off the same bus.

It is interesting to note that the serial packet replicator actually copies each packet and differentiates it by an index, as in the first stage of the other copy networks by Bellcore and

WashU [Let88] [Tu88]. The parallel fast packet filter operates similarly to the BGT's of the WashU network or the TNT's of the Bellcore network. So the MIN copy networks have both packet replication and fast packet filters in their designs.

4.3.5. The Columbia ChoPP multicast-capable network

The Columbia ChoPP (Columbia Homogeneous Parallel Processor) is a message passing network which introduces the concept of a multicast as a partitioned broadcast [SuB77] [SuBK77]. Multicasts are described as dynamically partitioned broadcast, and broadcast regions. This is notable as the earliest network implementation found which indicates the potential of multicast capability.

4.3.6. The Louisiana combining network

The University of Southwestern Louisiana is developing a unique shared memory combining network [Tz89] (Figure 29). In this network, combination is distinct from routing, in a partition akin to the copy-route partition in multicast networks. By factoring the network into these components, sequencing of requests can be maintained.

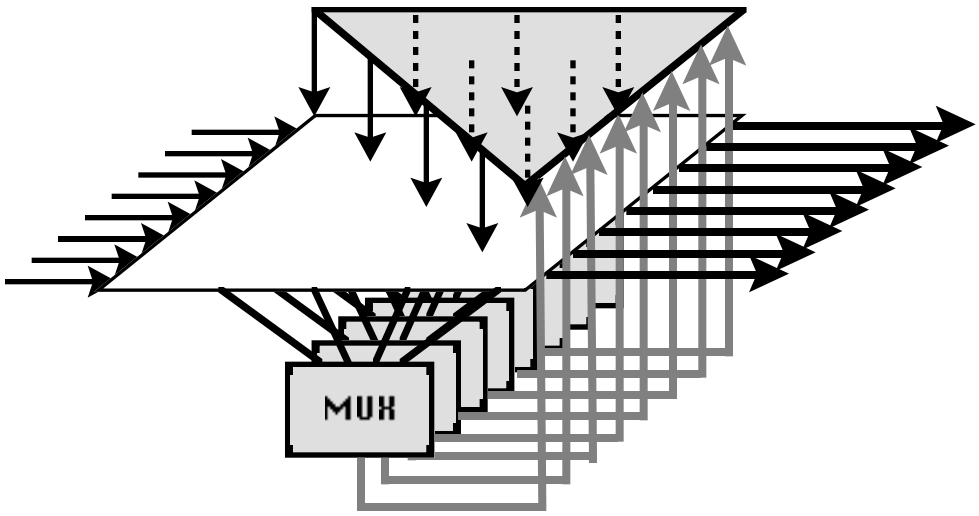


Figure 29: Louisiana combining structure

The network is composed of a routing network, with a combining structure which can detour packets from the routing network into the combine, and back into the router after combination occurs. All hotspot accesses are rerouted into the combine network, while other requests proceed directly to their destination, as in the two-level RP3 implementation. Since hotspot

requests are moved out of the main routing path, tree saturation cannot affect non-hotspot traffic.

There are two types of reduction considered in the nodes of the combining network. Systolic combine operates with a snaking queue, as in the NYU Ultracomputer output port queues. Simultaneous combination involves an associative table access, as is assumed the wait queue in the NYU operates.

One of the gains claimed in this network is that only N reducing elements are required for the combining network, while NlogN are required in most other schemes, such as the RP3 and NYU. While this is accurate, the gain is offset by the need for logN concentration of each input of the combining network, in order to facilitate routing into the combining network from any stage in the routing network.

Other difficulties in this design include the fact that combinations occur only within partitions of powers of 2, due to the tree structure of the combining network. In this scheme, two requests arriving on either side of the center of the combination network would not combine until they reach the root of the network. Other combining networks form a tree based at any location in the network, and reduce the pairs as quickly as possible.

4.4. Other domains

Other kinds of networks provide different multicast and reduction capabilities. Here we consider mostly multicast for broadcasting packet traffic, and reduction to eliminate memory hotspot contention. Other kinds of networks define the copy and reduce functions to facilitate the purposes of the packets, such as join networks for database manipulations. In these networks, the interconnection networks are structured in a way which facilitates reductions based on database combinations, and the switch elements perform joins internally [MenBL87].

Some reduction networks are designed specifically to perform the cascaded sum of the RAN in the Bellcore switch, or other arithmetic tree reductions. These networks are used in fast matrix and vector computers. Although these organizations are not strictly reduction networks, the principles are identical.

5. Observations

There are two other observations which are useful to note. First, reduction networks are based on the hotspot memory contention problem, and all implementations found assume sharing of hotspots among large sets of processors, where the data locations are static. It would be useful to consider how hotspot combining extends to a virtual data object scheme, where the location of the object is not explicitly known, but combining can occur at the object manipulation level. Current reduction operations support only integer reduction facilities. A parallel to the join reduction in database networks (i.e. higher level reduce operation) should be investigated for high level objects.

In multicast networks, the most difficult problem is the atomicity of multicast request satisfaction, and the determination of multicast scheduling in a distributed fashion. It would be useful to consider this distributed resource allocation problem, without permitting global knowledge or call processing facilities.

References

[Ba68] K.E. Batcher, “Sorting Networks and Their Applications”, AFIPS Spring Joint Computer Conf. Proceedings, 1968, p307-314.

[Be62] V.E. Benes, “On Rearrangeable Three-Stage Connecting Networks”, Bell System Technical Journal, Sept. 1962, p1481-1492.

[BrMW85] W.C. Brantley, K.P. McAuliffe, and J. Weiss, “RP3 Processor-Memory Element”, IEEE Proc. 1985 Int'l. Conf. on Parallel Processing, p782-789.

[BuT89] R.G. Bubenik and J.S. Turner, “Performance of a Broadcast Packet Switch”, IEEE Transactions on Communications, Vol. 37 No. 1, Jan. 1989, p60-69.

[Ch82] E.J.H. Chang, “Echo algorithms: depth parallel operations of graphs”, IEEE Transactions on Software Engineering, Vol. SE-8 No. 4, Jul. 1982, p310-400.

[ChM84] K.M. Chandy and J. Misera, “The Drinking Philosophers Problem”, ACM Transactions on Programming Languages and Systems, Vol.6 No 4, Oct. 1984, p632-646.

[Cl53] C. Clos, “A Study of Non-Blocking Switching Networks”, Bell System Technical Journal, Mar. 1953, p407-424.

[DaT89] G.E. Daddis and H.C. Torng, “A Taxonomy of Broadband Integrated Switching Architectures”, IEEE Communications Magazine,Vol. 27 No. 5,May 1989, p32-42.

[De89] W. Denzel, “Switching Fabrics for High Speed”, IBM European Summer Symposium on Very High Speed Networks - Garmisch FRG, 1989.

[DiK89] D.M. Dias and M. Kumar, “Preventing Congestion in Multistage Networks in the Presence of Hotspots”, IEEE Proc. 1989 Int'l. Conf. on Parallel Processing, Vol. 1, p9-13.

[EdGK85] J. Edler, A. Gottlieb, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, M. Snir, P.J. Teller, and J. Wilson, “Issues Related to MIMD Shared-memory Computers: the NYU Ultracomputer Approach”, IEEE Twelfth Annual Symposium on Computer Architecture, 1985, p126-135.

[EnHY88] K.Y. Eng, M.G. Hluchyj, and Y.S. Yeh, “Multicast and Broadcast Services in a Knockout Packet Switch”, 1988 IEEE Infocom, p29-34.

[Fl66] M.J. Flynn, “Very High-Speed Computing Systems”, Proc. of the IEEE, Vol. 54 No. 12, Dec. 1966, p1901-1909.

[GopJ84] I.S. Gopal and J.M. Jaffe, “Point-to-Multipoint Communication Over Broadcast Links”, IEEE Transactions on Communications, Vol. COM-32 No. 9, Sept. 1984, p1034-1044.

[Got84] A. Gottlieb, “Avoiding Serial Bottlenecks in Ultraparallel MIMD Computers”, IEEE Compcon, 1984, p354-359.

[GotGK83] 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 Transactions on Computers,Vol. C-32 No. 2, Feb. 1983, p175-189.

[GotK81] A. Gottlieb and C.P. Kruskal, “Coordinating Parallel Processors: A Partial Unification”, ACM Computer Architecture News, Vol.9, Oct. 1981, p16-24.

[GotLR83] A. Gottlieb, B.D. Lubachevsky, and L. Rudolph, “Basic Techniques for the Efficient Coordination of Very Large Numbers of Cooperating Sequential Processors”, ACM Transactions on Programming Languages and Systems, Vol. 5 No. 2, Apr. 1983, p164-189.

[HaMS86] J.P. Hayes, T. Mudge, and Q.F. Stout, “A Microprocessor-based Hypercube Supercomputer”, IEEE Micro, Oct. 1986, p6-16.

[HocJ86] R.C. Hockney and C.R. Jessope, Parallel Computers, Adam Hilger Ltd., Bristol, 1986, p158-178,261-265.

[HoE89] W.S. Ho and D.L. Eager, “A Novel Strategy for Controlling Hot Spot Congestion”, IEEE Proc. 1989 Int'l. Conf. on Parallel Processing, Vol. 1, p14-18.

[HuaK84] A. Huang and S. Knauer, “STARLITE: A Wideband Digital Switch”, IEEE Globecom, 1984, p121-125.

[HuiA87] J.Y. Hui and E. Arthurs, “A Broadband Packet Switch for Integrated Transport”, IEEE Journal on Selected Areas in Communications, Vol. SAC-5 No. 8, Oct. 1987, p1264-1273.

[JiP85] L. Jin and Y. Pan, “A Kind of Interconnection Network with Mixed Static and Dynamic Topologies”, 1985 IEEE Int'l. Conf. on Distributed Computing Systems, p160-166.

[KapK89] M.A. Kaplan and B. Kadaba, “High Speed Networking at IBM Watson Research”, IBM European Summer Symposium on Very High Speed Networks - Garmisch FRG, 1989.

[Kat87] H.P. Katseff, “Flow-Controlled Multicast In Microprocessor Systems”, 1987 IEEE Phoenix Conf. on Computers and Communications, p8-13.

[KhT87] S. Khakoo and J.S. Turner, “System Testing of a Broadcast Packet Switch”, Washington University Technical Report WUCS-87-4, Apr. 1987.

[Kr82] C.P. Kruskal, “Algorithms for Replace-Add Based Paracomputers”, IEEE Proc. 1982 Int'l. Conf. on Parallel Processing, p219-223.

[KuP86] M. Kumar and G.F. Pfister, “The Onset of Hot Spot Contention”, IEEE Proc. 1986 Int'l. Conf. on Parallel Processing, p28-34.

[LegKK86] G. Lee, C.P. Kruskal, and D.J. Kuck, “The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of ‘Hot Spots’”, IEEE Proc. 1986 Int'l. Conf. on Parallel Processing, p35-41.

[Ler85] R. Lee, “On ‘hot spot’ contention”, ACM Computer Architecture News, Vol. 13, Dec. 1985, p15-20.

[Let88] T.T. Lee, “Nonblocking Copy Networks for Multicast Packet Switching”, IEEE Journal on Selected Areas of Communications, Vol. 6 No. 9, Dec. 1988, p1455-1467.

[LetBA88] T.T. Lee, R. Boorstyn, and E. Arthurs, “The Architecture of a Multicast Broadband Packet Switch”, 1988 IEEE Infocom, p1-8.

[Mel88] J.M. Mellor-Crummey, “Experiences with the BBN Butterfly”, IEEE Compcon, 1988, p101-104.

[MenBL87] B. Menezes, D. Brant, D. Loewi, A. Dale, and R. Jenevein,, “An Interconnection Network Supporting Relational Join Operations”, 1987 IEEE Int'l. Conf. on Distributed Computing Systems, p128-135.

[Mo79] H.P. Moravec, “Fully Interconnecting Multiple Computers with Pipelined Sorting Nets”, IEEE Transactions on Computers, Vol.C-28 No. 10, Oct. 1979, p795-798.

[NaS81] D. Nassimi and S. Sahni, “Data Broadcasting in SIMD Computers”, IEEE Transactions on Computers, Vol. C-30 No. 2, Feb. 1981, p101-106.

[PfBG85] G.F. Pfister, W.C. Brantley, and D.A. George, et.al., “The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture”, IEEE Proc. 1985 Int'l. Conf. on Parallel Processing, p764-771.

[PfN85] G.F. Pfister and V.A. Norton, “‘Hot Spot’ Contention and Combining in Multistage Interconnection Networks”, IEEE Transactions on Computers, Vol. C-34 No. 10, Oct. 1985, p943-948.

[Ro87] G.H. Robbert, “Design of a Broadcast Translation Chip”, Washington University Technical Report WUCS-87-9, Apr. 1987.

[Sc80] J.T. Schwartz, “Ultracomputers”, ACM Transactions on Programming Languages and Systems, Vol. 2 No. 4, Oct. 1980, p484-521.

[SuB77] H. Sullivan and T.R. Bashkow, “A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I”, IEEE Fourth Annual Symposium on Computer Architecture, 1977, p105-117.

[SuBK77] H. Sullivan, T.R. Bashkow, and D. Klappholz, “A Large Scale, Homogeneous, Fully Distributed Parallel Machine, II”, IEEE Fourth Annual Symposium on Computer Architecture, 1977, p118-124.

[Th86] R.E. Thomas, “Behavior of the Butterfly Parallel Processor in the Presence of Memory Hot Spots”, IEEE Proc. 1986 Int'l. Conf. on Parallel Processing, p46-50.

[To87] J.D. Touch, [proprietary documents], Bell Communications Research, 1987.

[Tu87a] J.S. Turner, “Specification of Integrated Circuits for Broadcast Packet Network”, Washington University Technical Report WUCS-87-5, Apr. 1987.

[Tu87b] J.S. Turner, “The Challenge of Multipoint Communication”, Proc. of the ITC Seminar on Traffic Engineering for ISDN Design and Planning, May 1987.

[Tu87c] J.S. Turner, “Fluid Flow Loading Analysis of Packet Switching Networks”, Washington University Technical Report WUCS-87-16, 1987.

[Tu87d] J.S. Turner, “Notes on Extensions to Starlite”, private note to T.T. Lee.

[Tu88] J.S. Turner, “Design of a Broadcast Packet Switching Network”, IEEE Transactions on Communications, Vol. 36 No. 6, June 1988, p734-743.

[Tz89] N.F. Tzeng, “Design of a Novel Combining Structure for Shared-Memory Multiprocessors”, IEEE Proc. 1989 Int'l. Conf. on Parallel Processing, Vol. 1, p1-8.

[Wa82] D.W. Wall, “Selective Broadcast in Packet-Switched Networks”, Proc. Sixth Berkeley Workshop on Distributed Data Management and Computer Networks, 1982, p239-258.

[WuF80] C.L. Wu and T.Y. Feng, “On a Class of Multistage Interconnection Networks”, IEEE Transactions on Computers, Vol. C-29 No. 8, Aug. 1980, p694-702.

[YehHA87] Y.S. Yeh, M.G. Hluchyj, and A.S. Acampora, “The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching”, IEEE Journal on Selected Areas of Communications, Vol. SAC-5 No. 8, Oct. 1987, p1274-1283.

[YewTL87] P.C. Yew, N.F. Tzeng, and D.H. Lawrie, “Distributed Hot-Spot Addressing in Large-Scale Multiprocesors”, IEEE Transactions on Computers, Vol. C-36 No. 4, Apr. 1987, p338-395.