A Case for Aggregate Networks

Raymond R. Hoare and Henry G. Dietz
{ hoare | hankd } @ecn.purdue.edu

School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907

Abstract

Parallel processing networks, even full crossbars, that only implement point-to-point and multicast message passing are inefficient for collective communications because multiple messages must be transmitted to/from each processor to implement a single collective operation. However, all of the information needed for a collective communication can be made available to the network control logic within a single communication. By making this control logic capable of executing functions on the information aggregated from all of the processors, any collective communication can be implemented without additional messages or processor involvement. Networks with such logic are called aggregate networks and are capable of performing routing, computation, and storage/retrieval of global information. This paper gives a detailed example of each of these types of aggregate functions.

1. Introduction

Advances in microprocessor-based system performance, and the emergence of a commodity market for personal computers, makes parallel processing using clusters of commercial-off-the-shelf microcomputers an attractive technology. Local area networks (LANs) have also dramatically improved and dropped in price, with gigabit bandwidths and microsecond latencies available at reasonable cost. However, most of these networks are based on sending data from one processor to one or more processors. Using such networks, collective operations such as *Personalized All-to-All*, which require the participation of a group of processors, must be implemented using multiple point-to-point or multicast communications.

In this paper, we suggest that the network control logic should be a function unit that is able to concurrently execute one N-input function per processor per unit time, and not simply a passive message router. Given this ability, not only can collective operations be performed within the network, but a wider class of aggregate functions can become equally efficient.

This paper introduces *aggregate networks,* networks that are capable of executing *aggregate functions*, and provides three motivating examples of *aggregate operations. Aggregate operations* are functions whose parameters, return values, or internal data are associated with more than one processor. The execution of these functions is performed by the network within a central hub called the *Aggregate Function Unit (AFU)* and the result(s) of these functions is then sent back to the appropriate processor(s). In this way, each processor has to perform only two I/O operations: one to place a packet, containing the data and opcode, into the network and one to receive the result packet.

We will motivate this idea by describing three very different aggregate operations and by showing that these operations execute faster using an aggregate network than using a crossbar. A variety of other approaches to putting computation in the network are described in Section 2. Sections 3, 4, and 5, respectively, describe aggregate network implementations of *Personalized All-to-All, Summation* (and other associative reductions), and access to global information. Conclusions and future work are described in Section 6.

2. Related Work

The NYU Ultracomputer [GoG83][BiD93] and the IBM RP3 [PfN85] are both shared memory architectures in which the network interconnects each processor with each memory element. The Ultracomputer was the first architecture to propose combining within the network for messages that reference identical memory locations. Fetch-and-Add, Fetch-and-Increment, and other Fetch-and-Op functions require that the operation be associative and co-resident within a switch.

Active Messages from Berkeley [Cul96] allow functions to be executed at the network interface on the local or remote node but not within the network. Active Networks perform operations on single messages that pass through the network [TeS97][TeW96]. Sorting networks route messages based on the relative rank of their data with respect to the other messages [Bat68] [LeB96][Wen96]. Multistage data manipulation networks are discussed in general in [Sie90].

The Cray T3D hardware directly supports barrier synchronization, swap, and Fetch-and-Increment [AlG94]. The TMC Connection Machine CM-5 has a control network that supports reduction operations, prefix operations, maximum, logical OR and XOR [AlG94]. These architectures can be considered aggregate networks but they are very specific in the functions that they are designed to execute.

PAPERS, Purdue's Adapter for Parallel Execution and Rapid Synchronization, is a network that allows a number of aggregate computations to be performed within a custom network hub that is attached to a cluster of Linux PCs [DiH96][HoD96][Mat97]. This design uses a combination of barrier synchronization with a four-bit wide global NAND to construct a robust library of aggregate computations and communications.

Our generalized notion of aggregate networks extends the concepts from PAPERS with higher-level functions and memory within the AFU.

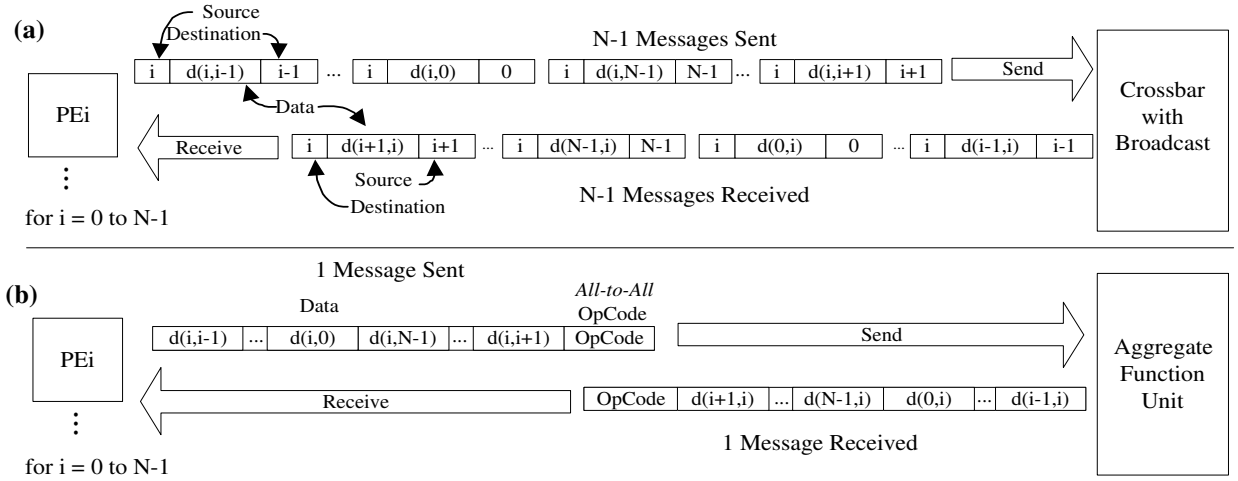

Figure 1. ***Personalized All-to-All*** **(a) Using a Crossbar; (b) Using an Aggregate Function Unit.**

3. Personalized All-to-All

Every operation that is executed within the network involves communication. The number of messages sent, along with the number of processors that are involved, characterize all communications. The simplest of these is a single message that is sent between two processors and is referred to as a *point-to-point communication*. Communications that involve more than one source or more than one destination are referred to as *aggregate communications*. The MPI library [SnO96], Collective Communication library [BaB95][LoB96], and parallel languages like High Performance Fortran (HPF) support a variety of aggregate communications.

To demonstrate how the network can be used to execute high-level aggregate communication operations, we will compare the execution times for a fixed-field-length *Personalized All-to-All* using a crossbar versus using an otherwise comparable *aggregate network*.

For a fixed-field-length *Personalized All-to-All* aggregate communication, N processors each send a unique datum to each of the other processors. For simplicity, we will assume that each processor contains an array, d, whose elements need to be distributed evenly among the other processors.

If a crossbar is used (Figure 1a.), communication can be overlapped such that the $N(N\text{-}1)$ point-to-point communications require only N-1 communication phases. This is done by scheduling the communications so that in each phase only one message is sent to each destination. During phase j (j =1 to N-1) and for i = 0 to N-1, PE_i sends $d_{(i+j) \bmod N}$ to $PE_{(i+j) \bmod N}$. Thus, contention is avoided and the total time that is required is that of *N-1* point-to-point communications.

To avoid contention, each communication phase must complete before the next phase starts. Conceptually, a barrier synchronization must be performed between each communication phase. For a crossbar, this can be done by using flow control logic which can keep messages from being sent by a single source when that source is still sending the previous message. If all of the processors start sending their data at exactly the same time and if all delays are exactly even for all of the processors, then the crossbar will not have any contention. However, if a delay is seen by some of the processors and not by the others, then the communication phases can overlap and contention can occur.

If an aggregate network is used, the scheduling algorithm can be implemented *within* the Aggregate Function Unit (AFU) by adding a barrier synchronization unit that requires only tens of nanoseconds, as described in [DiH96]. Thus, the extra synchronization phases added between the communication phases are performed within the AFU in as little as one clock cycle.

The AFU knows that the *Personalized All-to-All* operation is to be performed because an OpCode is placed in the header of the message. The payload of the message is the collection of data that is to be distributed to the other processors, shown in Figure 1b. The source of the message is known because the network topology is fixed and the destination of the fields is also known because the OpCode is given. Thus, the aggregate network only has to send one message to the AFU where a crossbar has to send *N-1* messages. Furthermore, the total amount of data that needs to be transmitted is less because a crossbar requires a source and destination tag with each message.

As the field sizes decrease, the amount of overhead increases for a crossbar. Conversely, when the field size becomes too large to buffer, the benefits of using the AFU decrease. The AFU has two options for large data fields:

(1) Break down the large fields into smaller ones that can be buffered and perform a multiple *Personalized All-to-All* communications. or

(2) Perform barrier synchronizations within the AFU between each of the communication phases. For a cluster of PCs or workstations, this synchronization can be coordinated between network interface cards and the AFU. In essence, the AFU would still perform the scheduling of the operation but the data would be buffered either in the network interface cards or in memory. If this method is used, the field sizes would not have to be uniform because each processor could send

an *end-packet* marker when it is finished. However, if fixed size packets are used, the AFU could determine if a phase is complete by counting the number of bits that each processor has sent; thus, additional messages would not be needed and barrier synchronization time would be practically negligible.

For an aggregate network, a *Gather* communication operation is a subset of the *Personalized All-to-All* operation because there is only one destination and *N-1* sources, each sending a unique piece of data. The messages that are sent only contain a single field of data, but the message that is received by the one gathering processor contains *N-1* fields. A similar method can be used for a *Scatter* operation, which is also a subset of the aggregate *Personalized All-to-All.*

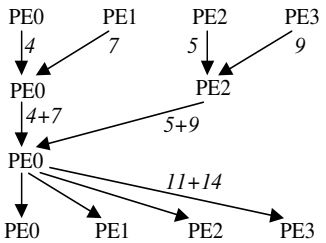

Figure 2. *Summation* of 4+7+5+9.

4. Aggregate Computation

In the previous section, we described how the network can be used to perform high-level communication routing functions. In this section, we focus on aggregate computations and how they can be implemented using either a crossbar or an aggregate network.

Enabling the network to perform *functions* implies that the data payload of a packet that is received from the network is a function of the data aggregated from a collection of processors - not the verbatim data sent into the network by a single processor. If these functions involve computation, they are defined as an *aggregate computations.*

For associative operations involving *N* data elements located on *N* different processors, *(N-1)* two operand computations must take place. All of the data could be sent to a single processor for execution and the result could then be broadcast. This would require a *Gather* operation which requires *N-1* communication phases (for a crossbar.) However, only $\log_2 N$ phases are required because a binary tree communication pattern can be used. As an example, a *Summation* (also called a *Reduce-to-All Add*) is shown in Figure 2. Each processor outputs a single value and the resulting sum is given to all processors. To do this the leaves of the tree send their values up the tree to the intermediate nodes which calculate the partial sums. These partial sums are then sent farther up the tree to be added together until the root of the tree has the final sum which is then broadcast to all of the processors. This requires $\log_2 N$ phases of compute and communication plus one final broadcast of the result. This method works well for all reduction operations and can be used for any associative operation.

Generalizing to more complicated aggregate computations, we may wish to perform computations on each pair of operands that require many clock cycles. To this end, we will assume that *k* is the number of clock cycles that are required to perform a single two-operand computation.

Thus, for *N* processors, a total of *(N-1)*k* clock cycles are needed to compute the aggregate computation. To get a rough estimate of the amount of time that is required to execute an aggregate computation, we will make a few assumptions: a binary tree communication pattern (as demonstrated in Figure 2), a crossbar (with multicast) is used in the network hub, each PE has a clock cycle of ε and that *L* is the total amount of time required to perform a single point-to-point communication.

$$T_{CrossbarComputation} = (L + k\varepsilon)\log_2 N + L \quad \text{(Eq. 1)}$$

If an Aggregate Network is used and all of the operands are all sent to the AFU, the AFU then performs the computation and broadcasts the result to all of the processors. To be very conservative we will assume that the AFU contains a serial processor that has a clock cycle of ε' time. Thus, the time required to execute an aggregate computation is that of the *serial* computation plus that of a single point-to-point communication, because placing data into the network only requires 1/2 *L,* as does receiving the data from the network.

$$T_{AFUComputation} = ((N-1)k\varepsilon') + L \quad \text{(Eq. 2)}$$

Thus, the speedup is then:

$$T_{ComputationSpeedup} = \frac{(L + k\varepsilon)\log_2 N + L}{((N-1)k\varepsilon') + L} \quad \text{(Eq. 3)}$$

If *k* is large with respect to *L,* then aggregate networks will perform better than a comparable crossbar. More specifically, the following inequality must hold for aggregate networks to outperform a crossbar network (i.e. speedup >1):

$$\frac{\log_2 N}{\varepsilon'(N-1) - \varepsilon\log_2 N} > \frac{k}{L} \quad \text{(Eq. 4)}$$

Assuming that the PE's processor is twice as fast as the AFU processor (i.e. ε'=2ε) the inequality can be simplified to show the relationship between *N* and *kε/L*

$$\frac{\log_2 N}{2(N-1) - \log_2 N} > \frac{k\varepsilon}{L} \quad \text{(Eq. 5)}$$

Figure 3 graphs *k(N-1)*, the total number of instruction cycles for the computation, versus N, the number of processors. Figure 3 assumes: ε'=10ns (a 100

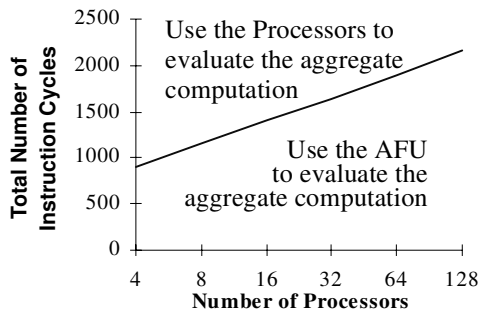

Figure 3. Central Versus Parallel Computation.

MIPS processor for the AFU processor), ε=5ns (a 200 MIPS processor for the PEs), and L=3μs (the average communication delay).

The area above the line represents computations that should be performed on the PEs across the network while the area under the line represents computations that, due to network latencies, should be executed *within* the network at the AFU.

5. Aggregate/Global Information

Aggregate information is data that is associated with more than one processor. This information tends to pertain to shared resources or to the state of the processors. For example, a shared resource table can let each of the processors know where a particular file (or part of a file) is located, who has access rights to it and other such information associated with the files. Distributed shared memory can also be tracked by using a common table so that the data can migrate to where it is needed. On a smaller scale, an aggregate network can contain a finite amount of memory that each of the processors has access to, thereby implementing a true shared memory. The structure of this memory can also be specified. For example, a globally accessible queue can be implemented in which all of the processors can add or remove elements from a single queue. This would enable dynamic load balancing for jobs that have multiple tasks that can be indexed. Thus, when a processor completes a task, another task index can be dequeued and processing resumes. In this way, faster processors can have access to more tasks while slower processors can spend more time on their task without slowing other processors down.

One of the major hurdles in clustering workstations is how one coordinates and synchronizes each of the workstations in a timely manner so that medium and fine grain applications can be executed efficiently. Therefore, some of this aggregate information can be the state of some or all of the processors. For example, a barrier synchronization, or *wait* operation, requires that each processor execute the *wait* instruction before any processor may continue executing code. The number and identities of the processors that are participating in the barrier synchronization can also be specified so that multiple barriers can take place at the same time. After the barrier groups are determined, the only amount of data that is required from each processor is simply a signal or flag that specifies that processor has reached the barrier. Thus, having the network keep track of this information can greatly speed up barrier synchronization as well as other operations that require aggregate information.

If aggregate information is to be stored within the network then the processors must be able to access this information in a timely fashion. To quantify our performance we will compare an aggregate network to a point-to-point network that has global information distributed among the processors. We will assume that the data can be indexed; by using this index the location of the data is known to all of the processors. We will also assume that the information in any particular table is randomly accessed by all of the processors. (If this is not the case, then an optimization could be made for point-to-point networks as well as for aggregate networks.) We also assume that remote reads and writes take the same amount of time, and that local accesses are negligible.

For a point-to-point network in which N processors share data through a distributed shared table, the amount of time required for a processor to access a record is $T_{Distributed\text{-}Data}$. If the same information is stored in the aggregate network, the access time is $T_{Aggregate\text{-}Data}$. For a point-to-point network (using a crossbar), a table access has a (N-1)/N probability of being a remote access. For a remote access, two point-to-point communications must take place, 2*L, and the remote processor must perform the access, R. For an aggregate network, the processor always has to access the table in the network and this takes two communications, 2* C, and a table lookup that is performed by the network, R. If N accesses are required, then we will conservatively assume they are serialized (i.e. they take N*R time.)

Given:

- R - time required for a table read or write
- L - time required for point-to-point message
- N - number of processors involved in the operation
- P - the % of PEs concurrently accessing shared data

If the table is evenly distributed across N processors and accessed at random, then (N-1)/N accesses will be remote and require two point-to-point messages. Thus:

$$T_{\text{Distributed-Data}} = ((N\text{-}1)/N)\ (2*L\ ) + R \qquad \text{(Eq. 6)}$$

All of the aggregate network accesses will be to the AFU and, thus, remote.

$$T_{\text{Aggregate-Data}} = L + R \qquad \text{(Eq. 7)}$$

If only one processor accesses shared information, then speedup is:

$$Speedup = \frac{2 \times L \times (N-1)/N + R}{L + R} \qquad \text{(Eq. 8)}$$

If P percent of the processors access the same table:

$$Speedup = \frac{2 \times L \times P \times (N-1) + P \times N \times R}{L + P \times N \times R} \qquad \text{(Eq. 9)}$$

Figure 6 shows *Speedup* with respect to N and P given that R=100ns for table access and L=3μs for point-to-point communications. Thus, it can be seen that even if only 10% of the processors want to access shared information that is located on the same processor, it is beneficial to use an aggregate network.

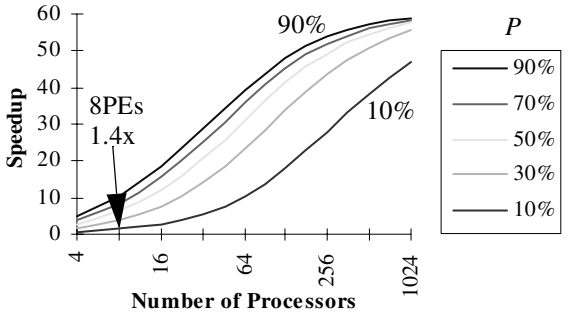

Figure 4. Speedup for Access to Global Data

There are some types of information that must be centrally located and cannot be distributed very easily. Some of these include: shared queues, counters, semaphores, shared file pointers, task scheduling, load balancing, and directory-based coherence mechanisms, among others. For such information, a client-server approach must be taken for point-to-point or multicast networks. Thus, contention becomes a dominant factor because most workstation architectures are not designed to handle such loads. This problem is very similar to theshared table problem, with the exception that 100% of the processors need to access the same tables.

6. Conclusions and Future Work

This paper has introduced three operations that can be executed faster using an aggregate network. The first operation, a *Personalized All-to-All* communication, demonstrated how high-level **communication** operations can be scheduled *by the network* to reduce contention and overhead. Thus, by giving the network information regarding an entire group of communications, advanced scheduling algorithms can be used to facilitate efficient communications. It was also shown that the number of messages that need to be sent can be reduced by sending all of the data within a single message. Thus, communication operations only require each processor to send one message and receive one message.

The second operation showed how an associative **computation** can be executed faster in the network even when a slow serial processor is used. It was also shown that these computations can vary in complexity, and even when the total number of processor cycles required exceeds one thousand, it may still be beneficial to use the network to perform the computation.

The third operation shows how **data** can be stored within the network and accessed more efficiently than if the data were distributed across all of the processors. This shared storage space is limited in size to the buffer capacity of the network, but can be very useful for storing global information and scheduling data. Access to this information using an aggregate network does not require the participation of the other processors. It was shown that even if just 10% of the processors (for N=8 to 1024) access the same table, then aggregate networks have a speedup between 1.4 and 47 over a crossbar implementation.

Thus, the point of this paper is not only to detail three operations, but also to describe how aggregate networks can be used to perform a combination of **communication, computation,** and **global data**.

An SRAM-based reconfigurable hub was constructed using Altera Corporation's 10K Flex in-service-programmable FPGA. The programmable FPGA-based AFU provides for nine processors to be attached (limited by I/O pin count). By using multiple FPGA boards, we expect to be able to network 16, 32, and possibly 48 processors. Preliminary results have shown 5µs process-to-process latency for barrier synchronization and *ReduceAND*. *Min, max, add, all-to-all*, and a number of voting routines are all expected to be completed in the near future.

For more information see:
http://garage.ecn.purdue.edu/~papers/
http://shay.ecn.purdue.edu/~hoare/

References

[AlG94] G. Almasi and A. Gottlieb, *Highly Parallel Computing, Second Edition,* The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.

[BaB95] V. Bala, et.al, "CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computers," *IEEE Trans. on Parallel and Distributed Systems*, Vol 6., No. 2, February 1995, pp. 154-164.

[Bat68] K. Batcher, "Sorting Networks and Their Applications," *Spring Joint Computer Conf.*, 1968, pp. 307-314.

[BiD93] R. Bianchini, S. Dickey, J. Edler, G. Goodman, A. Gottlieb, R. Kenner and J. Wang, "The Ultra III Prototype," *Proc. Parallel Systems Fair,* April 1993, pp. 2-9.

[Cul96] Culler, David E., Active Messages, version 2.0. <http://now.cs.berkeley.edu/AM/active_messages.html>

[DiH96] H. Dietz, R. Hoare and T. Mattox, "A Fine-Grain Parallel Architecture Based on Barrier Synchronization," *Proc. Int'l Conf. on Parallel Processing*, August 1996, Vol. 1, pp. I247-I250.

[GoG83] A. Gottlieb, et. al., "The NYU Ultracomputer Designing a MIMD Shared Memory Parallel Computer," *IEEE Trans. on Computers*, February 1983, pp. 175-189.

[HoD96] R. Hoare, H. Dietz, T. Mattox, and S. Kim, "Bitwise Aggregate Networks," *Proc. Eighth IEEE Symp. on Parallel and Distributed Processing*, October, 1996, pp.306-313.

[LeB96] J. Lee and K. Batcher, "Minimizing Communication of a Recirculating Bitonic Sorting Network," *Proc. 1996 Int'l Conf. on Parallel Processing*, August 1996, pp. I251- I254.

[LoB96] B. Lowekamp and A. Beguelin, *ECO: Efficient Collective Communication Operations for Heterogeneous Networks*, Tech. Report CMU-CS-95-191, CS Department, Carnegie Mellon University, September 1995.

[Mat97] T. Mattox, *Synchronous Aggregate Communication Architecture for MIMD Parallel Processing*, Master's Thesis, School of Electrical and Computer Engineering, Purdue University, 1997.

[PfN85] G. Pfister and V. Norton, "'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *Proc. 1985 Int'l Conf. on Parallel Processing*, August 1985, pp. 790-797.

[Sie90] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Edition,* McGraw-Hill, New York, NY, 1990.

[SnO96] M. Snir et al, *MPI, The Complete Reference*, The MIT Press, Cambridge, Massachusetts, 1996.

[TeS97] D. Tennenhouse, et. al., "A Survey of Active Network Research," *IEEE Communications Magazine*, Vol. 35, No. 1, January 1997, pp.80-86.

[TeW96] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," *Computer Communication Review*, Vol. 26, No. 2, April 1996.

[Wen96] Z. Wen, "Multiway Merging in Parallel," *IEEE Trans. on Parallel and Distributed Systems,* Vol. 7, No. 1, January 1996, pp.11-17.