8th Intl. Conf. on Supercomputing (ICS)
Manchester, England, July 1994

Architecture Implications of High-Speed I/O for Distributed-Memory Computers

Thomas Gross and Peter Steenkiste
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We consider the problem of high-speed I/O for a single application running on multiple nodes of a distributed-memory parallel computer. Our model is that the parallel system is connected to an I/O system that provides the interface between the internal connections of the parallel system and one or more external connections, such as HIPPI links. We identify two primary operations for this I/O system: scattering data from a high speed link across several lower speed links and gathering data from multiple links onto a single high speed link. We show that these core operations are the basis of the I/O system, independent of the relative speeds of the internal and external connections.

We identify several architectural features that are critical for supporting high-speed scatter and gather operations. They include flexible routing methods in the parallel system, low overhead communication, and the ability to support multiple data streams in and out of the memory on the I/O node.

1 Introduction

Parallel systems based on off-the-shelf processors with private memories are an attractive platform for many demanding computations, and as the use of parallel systems has increased, the issue of how to connect such high-performance systems to other systems for "input" and "output" has received considerable attention. Input and output refer to the generic task of getting data into or out of the system. The origin or destination of the data can be another parallel system, a workstation on the network, a storage system, a high-definition display, etc.

The common model is to include in the parallel system "I/O nodes" that provide interfaces to various high-performance subsystems. Such connections can be provided either by a network (e.g., a machine-room, building, or local-area network) or via a direct, dedicated link, as is sometimes done for a parallel file system. Among the possible network choices, HIPPI is currently the most popular one, and most manufacturers of distributed-memory parallel systems either provide or have announced a HIPPI connection (e.g, CM-2 [20], CM-5 [7], iSC/860 [12], NCube2 [14], iWarp [4], Paragon XP/S [11], Maspar [2, 15]). As far as an application on the parallel system is concerned, the exact characteristics of the external links do not matter, and the I/O node provides an appropriate abstraction.

We can think of the I/O nodes as establishing the "periphery" of the parallel system, although in practice, many parallel systems allow I/O nodes to be placed in a variety of positions. However, getting the data from the external world to the I/O node of a parallel system is only part of the problem. In practice, data sets used by the parallel system are distributed, so the I/O architecture of a parallel system has to address the issue of how to get data from the "interior" nodes to the periphery and vice versa.

Clearly, if there are multiple applications running on a parallel system, then the I/O activities of each application can proceed in parallel. This observation provides an easy basis to scale performance: perform multiple independent I/O operations together. However, this solution does not address what we consider to be the tough part of providing I/O for a distributed-memory parallel system: how to provide and scale I/O for a single application that occupies all nodes of the parallel system. Such computations are the primary reason for a parallel system's existence, and while it may be valuable to use a large system for multiple applications during development, in production mode all the nodes are applied towards a single application.

In this paper, we derive a simple model for high-speed I/O on distributed-memory machines. The emphasis is on high throughput: HIPPI or double-wide HIPPI rates, i.e. up to 1.6 Gigabit per second. Fiber channel allows similar bandwidths, and also allows striping across multiple links, thus supporting even higher rates. The aggregate internal interconnection network of most distributed-memory machines easily supports these bandwidths. What makes it hard to achieve these bandwidths in practice is the need for combining/distributing data: converting between a single contiguous data stream (on the network/external side) and data that is distributed according to a specific mapping (inside the parallel system). In networking terms, this is part of the presentation layer in the ISO layering model.

This paper is based in part on our experience with building a HIPPI interface board for the iWarp distributed-memory machine [18]. However, the ideas apply to other architectures.

The research was supported in part by Advanced Research Projects Agency (DOD) monitored by the Space and Naval Warfare Systems Command (SPAWAR) under Contract N00039-87-C-0251, and in part by the Advanced Research Projects Agency/CSTO monitored by the Space and Naval Warfare Systems Command under contract N00039-93-C-0152.
The Nectar Project at Carnegie Mellon acknowledges the use of computational resources provided by the Pittsburgh Supercomputing Center (PSC).

The outline of the paper is as follows. In Section 2 we discuss different architectures for I/O in distributed-memory machines and give examples of existing implementations. In Section 3 we present a model that incorporates the features of the parallel systems and their I/O architectures that are critical for I/O performance. In Section 4 we present measurements for the iWarp distributed-memory system, and we comment on architectural features supporting high-speed I/O.

2 I/O architectures for distributed-memory machines

We start with a brief review of existing I/O architectures for parallel systems. Common to all architectures is the existence of I/O nodes that forward data between the internal nodes (also called compute nodes) and one or more external connections.

2.1 Dimensions

There are many dimensions along which we can classify different I/O architectures for distributed-memory parallel systems:

- I/O topology: I/O can be done over the same links that are used for (internal) node-to-node communication (e.g., CM-5, iWarp, Paragon, NCube2, iSC/860), or over separate links (e.g., CM-2). Intermediate solutions can also be found (e.g., on the MasPar, I/O and global communication share the same links).
- Routing methods: How are the messages routed from the I/O nodes to the interior nodes? Is there a single fixed path (e.g., as in the Intel Delta or Paragon XP/S systems), or are there multiple paths (e.g., in iWarp or the CM-5). How is the route selected (fixed or adaptive), and where is this decision made (source node, network, etc.).
- I/O connectivity: All I/O traffic from a specific interior node must go through a pre-defined I/O node, or a collection of I/O nodes provides services to the interior nodes such that each node in the collection can serve as an agent for a given interior node.
- Bandwidth: Speed of the internal links and speed of the external link(s). In all cases, our measure of speed is the effective bandwidth, which accounts for all overheads (in software and hardware).
- I/O node architecture: This can be the same as for the interior nodes, or different. There are numerous parameters that can be used to compare the I/O nodes with the interior nodes: choice of processor, size, and organization of memory, etc.
- I/O node placement: Can any node of the parallel system be an I/O node (e.g., Paragon XP/S), or is the placement of I/O nodes restricted (e.g., to edge nodes of the base grid in an iWarp torus).

A large number of these dimensions reflect different implementation choices (e.g., the I/O topology and connectivity, placement of I/O nodes) and it is not clear to what extent such dimensions are (or should be!) visible from an application or from a tool that maps an application onto a parallel system. However, there is one dimension that no mapping can ignore: the relative speed of the internal and external links.

2.2 Ratio of internal and external link bandwidth

The bandwidth of the internal and external links are driven by separate sets of constraints. These bandwidths are determined largely independently, and as a result, the two bandwidths are typically different.

There are several reasons why the bandwidth of a single internal link may be lower than the external bandwidth, even if both links are designed at the same time:

- The number of wires used by an internal link is less than the number of wires used by an external link, for example because of higher connectivity. That is, the external link is $K \times I$ bits wide, with I the number of wires for an internal link. Even if both links operate at the same speed, the external link's bandwidth is higher.
- A different technology is used for internal and external links. Power budgets, space constraints for connectors, heating concerns, and cost may motivate the designer to chose a technology for the interior links of a (scalable) parallel system that is different from the technology of the external links (e.g. metal versus fiber).
- The internal links are designed to provide low latency, at the cost of a reduced (peak) bandwidth. For applications with small messages on average, it may be more important to experience low startup overheads than high peak bandwidth.

On most systems today, the internal bandwidth is considerably lower than the external bandwidth provided by a HIPPI interface (100 MBytes/sec): iWarp (40 MBytes/sec peak), CM-5 (5 - 20 MBytes/sec), and iSC/860 (2.8 MBytes/second). Note that this is the single-link bandwidth. Some systems (e.g., iWarp, CM-5) allow multiple paths to be used in parallel and can therefore support a higher aggregate internal bandwidth between two points in the parallel system.

However, there exist also good reasons why the internal link might be faster than the external links:

- External links are typically based on a standard so that the system can communicate with independently developed devices. Standards take a while to develop, and standardized links often lag behind in performance compared with what the technology can support at a given point in time. Internal links do not have to follow a standard and can use more aggressive technology.
- Internal links are typically much shorter and run in a more controlled environment than external links. External links also run more heavy weight protocols (e.g. TCP/IP) than internal links because they have to recover in software from hardware errors. Both factors make it easier to sustain high-bandwidth communication over internal links than over external ones.

For example, the Paragon XP/S system and the Cray T3D [1] have internal bandwidths that are is higher than the standard HIPPI bandwidth.

The above discussion focussed on the peak bandwidth of single internal and external links. What actually matters is the bandwidth that can be achieved in practice, and this is not just determined by the bandwidth of a single link. On one hand, various resource constraints lower the effective link bandwidth, For example, some of the systems that have HIPPI channels are only able to sustain throughput that is a small fraction of its peak bandwidth, effectively creating a situation where the external link speed is lower than the internal speed, either all the time or just for some block sizes. On the other hand, most parallel systems allow a number of I/O nodes to be added to a system, so that there can be multiple network connections, with a corresponding increase in bandwidth.

2.3 Impact of the application requirements on I/O organization

The reason to provide I/O is to support applications on the parallel system, and ultimately, the requirements of these applications drive the system design. The objective for a system designer is to build a balanced system that supports the applications. Our definition of the term "system design" includes the use of both appropriate hardware and software. One of the approaches to a bandwidth discrepancy between internal and external links is to structure the communication software so that it can use multiple links in parallel.

For a system with certain internal and external link bandwidth, we can use the I/O requirements of the application to classify the activity of the I/O nodes into several categories. We consider first the case of "input" or "receive" operations, i.e. the application on the parallel system obtains data from the outside world.

- If the application requirements are below both the external and internal bandwidth, the application can handle the data in the nodes of the parallel system, and the role of the I/O node is limited to forwarding the data from an external link to an internal link. This is simple case is not considered further in this paper.
- If $external \geq application > internal$, then the I/O node must *scatter* data from the external link onto multiple internal links in the system. Figure 1 depicts this situation. Note that the application on the parallel system must be structured so that the single data set delivered by the I/O system ("1234" in Figure 1) is required in this order by the nodes. In this example, the data set is broken into individual blocks ("1", "2", "3", "4"). There are other distributions that are commonly accepted by compilers (e.g., cyclic and block-cyclic)[9], and these distributions induce different scatter operations.
- If $internal \geq application > external$, then the I/O node (or the I/O system) must *gather* data from multiple external links and forward these data over a single internal link to the requesting node. Figure 2 depicts this situation. The same comments about distributions apply; depending on the data distribution chosen by the compiler, the external data sets must be combined in different ways to meet the requirements of the application.

Scattering and gathering are crucial activities of the I/O nodes. All I/O node designs that we are aware of include some form of dual-ported memory to build the interface between the network (external links) and the parallel system (internal links). The design of these memories should take into account these operations, and allow fast access for the access patterns of scatter and gather operations.

When we consider the dual case, i.e., the application on the parallel system produces output (also known as the "transmit" case from the network's point of view), we discover that the same operations must be supported by the I/O node. If the external link bandwidth is higher than the internal link bandwidth, the I/O node performs a *gather* operation for sending: data arrive via multiple streams and are merged on the I/O node into a single stream for the faster external link. And if the internal link is faster, an output operation from the parallel system to the I/O periphery involves a scatter operations as the data arriving on an internal link are passed on to multiple external links. Scatter and gather operations over internal or external links are basic requirements for I/O.

Some applications have I/O requirements that exceed the bandwidth of both the internal and external links. If the routing methods supported by the parallel system and the network allow data to travel between the I/O node and the target node(s) in the parallel system (or the node(s) on the network) via different routes, then this flexibility provides another option to deliver the required bandwidth.

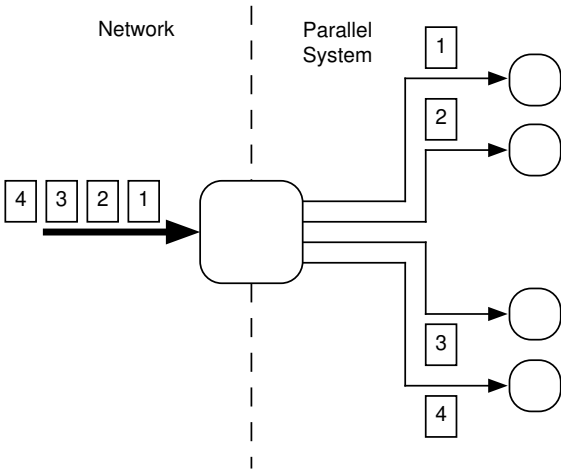

Figure 1: Scattering data as part of an input operation

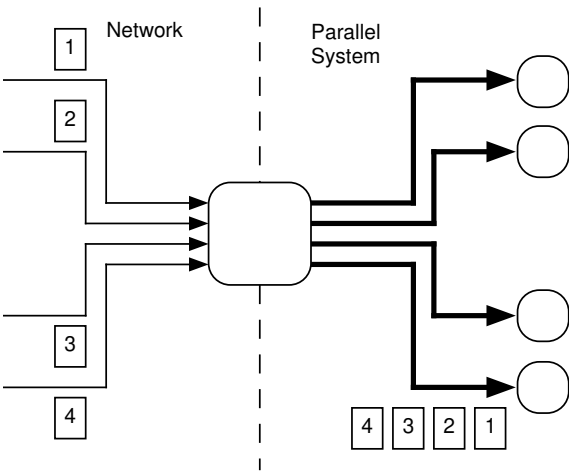

Figure 2: Gathering data as part of an input operation

- If $application > external \wedge application > internal$, then the I/O system has to take data from multiple external links and distribute it over multiple internal links. That is, the I/O node performs combined scatter/gather operation. This is the case independent from the relationship between the internal and external bandwidths. Figure 3 depicts a simple example.

Systems like iWarp and the CM-5 provide multiple paths between nodes in the system, and these systems should be able to capitalize on this property to achieve higher effective I/O bandwidths. Systems like the Paragon or the Delta machine support only a simple X-Y router that provides for exactly one path between any pair of nodes. If the I/O node has to distribute data over multiple nodes in the same routing dimension, then the routing restrictions keep some links to the I/O system underutilized, whereas other links are oversubscribed, as illustrated by Figure 4. In this case, no parallelism in transferring the data inside the parallel system can be exploited.

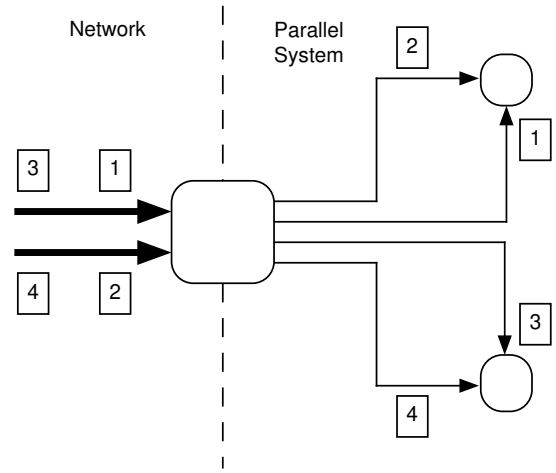

Figure 3: Combined scatter/gather as part of an input operation

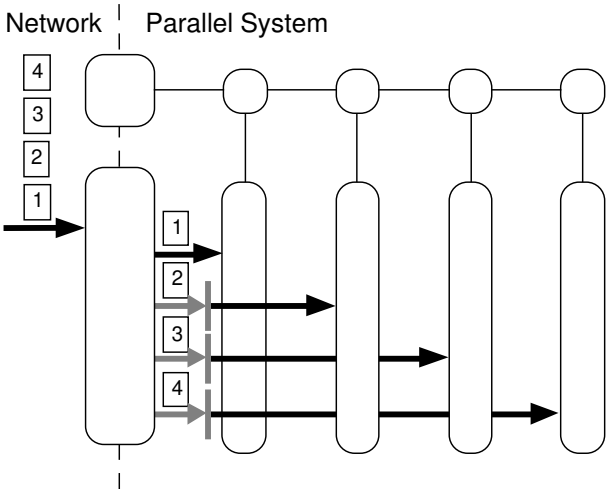

Figure 4: Oversubscribed and idle interior links with X-Y routing

2.4 Architectural issues for I/O nodes

Scatter and gather operations stress the local memory system of the I/O node since the techniques that are used to provide a high memory bandwidth economically are rendered ineffective by the addressing patterns of scatter and gather. In this section we discuss two issues: the memory organization and the choice of processor.

The interface node must be able to combine multiple data streams coming from different links into a single serialized stream, since the internal link bandwidth is often lower than the network bandwidth. This has an impact on the memory design, which can be based either on static RAM (small, but always fast) or dynamic or video RAM (large, but access cost non-uniform). Static RAM allows memory accesses to be interleaved with almost no restrictions, but because of the cost and real estate constraints, the memory is likely to be small, so the memory must be managed very carefully. With VRAM or DRAM, it is possible to build a much larger memory, but since some access patterns result in poor bandwidth, accesses to the memory must be managed carefully.

The interface node has to be linked to the parallel system's interconnect. How this is done is largely determined by the node architecture.

- If there is a single component that includes both the processor and the interconnect switching element (e.g., iWarp, NCube), use of this component is the most cost-effective way to tie a node into the system's interconnect. In such a system, data flows from the parallel system through that component to the memory of the I/O node and then on to the network. The architecture of the I/O node must be largely the same as for the compute nodes, although there could be additional processors to help in managing the I/O system. The memory hierarchy is the most critical issue, since it must be possible to interleave multiple data streams and memory accesses by the CPU of the I/O node.

- If the processor is separate from the switching element (e.g., iSC/860, Delta, Paragon XP/S), communication in the system is usually memory-based (all data extracted from the interconnect is first placed in memory). The I/O node can use a processor other than the one used as compute node, as long as this processor is able to implement the same low-level interconnect access functions (e.g., reading and writing a FIFO, servicing interrupts at a specific rate, etc.). The biggest impact of the choice of processor is that there is now more freedom in defining the memory hierarchy.

One some systems (e.g., Maspar-1), the I/O "node" is just a memory, and send/receive on the I/O node means read/write. The advantage is that the send/receive overhead is very low, which is important in practice. In general, however, this is an expensive solution, and the mechanism with which nodes access the I/O node is different from inter-node communication.

2.5 Internal file I/O versus network I/O

Two major forms of I/O used on distributed-memory computers are disk/file I/O and network I/O. In this section we look at their characteristics and relationship in large distributed-memory systems.

There is a lot of similarity between the problems of network I/O and disk I/O on a distributed-memory machine, but there are also some significant differences:

- High performance systems often have a HIPPI connection to a HIPPI network, so network I/O usually falls into the case where external bandwidth is higher than internal bandwidth.

- Disk systems are typically slower than the internal link bandwidth. However, on most high-performance distributed-memory machines, files are distributed across multiple disks that are attached to different nodes in the array [16, 8]. In this case, disk I/O requires combined scatter/gather operations.

- With disk I/O it is possible to write data to a file on a disk out of order, so assuming this option is supported, data does not necessarily have to be sent to the disk nodes in exactly the right order. The situation for networks is quite different in that the parallel system must meet the order expected by the outside.

An interesting issue is the relative location in the system of nodes supporting disk I/O and networks [17]. Figure 5 shows two extreme cases. In the first case, both the disk and network interfaces are part of a single "I/O space", and it is possible to access files on the disk system without going through the

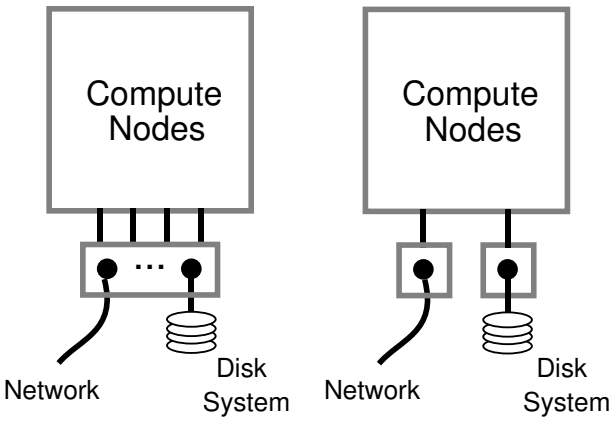

Figure 5: Organizations of I/O space

parallel system. If the disk or network interface is distributed across multiple I/O nodes, then these nodes can also communicate without involving the parallel system. An example of this organization is provided by the CM-2. The other extreme is to have the network and disk subsystem connected to the parallel system independently; any communication between the two has to go through the array. Note that very often systems can be configured either way.

Which organization is the most attractive depends primarily on two factors: the typical usage of the system and the relative bandwidth of the internal and external links:

- Relative bandwidths: If different I/O nodes are part of a single I/O space, some of the links of these I/O nodes must be used for communication inside the subspace, so the total bandwidth between the I/O nodes and the parallel system is reduced. If the external bandwidths are much higher than the internal link bandwidths, then it is possible to achieve higher throughputs by connecting each interface into the parallel system independently.
- Usage: A single I/O subspace is attractive if a lot of the communication between the array and the outside world happens through the file system. An example is an application that performs file I/O; I/O with the outside world is done by having other systems access the file system of the distributed-memory machine [13]. In other cases it is important that applications on the parallel system have high-speed access directly to the network. Examples are when the distributed-memory machine is used as a node in a loosely-coupled distributed system, or if it uses a network file system.

However, if we consider the operations of the I/O node, without regard of the type of data being transferred, we observe that scatter and gather are again the primary operations that must be supported by the I/O node. In this paper, we use network I/O in our examples, but the results are more general.

3 I/O model

I/O can be viewed as a pipeline with the following stages (for output): compute nodes (array), I/O network, I/O node, and external link. These stages can be combined in a simple model, and as a first approximation we can determine the total data throughput by identifying the slowest element in the pipe. The actual data flow can be even lower than the throughput of the slowest stage, because the throughput of the stages is not necessarily constant, and the buffering between the stages is limited, so a stage can sometimes be idle. The throughput can however never be faster.

In the following sections we present a simple model for the major components data has to travel through when it is sent from the parallel system to the outside world, and we then put these pieces together for a generic distributed-memory system. These models provide a basis for a discussion of which architectural features are most critical for I/O performance in a practical system.

3.1 Compute and I/O nodes

The compute and I/O nodes have the same parameters in the I/O model, although the function of the I/O node is more complex since it must be able to communicate with the outside world and function as a router between the external world and the internal interconnect. The two relevant parameters the effective I/O bandwidth for accessing a single link, and the degree of parallelism in accessing multiple links:

- Communication performance of the node over a single link. Communication cost is often modeled as a constant overhead per send and receive, plus an overhead proportional to the message length. We call L the maximum sustainable link bandwidth and O the constant overhead for each send or receive.
- Degree of communication parallelism. How well do the nodes support simultaneous communication over multiple internal links? The upper limit is the number of links into the interconnect, but the number can be lower in practice because of other bottlenecks: the hardware resources may restrict the use of links (e.g. only one DMA engine, or only one bus to memory), or multiple links can be operated in parallel but incur a performance degradation.

3.2 Connecting the I/O nodes to the parallel system

The topology and link speed of the parallel system clearly have an impact on the bandwidth of both the internal interconnect (important for getting data to the I/O network) and on the bandwidth between the parallel system of N nodes and the I/O node(s). In the most general case, the I/O space consists of more than one I/O node, and these I/O nodes have connections to other I/O nodes and to (compute) nodes in the parallel system. Depending on the topology of the system, we have the following peak bandwidths between a parallel system with N nodes and a single I/O node:

- Mesh with a single layer of I/O nodes at the boundary of the array. The maximum bandwidth is L over the direct connection with the array plus $2 \times L$ indirectly through the I/O network for a total maximum bandwidth of $3 \times L$ (Figure 6). The maximum bandwidth for traffic inside the I/O space is similar: L directly over the I/O network plus $2 \times L$ indirectly through the internal network. If we have more than one layer of I/O cells, the total maximum bandwidth goes up to $4 \times L$ in both cases, but the cost is obviously also higher, since extra nodes are needed. Of course, routing restrictions may impose a lower bound if not all paths can be taken to deliver data to an I/O node.
- Torus (as in iWarp). I/O nodes have a bandwidth of $4 \times L$ to and from the array, either directly (each I/O node is only connected to the internal interconnect), or in part indirectly (if the I/O nodes are interconnected).
- Hypercube. Each node has a fixed number of network connections, $log(N)$ of which are used to build a hypercube. One way of setting up I/O is to reserve one link for

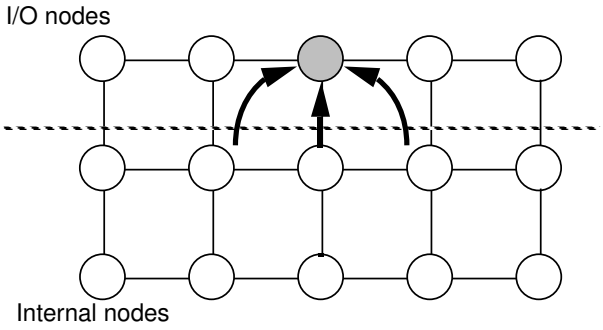

Figure 6: Access to I/O node in a mesh

I/O. The total system consists of one internal hypercube and one or more smaller I/O hypercubes, which have each of their Q nodes connected to the internal interconnect using the reserved link. The maximum bandwidth between a single I/O node and the internal compute nodes is L directly and $L \times log(Q)$ indirectly.

- Distinct interconnects for communication and I/O. If a separate interconnect is used for I/O, it typically consists of a star: each group of A array nodes is connected to a single location through an I/O link, so the available bandwidth is $L_{I/O} \times \frac{N}{A}$.

3.3 Modeling the complete I/O pipeline

Our goal is to express throughput T as a function of a small set of parameters that characterize the distributed-memory system and the application. The throughput T is expressed relative to the maximum sustainable link speed L; L is expressed in bytes/second. Let us first summarize the parameters of the model:

- the sustainable link bandwidth L: the minimum of the raw link bandwidth, the node memory bandwidth available for communication, and the bandwidth of the datapath between the node memory and the interconnect. (Contention inside the parallel system is either ignored or factored into the link bandwidth.)
- the communication overhead O in seconds: this is the per-packet overhead associated with sending or receiving a packet on the internal interconnect.
- the number of parallel paths n between the parallel system and a single I/O node. As discussed above, this number depends on the architecture of the I/O space.
- the block size B in bytes: size of contiguous blocks that are allocated to the same compute node. The block size depends both on the total data size and on how the data is mapped onto the parallel system [6][18][3]. We call b the time it takes to send a block over a link at the full link bandwidth, i.e. $b = B/L$, so the total cost of sending a packet of size B is $O + b$.
- the memory bandwidth M available for communication: given the definition of L, $M > L$ in the nodes, so L is the critical parameter with respect to communication over a single link. On the network interface we might have to combine multiple data streams, so M can become the critical performance parameter. We will define $m = M/L$ as the number of data streams that can be supported by the memory on the network interface. We call MSP the minimum of m, the number of streams, and n, the number of paths.
- the maximum sustainable bandwidth W over the external network: W is determined by the peak bandwidth of the network and the implementation of the I/O node. It is largely independent of the architectural features of the distributed-memory system.

In most systems today the internal interconnect bandwidth is lower than the high-speed external network bandwidth, so we focus on the case of gathering data for output. We express the throughput T for the case that different compute nodes in the parallel system send data to the I/O node over independent paths, and the I/O node combines the data streams and passes the data on to the external network. The throughput of each of the stages of the I/O pipeline is:

- Compute nodes: n nodes send n independent streams of packets of size B to the I/O node. The throughput for each stream is $B/(O + b)$. To focus on the overall performance, we want to express this throughput relative to the thoughput of a single link (L), so the total relative throughput between the parallel system and the I/O node is

$$T_{CN} = n \times (\frac{\frac{B}{O+b}}{L}) = n \times \frac{b}{O+b}$$

- I/O network: The I/O network includes all the links over which data travel from the compute nodes to the I/O node. The throughput is by definition $n \times L$, so the relative throughput is simply

$$T_N = n$$

- I/O node: The I/O node combines n streams in memory. Each stream consists of packets of size B, and the bandwidth into memory is $MSP \times L$. The throughput is $B/(O + (B/(MSP \times L)))$, and the relative throughput is

$$T_I = \frac{b}{O + \frac{b}{MSP}}$$

- External network: By definition, the throughput is W, so

$$T_E = \frac{W}{L}$$

The most significant cost not included the model is the cost of synchronization: in practice, compute nodes cannot just forward blocks to the I/O node, but they will have to coordinate their transfers. This cost depends strongly on the architecture of the distributed-memory system.

Figure 7 and Figure 8.(a)-(c) show the relative throughput T as a function of the message transfer time b for the first three stages of the pipeline. The message transfer time ranges from 1 microsecond to 10 milliseconds. The X axis of Figure 7 also shows the corresponding message sizes for two link speeds. For $L = 40$ MBytes/sec, we have block sizes B between 40 Bytes and 400 KBytes, and for 100 MBytes/sec, block sizes range from 1 to 1000 KBytes. Different figures depict results for different values of n and m, and each figure shows the throughput for different message overheads O (0.1, 1, 10 and 100 microseconds).

Figure 7 shows the thoughput for the case of a single link between the parallel system and the I/O node. Assuming the message overhead on a compute node is the same as on the I/O

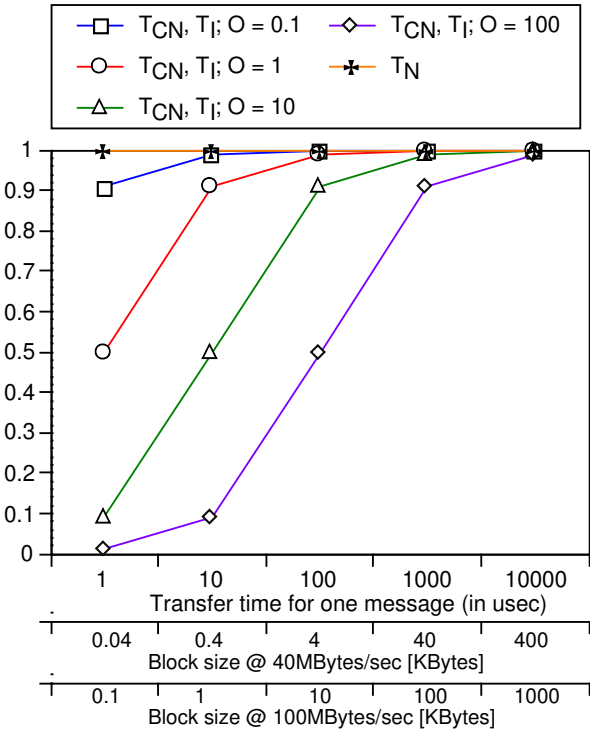

Figure 7: Throughput for $n = 1$ and $m = 1$, relative to link bandwidth L

node, the relative throughput of the compute node stage (T_{CN}) is the same as the T_I for the I/O node. (See lines labeled T_{CN} and T_I.) Figure 7 depicts clearly the importance of the message overhead: for small block sizes, the message overhead must be very low to get good throughput.

Figure 8.(a) depicts the case that 4 links are used between the compute nodes and the I/O node, and the memory system on the I/O node can sustain all 4 links. The relative throughput is now potentially four times the link bandwidth, so parallel paths to the network interface are important. If the message overheads are the same on the I/O and compute nodes, the I/O node now becomes the bottleneck since it has to deal with more messages. Low message overheads remain important.

Figures 8.(b) and 8.(c) show variations of the above case (4 parallel paths to the network interface), but this time the memory system on the I/O node can only support respectively 3 and 2 of the links, either because the memory bandwidth is limited or the memory cannot efficiently interleave the data coming from the different links, or because of routing restrictions. We now see that the bandwidth drops significantly compared with Figure 8.(a).

In summary, the model clearly indicates that the following architectural features of distributed-memory machines have a serious impact on I/O performance:

1. Small messages must be handled efficiently, since the distribution of data over the private memories of the nodes often results in small blocks of data [6, 10].

2. The routing system must provide enough flexibility to allow the user or compiler to control the flow of traffic. This determines how much communication can occur in parallel.

3. The ability for the I/O node to handle high bandwidth communication, interleaving data streams, and low overhead communication is critical.

4 Performance results and discussion

We implemented in cooperation with NSC a HIPPI interface for iWarp [19], a parallel system developed jointly between Carnegie Mellon and Intel Corporation. In this section, we present performance data for this system and try to relate the reality and experiences of this I/O system to the model of the previous section. An iWarp system is organized as a 2-dimensional torus. Each node is connected to its four neighbors by 2 unidirectional links with a peak bandwidth of 40 MBytes/sec. Activity on all links can proceed in parallel, so each node has a peak communication bandwidth of 320 MBytes/sec. The local memory bandwidth of each node is 160 MBytes/sec [4].

4.1 Measurements

In this section, we present some measurements to illustrate how the effective bandwidth varies as a function of the message size and how multiple links can be used to provide an effective bandwidth that is higher than the individual link bandwidth. We concentrate on the transmit case of an I/O stream to single device on the network (e.g., sending an image from the iWarp system to a HIPPI framebuffer or the Cray C90 at the Pittsburgh Supercomputing Center).

The curves in Figure 9 show the throughput between the iWarp array and the network interface for different messages sizes and with the use of different numbers of links between the I/O node and the compute nodes, relative to the bandwidth of an individual link. We realize about 60 MBytes/sec with 2 links, and up to 145 MBytes/sec with 4 links, well above the bandwidth of the external HIPPI network. Figure 9 shows that using multiple communication paths between compute and I/O nodes allows us to achieve a very high throughput, much higher than the individual link bandwidth. Furthermore, we see that that the iWarp component on the I/O node is effective in gathering data from multiple internal communication links into memory, a function that is important when supporting a total throughput that is higher than the internal link speed. Similar results apply to scattering data across multiple pathways. Note that in the case of 2 and 4 links, the size of the messages transferred over each link is respectively one half and one fourth of the message size shown on the X-axis. The results of Figure 9 also show that the iWarp communication system supports high bandwidth communication even for small message sizes.

We now look at how the iWarp measurements fit the model presented in the previous section, and we discuss in more detail the architectural requirements for efficient scatter and gather operations in the context of the iWarp measurements.

4.2 Matching the model

We find a good match between the measurements of Figure 9 and the model of the previous section, for a message overhead O of 4 microseconds and an MSP of 0.88, 1.6 and 3.7 for 1, 2 and 4 ports respectively. The histogram in Figure 9 gives the percentage error between the measurements and the model for these values. The MSP values are less than the ideal values (1, 2 and 4) because of memory access conflicts with the instruction and data accesses by the CPU.

4.3 Inexpensive communication

The important issue is that communication is efficient and incurs a low latency. There are two factors that contribute to latency: the actual wire (or transmit) latency, and the software

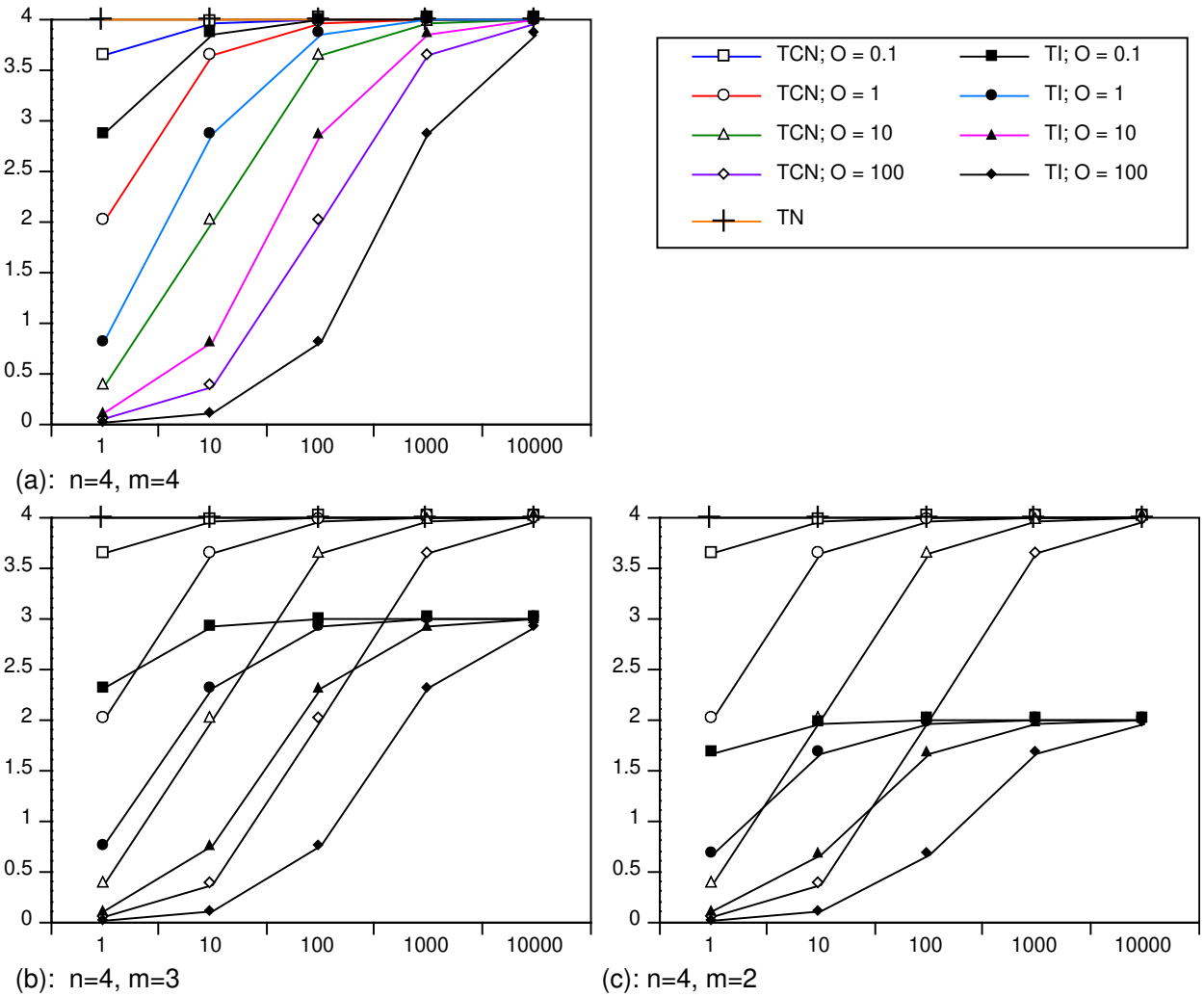

Figure 8: Relative throughput as a function of message transfer time, for different values of n and m

overheads. The first one is tolerable in a reasonably designed system, and it is in practice the software overheads of sending and receiving messages that determine the (lack of) performance. Since iWarp supports systolic as well as memory-memory communication (in memory communication a complete message is stored in or retrieved from memory for sending and receiving; in systolic communication the processor reads/writes individual words directly) [5], we experimented with the use of both communication paradigms to support the network I/O discussed in this paper. In summary, we had the following experience:

- We often use systolic communication for fine grain reshuffling of data since it involves very little overhead [6].
- However, memory communication is also important since it allows a decoupling of the various send and receive operations on the I/O node. Specificly, the DMA-engines are useful since they allow communication to occur in the background (for large blocks of data).
- Having both mechanisms available on iWarp component proved important since it offered the software system a choice between low-overhead, tightly coupled communication (systolic, for small blocks) and background communication (memory-communication, via DMA engines, for large blocks).

On architectures that do not support systolic communication, minimizing the constant per-message overhead has a very large impact, since this overhead determines how well a compute node can get a small block of data to the I/O node.

4.4 Flexible routing

It is important that the routing system allows the application to route packets in a flexible way so that it is possible to set up parallel paths between the array and the I/O node. In iWarp this can be achieved by setting up (via software) direct connections between non-adjacent nodes[4]. The route of such a connection is determined by the application and takes congestion in the interconnect into account. If a fixed routing strategy is followed, i.e. first X then Y, then this strategy limits the number of paths to one even if multiple physical links are available.

Flexible routing is also important to avoid communication bottlenecks. Parallel routes should not use the same links in the

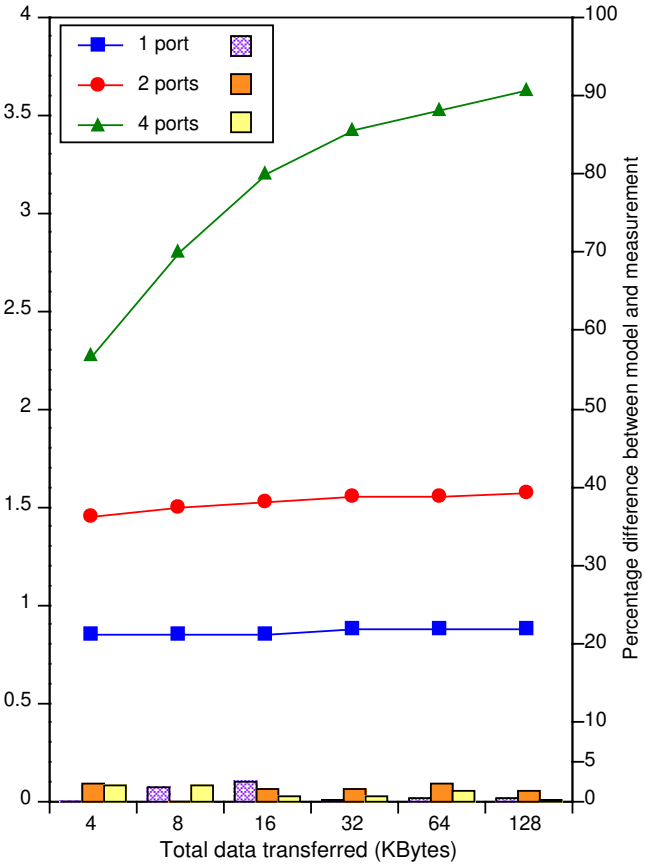

Figure 9: Relative bandwidth between iWarp array and I/O node

interconnect, and they should also not make use of any links that are already heavily used by the application. Note that this implies that the application has to be involved in setting up the paths to the I/O node since only the application has information on the link utilization.

4.5 Locality of memory access

Scatter and gather means that data from different sources will have to placed contiguously in memory so that the data can be streamed over the external network. This raises the problem of locality of memory accesses, both on the internal compute node that is the final destination of a block of data, as well as on the I/O node. We have to consider two major cases, illustrated by Figures 10 and 11, based on how the CPU of the node interfaces to the communication system. These figures show a node with two links to the interconnect, as well as some FIFOs that are part of any realistic communication system. In the discussion, we focus on the I/O node, although the same issue must be addressed for each compute node.

- The parallel system interconnect is directly accessed by the CPU, i.e. there is a direct path from the CPU to the communication system that does not share any resources with other links that this CPU is connected to (see Figure 10). One way to think about this is that from the perspective of the CPU, the interprocessor communication system occupies its own ”I/O space”. A systolic processor provides one example of such an arrangement; another is a system with a separate local memory bus and an I/O bus. In this case, data is moved under explicit program control to memory. As a result, memory accesses from the CPU (for cache fills, updates of local variables, etc.) and accesses related to communication (data to be transferred for different streams) interleave at a relatively fine grain.

  One option to address absence of locality is to use static RAM in the design of the memory system so that the poor access locality will not degrade memory bandwidth; this is the implementation that was chosen for the iWarp HIPPI interface. Another, more complex option, is to have the CPU access memory through a cache and to use write and read buffers to turn transfers between the internal interconnect and the memory into burst transfers. This would allow the use of larger and cheaper dynamic RAMs, but the cost is a more complex design.

- The CPU is an off the shelf chip, and the parallel system interconnect is attached to the memory bus (see Figure 11). This scenario is close to what one would find on a workstation. A designer still has to worry about locality of access in memory, but solutions similar to those used in workstations are applicable, e.g. the use of DMA engines that use large burst sizes.

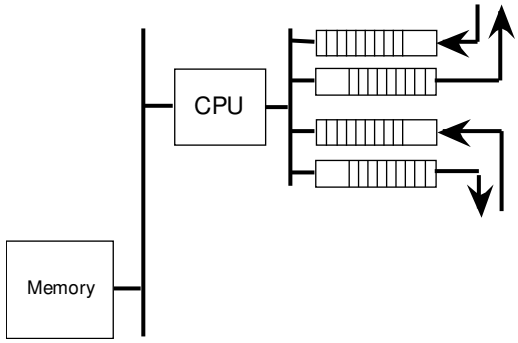

Figure 10: Node organization: separate paths for communication system and memory system access

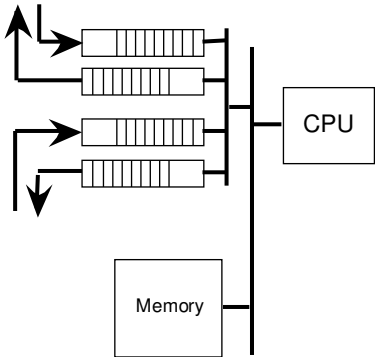

Figure 11: Node organization: communication system and memory system access share path

5 Conclusions

The effective bandwidth available for I/O from a parallel system to and from the outside world is determined by a number of

factors: the bandwidth of the internal and external links, the organization of the I/O node that forms the link between internal interconnect and external network, the overheads imposed by the software system, and the characteristics of the application.

I/O nodes move data from the parallel system to the network (or dedicated I/O system). Such moves involve memory and usually pass through a dual (or multi-) ported memory system. When we consider the activities of such an I/O node, we discover that scatter and gather are the core operations. Any attempt by the I/O node to compensate for a difference in internal and external link bandwidths results in such scatter and gather operations.

The distribution of data onto the compute nodes of a parallel system is often managed by a compiler, and the characteristics of the application, specifically the block size used in the data mapping, has an impact on I/O performance. As our model predicts (and is confirmed by measurements on the iWarp system), small block sizes result in poorer link utilization. Therefore, any support the parallel system can offer to reduce the message overhead has a direct impact on the effective I/O bandwidth.

Finally, routing methods that provide enough flexibility to use multiple paths between a pair of nodes have a positive impact on overall performance. By using alternate paths between the I/O node and a compute node, the effective bandwidth for I/O is increased. To realize this benefit, the I/O node (including its memory system) must be designed to support multiple incoming and outgoing streams when transferring data between the network and the parallel system.

References

[1] Don Adams. *Cray T3D system architecture overview*. Cray Research Inc., September 1993. Revision 1.C.

[2] Tom Blank. The Maspar MP-1 architecture. In *IEEE Compcon Spring 1990*, pages 20–24, San Francisco, February/March 1990. IEEE, IEEE Computer Society Press.

[3] Rajesh Bordawekar, Juan DelRosario, and Alok Choudhary. Design and evaluation of primitives for parallel I/O. In *Proceedings of Supercomputing '93*, pages 452–461, Oregon, November 1993. ACM/IEEE.

[4] Shekhar Borkar, Robert Cohn, George Cox, Sha Gleason, Thomas Gross, H. T. Kung, Monica Lam, Brian Moore, Craig Peterson, John Pieper, Linda Rankin, P. S. Tseng, Jim Sutton, John Urbanski, and Jon Webb. iWarp: An integrated solution to high-speed parallel computing. In *Proceedings of the 1988 International Conference on Supercomputing*, pages 330–339, Orlando, Florida, November 1988. IEEE Computer Society and ACM SIGARCH.

[5] Shekhar Borkar, Robert Cohn, George Cox, Thomas Gross, H.T. Kung, Monica Lam, Margie Levine, Brian Moore, Wire Moore, Craig Peterson, Jim Susman, Jim Sutton, John Urbanski, and Jon Webb. Supporting systolic and memory communication in iWarp. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 70–81, Seattle, May 1990. ACM/IEEE. Also published as CMU Technical Report CMU-CS-90-197.

[6] Claudson Bornstein and Peter Steenkiste. Data reshuffling in support of fast i/o for distributed memory machines. In preparation, 1993.

[7] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*. Thinking Machines Corporation, 1991.

[8] Erik DeBenedictis and Peter Madams. nCUBE's parallel I/O with UNIX compatibility. In *Proceedings of the Sixth Distributed Memory Computing Conference*, pages 270–277. IEEE, April 1991.

[9] S. Hiranandani, K. Kennedy, and C. W. Tseng. Compiling Fortran D for MIMD distributed-memory machines. *Communications of the ACM*, 35(8):66–80, August 1992.

[10] R.W. Hockney and Jesshope C.R. *Parallel Computers*. Adam Hilger Ltd., Bristol, U.K., 1981.

[11] Intel Corporation. *Paragon X/PS Product Overview*, March 1991.

[12] Sigurd L. Lillevik. The Touchstone 30 gigaflop Delta prototype. In *Proceedings of the Sixth Distributed Memory Computing Conference*, pages 671–677. IEEE, April 1991.

[13] Susan LoVerso, Marshall Isman, Andy Nanopoulos, William Nesheim, Ewan D. Milne, and Richard Wheeler. sfs: A parallel file system for the CM-5. In *Proceedings of the Summer 1993 USENIX Conference*, pages 291–305, Cincinnati, Ohio, June 1993. USENIX.

[14] nCUBE Corp. *nCUBE2: Technical Overview*. nCUBE Corporation, Foster City, CA., 1992.

[15] John R. Nickolls. The design of the Maspar MP-1: A cost effective massively parallel computer. In *IEEE Compcon Spring 1990*, pages 25–28, San Francisco, February/March 1990. IEEE Computer Society Press.

[16] Paul Pierce. A concurrent file system for a highly parallel mass storage subsystem. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, volume 1, pages 155–160, California, March 1989.

[17] T. W. Pratt, J. C. French, P. M. Dickens, and S. A. Janet Jr. A comparison of the architecture and performance of two parallel file systems. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, volume 1, pages 161–166, California, March 1989.

[18] Peter Steenkiste, Michael Hemy, Todd Mummert, and Brian Zill. Architecture and evaluation of a high-speed networking subsystem for distributed-memory systems. In *Proceedings of the 21th Annual International Symposium on Computer Architecture*. IEEE, May 1994.

[19] Peter A. Steenkiste, Brian D. Zill, H.T. Kung, Steven J. Schlick, Jim Hughes, Bob Kowalski, and John Mullaney. A host interface architecture for high-speed networks. In *Proceedings of the 4th IFIP Conference on High Performance Networks*, pages A3 1–16, Liege, Belgium, December 1992. IFIP, Elsevier.

[20] Lewis W. Tucker and George G. Robertson. Architecture and applications of the Connection Machine. *IEEE Computer*, 21(8):26–38, August 1988.