Interconnection Networks in Distributed Shared Memory Multiprocessors

By: Angel Dominguez

M.S. of Electrical and Computer Engineering
Scholarly Paper
April 22, 2001

1. Introduction

The past decade has shown explosive growth in the performance and capability of computer systems, driven by advances in VLSI technology that have allowed an ever greater number of components to fit onto a silicon chip. Computer architectures have evolved to harness this growth and translate it into increasingly powerful computer systems. To take advantage of the larger volume of resources available to designers, parallel computer architectures have been developed to better utilize multiple resources in concert. The use of parallel design appears in various forms in current computer systems and includes such techniques as pipelining, superscalar processors, VLIW processors, and multiprocessor systems. As uniprocessors are reaching their performance limits, it is only natural that the use of multiprocessor parallel systems has become popular for large and complex applications entailing enormous amounts of computation.

In essence, a parallel computer is a collection of processing elements that communicate and cooperate to solve large problems fast [1]. Given this definition, it is reasonable to view parallel architecture as the extension of conventional computer architecture to address the issues of communication and cooperation among the processing elements. The design of this communication architecture specifies both the basic communication and synchronization operations required at the hardware/software interface, as well as how to implement these operations in high performance communication hardware. The bulk of parallel computer design has thus fallen into two large groups according to their programming models for communication support. These consist of the message passing approach and the shared memory approach.

In the message passing machine, each processor has its own memory modules, and all the processors are connected through an interconnection network. A processor cannot directly access the data on another processor's memory, and instead sends and receives messages through library functions to exchange data with another processor. On the other hand, shared memory machines share a unique, global memory address space that is accessible to all of the processors. Communication between processors is then done by writing and reading from the same global memory locations, similar to the traditional multithreaded approach in uniprocessors. Despite their differences in communication structure, both types of parallel systems rely heavily upon the interconnection network that links their processors, and numerous designs have been proposed and implemented to achieve optimal communication speeds for these systems.

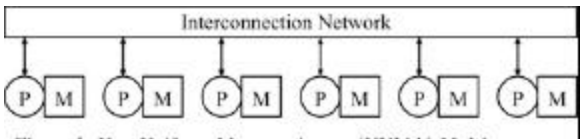

Figure 1 Non-Uniform Memory Access (NUMA) Multiprocessor

Since the field of parallel architecture is such a large one, only distributed shared memory (DSM) multiprocessors will be focused on in this paper, with the emphasis being placed on the typical interconnection networks employed in this class of systems. Distributed shared memory machines are one of the prevalent types of shared memory machines in use today, and generally fall into the category of Cache-Coherent Non-Uniform Memory Access (CC-NUMA) architectures, which are illustrated in Figure 1. The processors in these machines each possess their own memory modules, accessible by all processors, and private caches, which must be kept coherent with respect to all remote operations on cached data blocks. In order to provide programmers of these systems with a view of a globally shared memory space, a great deal of communication is required between processors to transfer data requests between processors and remote memory modules, and to ensure cache coherence in the system. The effectiveness of the shared memory approach then depends on the latency incurred on memory accesses, as well as the bandwidth of data transfer that can be supported. Indeed, the performance of the interconnection network used can be one of the main bottlenecks for a DSM [2].

The interconnection networks present in DSM machines can vary greatly, depending on several factors. The processing nodes can either be loosely or tightly coupled by the network. Loosely coupled machines are also known as clusters and can consist of multiple individual computers tied together by local area networks (LANs) or wide area networks (WANs). Tightly coupled architectures usually consist of processing nodes that have their CPU, memory and network modules tightly integrated and connected with specially designed, low-latency, high-bandwidth networks inside of one large chassis. The amount of coupling present in a DSM is directly related to its cost, size, and performance [3]. A DSM using specially designed, high performance networks with tightly integrated nodes tends to cost more and perform better than a design that uses commercial networking products and readily available workstations as processing nodes. The types of networks used in both approaches can further vary according to their topologies and inner communication structures.

The rest of this paper will present a few examples of tightly and loosely coupled DSMs and the alternative networks they employ. The effectiveness of their network design as well as of the techniques employed to improve performance will be presented. Some results from benchmarks run on these machines will also be shown to gauge the effectiveness of their overall design.

2. Tightly Coupled Machines

2.1 The SGI Origin

Arguably the most commercially successful DSM in the parallel computing field is the SGI Origin line of multiprocessors [4]. The SGI Origin 2000 is a CC-NUMA multiprocessor that uses a directory-based cache coherence protocol. Its DSM architecture provides global addressability of all memory, and its I/O subsystem is also

globally addressable. DMA operations can be performed between any processor and any I/O device address. The basic building block of the Origin is a dual-processor node that contains up to 4GB of main memory, a Hub, and an Xbow I/O interface. The system can have up to 512 Origin nodes, for a maximum of 1024 processors and supports up to 1TB of main memory. The Hub is a combined communication/coherence controller and network interface for each node, while the Xbow handles all traffic between the nodes and their I/O devices. A diagram of these components is shown in Figure 2 [4]. The bandwidth of the bus connecting the two processors to the Hub in a node is 780 MB/s, which is also the bandwidth of the Hub's bus to the memory on the node. The bandwidth of the buses from the Hub to both the network router chip and the Xbow controller are 1.56GB/s each. The design of the Origin represents numerous features common to high-performance DSMs, as well as incorporating many of the common network optimizations in use today.

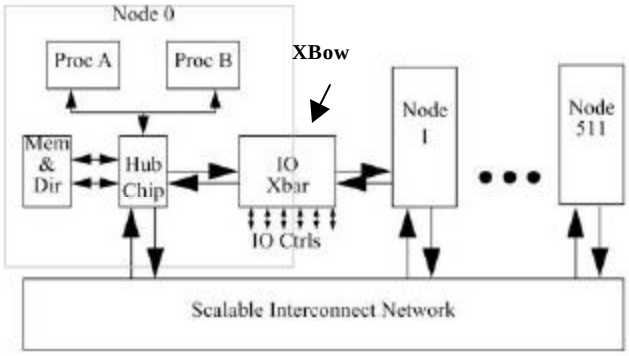

Figure 2 Origin block diagram

The Origin interconnection network uses SPIDER [5] router chips to form hypercube network topologies. The SPIDER chips are scalable, and connect to form efficient hypercube topologies as the number of nodes is increased. The SPIDER chip features six pairs of unidirectional links per router, low-latency wormhole routing, four virtual channels per physical channel, congestion control for adaptive routing between virtual channels, support for message priorities via packet aging, CRC checking on each packet with retransmission on error via a sliding window protocol, and software programmable routing tables. Each pair of links has high bandwidth (1.56 GB/s total per link in the two directions) and low latency (41 ns pin-to-pin through the router). The performance of the Origin's network relies on its high-bandwidth interconnection topology and the features of the SPIDER routers, which will be discussed individually.

Architecturally, the Origin enjoys many advantages from the design of its interconnection network and router chips. It has been shown that network speed, link width, and network topology all greatly affect the performance of a DSM [8]. The Origin employs a high-speed network, capable of ideally transferring 780MB/s along each unidirectional link. More realistically, it has been reported that the network bandwidth is about 613MB/s, measured for a one-way 512MB data transfer in 16KB chunks [9]. This is still quite fast, as each router has six pairs of such

links, with two pairs used to connect its two nodes and the other four pairs connected to adjacent routers in the hypercube. The link width in the Origin is assumed to be 2 bytes, as the SPIDER operates at 400 MHz and ideally transfers 780MB/s. Dai and Panda[8], observed that as the granularity of a parallel application decreases, increasing the link speed has a substantial benefit on the overall execution time. They also observed that for increasing link widths, there was not a significant amount reduction in execution time, and that increasing network speed was more beneficial than increasing the link width. From this it seems that the Origin's choice of a faster clock rate and lower width favors overall performance.

2.1.1 Origin's K-ary N-cube Topology

A general performance analysis of k-ary n-cube interconnection networks has long been a standard method of evaluating the effectiveness of typical multiprocessor topologies [7]. The Origin hypercube has a k (radix) value of 2, so that there are 2 vertices per dimension and it can have n spatial dimensions. Each vertex in the Origin corresponds to a SPIDER router and two Origin nodes. A 3-dimensional hypercube (binary 3-cube) is formed for 32 processors, a fat hypercube is formed for 64 processors, and a hierarchical fat hypercube is formed beyond 64 processors. Figure 3 [4] shows these arrangements for a 32 and 64 processor system. The topologies formed by scaling the Origin system enjoy a constant bisectional bandwidth, a desirable attribute in parallel systems [6]. With the topological features of the Origin in mind, the impact of its design can be evaluated.

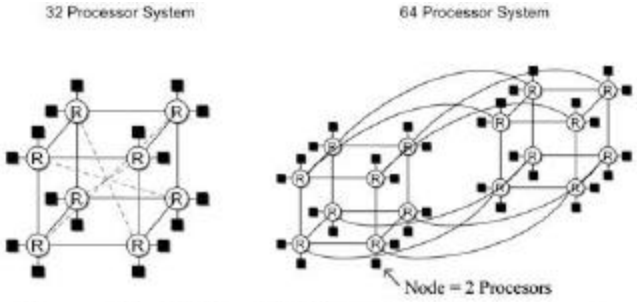

Figure 3 32P and 64P Bristled Hypercubes

The way the network topology of the Origin is scaled provides it with a constant bisection bandwidth. The bisectional bandwidth of a topology is determined by dividing the system in half (bisecting) and counting the number of cut link connections that would have let one half communicate with the other half. As the number of processors in a system grows, it is highly desirable to have the bandwidth of the topology grow proportionally to prevent bottlenecks and reduce the amount of congestion. An advantage of the Origin scalable topology is that the hypercube system shows a constant 1/8 cut-links per CPU, regardless of its size, while maintaining constant link width. This bandwidth scaling is apparent from the fact that an n+1-dimensional hypercube can be constructed by joining two-dimensional hypercubes at their common vertices. Dally [7] has shown that maintaining a constant

bisection bandwidth without decreasing the link width is essential to favorable network performance in a k-ary n-cube. Indeed, it has been shown by Jiang [9] and Laudon [4] that the latencies for memory accesses scale somewhat linearly as the Origin machine size is increased, according to the number of switches a packet has to traverse. The choice of the Origin's topology and interconnection scheme allows it to scale well, and avoid the congestion and bottlenecks that plague some DSM topologies.

2.1.2 Routing in the Origin

Besides physical parameters, there are other design considerations that improve network performance in the Origin. To motivate these optimizations, a review of interconnect network properties follows. An interconnection network is specified by its topology, routing and flow control [12]. The topology of a network is the arrangement of nodes and channels into a graph while routing deals with how a packet chooses a path in this graph. Flow control handles the assignment of channel and buffer resources to a packet as it travels along its path. Packets are the smallest unit of information in a network that contains routing and sequencing information. A packet contains one or more flow-control digits, also called flits. A flit is the smallest unit over which flow control is performed. Information is transferred over physical channels in physical transfer units called phits, which are usually the same size or smaller than flits. It is known that the most costly resource in an interconnection network is physical channel (wire) bandwidth, and the second most costly resource is buffer memory [11]. In a simple interconnection network, throughput is limited because of the way buffers and channels are allocated [10]. Usually a single FIFO buffer at a router is associated with each channel in such a network. While this simplifies the router design, it often results in a router becoming blocked once its buffer fills due to congestion on one of its outgoing links. This prevents other packets from using the blocked router to go to other uncongested channels. This is especially true in store-and-forward routed networks [13] that wait until an entire packet has arrived at a router before sending it on to the next router in its path.

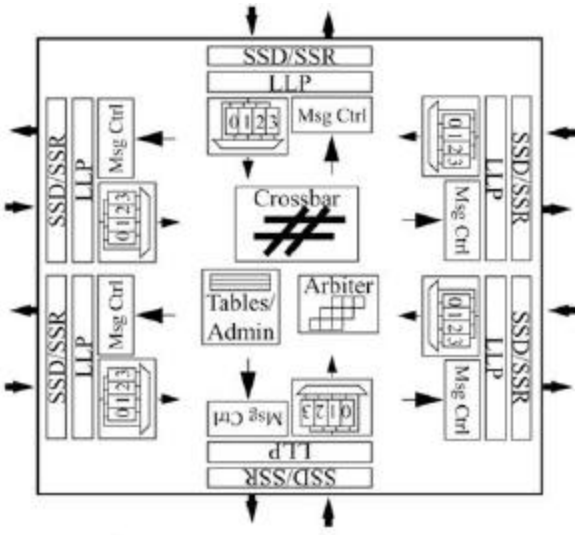

Figure 4 SPIDER ASIC block diagram

One improvement over standard network technique is seen in the Origin's use of low-latency wormhole routing in its SPIDER chips. Wormhole routing is a flow-control protocol that pipelines packet transfer by advancing each flit of a packet as soon as it arrives at the router [14]. It blocks packets in place if the outgoing channel is unavailable. The advantages of wormhole routing are that it reduces the latency of message delivery when compared to store-and-forward routing [11], and it requires that only a few flits be buffered at each router as the packet makes its way to its destination. This reduces the amount of buffering necessary at each router and allows faster, cheaper routers to be built. Wormhole routing is widely used, and works especially well with the virtual channel flow control technique.

2.1.3 Flow-Control Optimizations

Combined with wormhole routing, the SGI Origin implements four virtual channels per physical channel, using virtual channel flow control [11]. In a simple router, each input channel has a single FIFO buffer associated with it, and all such input buffers connect to an output channel switch. A conventional router then restricts buffer allocation so that only flits from the same packet can be stored on a given input buffer. If a packet becomes blocked from leaving the router on its chosen output channel, this prevents the buffer from being used for other incoming packets. While this problem is alleviated somewhat by wormhole routing, it can be further helped with the use of virtual channels. With this scheme, each buffer can be split into several buffers that can each be used for different packet flits. These buffers form virtual input channels and a blocked outgoing packet will then only block one of these virtual channels, allowing the rest of the virtual channels to be used for routing other incoming packets through idle output channels. Figure 4 shows a block diagram of the SPIDER routing chip, with the virtual channels associated at each physical input channel [4]. The reduction of the original buffer size into smaller virtual channel buffers is also not as severe of a problem when wormhole routing is used, as usually only a few flits for a packet need to be buffered at a router. Dally [11] has shown that the use of virtual channels in an interconnection network significantly increases throughput and reduces latency as network traffic increases towards saturation. Additionally, the use of virtual channels allows adaptive routing and message priority techniques to be naturally implemented, which can further improve network performance in the SGI Origin.

The Origin reserves one of its virtual channels for request network transactions, another for responses, and the remaining two can be used for congestion relief and high-priority transactions when needed, or are used for I/O traffic otherwise. This combination of channels allows the Origin to support adaptive routing techniques to avoid congested areas, priorities for packets in the network, and error-free packet transfers. Inside of each SPIDER chip is a software programmable routing table. When a router forwards a packet through its request virtual channel to the next router in the packet's path, the receiving router sends back an acknowledgement through the response virtual channel to the sending router. This acknowledgement contains information on whether the flit was accepted,

depending on buffer space in the receiving router. This information is used to update internal software-programmable routing tables in the router chips, allowing it to identify channels that are congested with network traffic, and letting it specify alternate routes for packets. This adaptive routing helps to distribute traffic evenly across the network, and avoid heavily congested areas, increasing overall network throughput. The acknowledgements sent between routers are generally quite small, and consume a very small amount of the network bandwidth in the Origin. Additionally, when a packet arrives at a destination node's router, it is CRC-checked for errors, and the router maintains a sliding-windows protocol that allows for retransmissions on errors. This ensures error-free network data transfers.

2.1.4 Congestion and Deadlock Issues

In addition to adaptive routing to avoid congested areas, message priorities can be assigned to packets traversing the network, giving preference to high-priority messages for routing towards its destination. Priorities are increased the longer a message remains in the network, via a packet-aging scheme. This ensures that the longer a packet blocks due to congestion, the more its priority is increased, which ensures prompt delivery of those messages which have spent more time in the network. These message priorities help reduce the average latency of packets in the network, as well as providing a way for important messages to quickly traverse the interconnection network.

While the enhancements employed in the SPIDER chip increase network performance in the SGI Origin, they also introduce some common problems that must also be dealt with, namely those of out-of-order message arrivals and deadlock in the network. In the Origin, messages can arrive out-of-order with respect to when they were sent, due to the adaptive routing and congestion avoidance techniques used. This can cause race conditions in the way data is updated by the machine. To deal with this problem, the Origin uses a more complex cache-coherence protocol than what simpler DSMs use, which detects out-of-order message arrivals and resolves their arrivals correctly. The possibility of deadlock between routers in a network is also resolved by the Origin through the use of its virtual channels [14] and its flow-control protocol, which detects deadlock situations locally and removes messages from the network to break deadlock cycles.

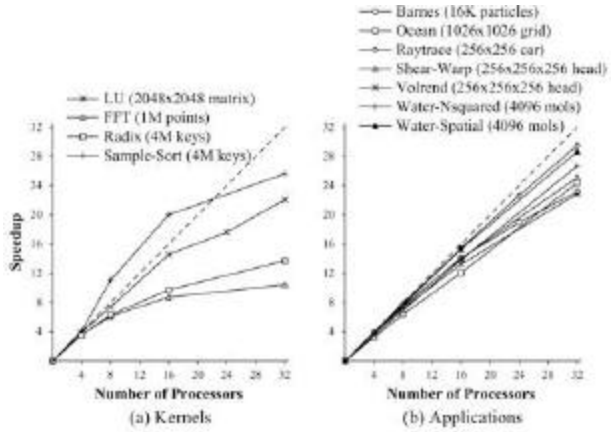

Figure 5: Application speedup with 32 processors.

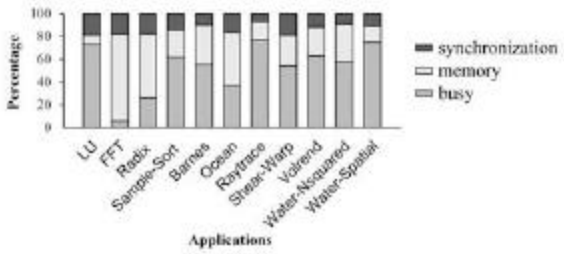

Figure 6: Average time breakdown with 32 processors.

2.1.5 Performance Evaluation of the Origin System

To evaluate the overall effectiveness of the SGI Origin as a DSM, the results from several benchmark programs that make use of shared-memory parallelism have been published [9]. Representative kernels and applications from the SPLASH-2[15] application suite were chosen to be executed and their performance analyzed on the Origin. The speedups obtained over the uniprocessor execution time of each application are plotted for different numbers of processors in Figure 5. Figure 6 shows splits the average execution times of each program into the percentages of time spent on synchronization stalls, memory stalls, and busy time spent executing code. Overall, the Origin generally delivered good parallel performance for the programs shown, although only a maximum configuration of 32 processors was tested. Of these, FFT, Radix and Ocean were found to waste significant amounts of time during communication operations, indicative of poor network response. Both FFT and Radix exhibit all-to-all communication patterns and bursty communication traffic during execution, which strains the network despite its high bandwidth design. Upon investigation, this appears to be due to the fact that a node consists of two processors which share a coherence controller and that a router is shared between two nodes. The FFT and Radix programs cause contention for access to the network at these interfaces, which significantly reduces their performance. Ocean also spends a large amount of time waiting for communication operations since it repeatedly requests large blocks of remote data during execution, but still shows very good speedup compared to the uniprocessor case. Overall, the aggressive communication architecture greatly helps execution speeds for most programs by achieving a low latency ratio between remote and local memory accesses.

The performance of the Origin shows the advantages obtained by using a tightly coupled DSM architecture. The use of a high-speed interconnection network with numerous optimization techniques helps reduce the impact of supporting a shared memory environment across physically distributed processing nodes. Unfortunately, the decision to share some network resources between processors to save on space and cost issues has been reflected in the poor communication performance of some applications, despite the Origin's powerful network topology. The other major tradeoff with a tightly coupled machine such as the Origin is reflected in its cost, both in terms of time spent

during development and final production costs. For example, a typical 24 processor (250 MHz) Origin 2000 costs well over $150,000 [16]. The high cost of such tightly coupled DSM machines has prompted research to investigate loosely coupled architectures in an attempt to find a cheaper way to achieve adequate parallel performance.

2.2 Other Tightly Coupled DSM Machines

Besides the SGI Origin, there are many other examples of tightly coupled DSM multiprocessors that make use of different interconnection networks. This section will highlight some of the interesting designs and features of the interconnection networks used in other machines.

2.2.1 The Stanford FLASH Machine

The Stanford FLASH Multiprocessor [17] is another scalable CC-NUMA machine that has been widely studied in the field, and whose network structure has been used in many shared memory machine designs. It makes use of a 2D-mesh topology, which is essentially a k-ary 2-cube, where k gives the number of nodes along each of the k rows in the grid. Each node consists of a commercial MIPS processor, a shared memory module, and a node controller used to interface with the network and I/O devices for that node. Figure 7 shows an overview of the FLASH architecture. FLASH makes use of a programmable microcontroller called MAGIC as its node controller, which controls all data transfers both within the node and between the node and the network. The latest information on the FLASH project shows that only 3 processors had been connected using an Origin 2000 interconnection network, and work continues to create a larger machine. A simulation of a DSM very similar to FLASH was undertaken to explore the effects that the choice of interconnection network parameters would have on its performance [8].

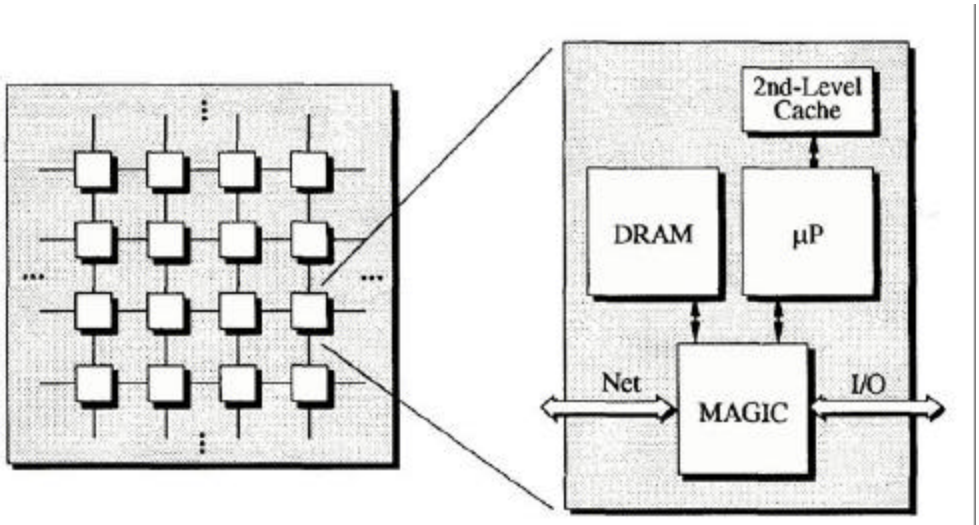

Figure 7: Flash Architecture

2.2.1.1 FLASH Network Design Evaluation

The FLASH network simulation was done for a 64-processor machine for which the performance impact of different network speeds, link widths and topologies was observed. All simulations were based on a k-ary n-cube interconnection network using wormhole routing. Four applications from the SPLASH-2 suite were executed and their relative execution times measured. Figure 8-(a) shows the results as the link speeds of an 8x8 mesh with 16-bit wide links were varied from 100, 200 and 400 MHz, with a router delay of 40ns. Figure 8-(b) presents the execution times for the same network, fixed at a link speed of 200 MHZ, as the link widths were varied between 8,16 and 32 bits. Finally, Figure 8-(c) lists the execution speeds as the topology of the machine is changed from 2D (8-ary 2-cube), 3D (4-ary 2-cube) and 6D (2-ary 6-cube) under a constant bisection bandwidth constraint [7]. From these results, several observations were made. Better performance was achieved by increasing the link speed instead of the link width. Increasing the dimension of a network under a constant link width constraint was somewhat beneficial. Increasing the dimension of a network under constant bisection bandwidth (with decreasing link width) was detrimental to performance, showing that a 2-D mesh was optimal in this case. Routing delays play an important factor in the design of other network components, while reducing network contention experienced by short messages was crucial to overall performance.

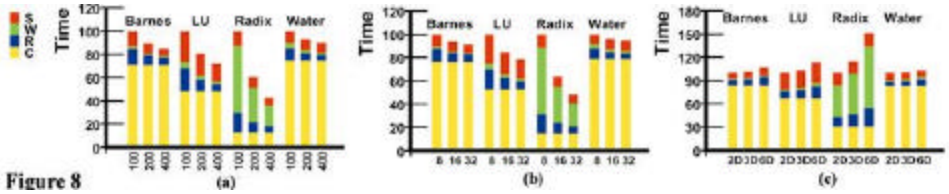

Figure 8

2.2.2 Highlight of Other Tightly Coupled DSM Machines

The MIT Alewife [18] is another DSM that uses a 2D-mesh topology, implemented as a wormhole routed, direct network. It also uses a special chip to coordinate a node's processor, memory and network interface. The Sequent STiNG is composed of 4-processor nodes linked together by a single unidirectional Scalable Coherent Interface ring[19], that provides a low-latency interconnection network with enough bandwidth for future growth. The NUMAchine [20] created by the University of Toronto, uses a hierarchical 3-level unidirectional ring structure to implement a 64-processor DSM with a connection bandwidth of 400MB/s at any point in the network. Both the NUMAchine and STiNG designs attempt to use as many commercial, off-the-shelf components as possible to reduce production costs, although integration of components from different manufacturers also required significant effort. Recently, a DSM based on a bi-directional ring has been proposed to improve performance over unidirectional ring designs [21].

3. Loosely Coupled Machines

Although tightly coupled DSMs generally provide high performance for parallel applications, they are also restricted by their prohibitively high cost, need for custom designed hardware, and requirement that they be physically integrated. In an attempt to overcome these deficits, there have been several efforts to create loosely coupled DSMs with adequate levels of performance. These networks tend to be comprised of collections of individual commercial workstations or personal computers connected using commercially available networks.

3.1 TreadMarks System

TreadMarks [22] is a software DSM system designed to run on commonly available Unix systems. It was initially implemented on the DEC Ultrix operating system using 8 DECStation-5000/240 workstations. These workstations were connected by both a 100Mbps point-to-point ATM [13] LAN and by a 10Mbps Ethernet [13] LAN to compare network performance. Each machine has a Fore ATM interface that is connected to a Fore ATM switch. The connection between the interface and the switch operates at 100-Mbps, and the switch has an aggregate throughput of 1.2-Gbps. The interface board does programmed I/O into transmit and receive FIFO buffers, and requires fragmentation and reassembly of ATM cells by software. Interrupts are raised at the end of a complete message, or when a FIFO buffer becomes full. TreadMarks was also implemented on the SunOS operating system using SPARCstation-1 and -2's connected by a 10-Mbps Ethernet LAN to ensure usability on different UNIX systems. Overall, the TreadMarks system features many of the advantages that can be obtained through the use of loosely coupled systems instead of tightly coupled ones.

3.1.1 Implementation Details for TreadMarks

The implementation of TreadMarks is done at the user level of the UNIX operating system, so that no modifications to the system kernel are necessary. There is no reliance on any particular compiler for implementation on a compatible operating system. TreadMarks is used as a user-level library linked in with shared memory applications, and makes use of conventional Unix socket, memory and signal handling interfaces to implement communication and memory management. Interprocessor communication between workstations in TreadMarks is accomplished through the UDP/IP[13] protocol on an Ethernet or ATM LAN, or through the AAL3/4 protocol on the ATM LAN. AAL3/4 is a connection-oriented, unreliable message protocol specified by the ATM standard. Since neither of these protocols guarantees reliable delivery, TreadMarks uses operation-specific, user-level protocols on top of UDP/IP and AAL3/4 to insure delivery. Since TreadMarks was designed for use over commodity networks that generally have higher latency and lower bandwidth than specially designed interconnection networks, its design attempts to reduce the amount of communication necessary to maintain shared-memory consistency. Towards this end, it presents a release consistency model [23] to the user, which requires less communication than conventional sequentially consistent shared-memory, but provides a very similar programming interface. TreadMarks uses a lazy version of this model to further reduce the number of messages and amount of

data sent compared to eager implementations, which are more commonly found on hardware DSMs. The use of a multiple-writer protocol is also supported, to reduce communication due to false-sharing of cache blocks between processors. False-sharing happens when processors modify different addresses inside of the same cache block and the cache-coherence protocol falsely assumes that the entire block needs to be updated each time across the DSM.

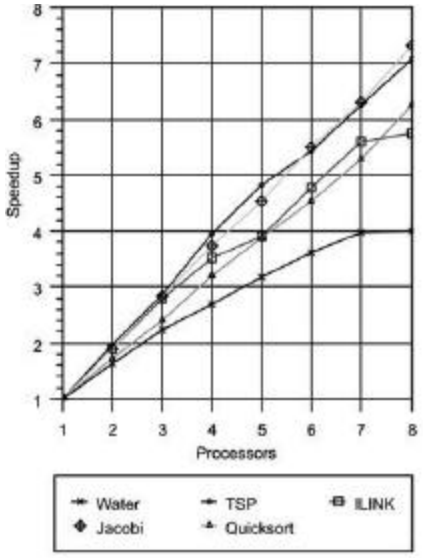

Figure 9: TreadMarks Speedups

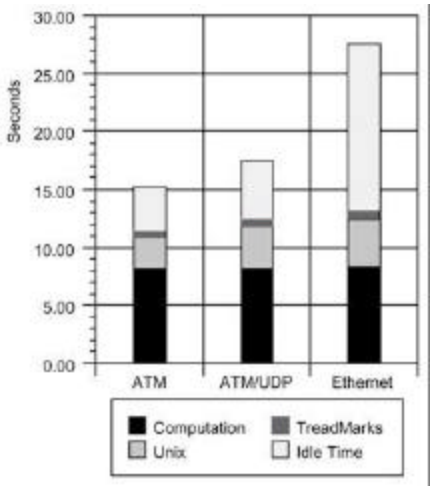

Figure 10: Execution Time for Water

3.1.2 Evaluation of TreadMarks Performance

The basic communication costs of TreadMarks are significantly higher than a typical hardware DSM. The minimum roundtrip time for the smallest possible message was found to be around 500 µs when using explicit send and receive operations, while this increased to 670 µs when using signal handling interrupts. The Origin, on the other hand, shows an average roundtrip time of 775 ns for an 8-processor system. Similarly, other communication operations used in a DSM for shared memory exchanges and to maintain cache-coherence are more expensive on TreadMarks by several orders of magnitude compared to tightly coupled architectures. To evaluate the effectiveness of TreadMarks, several applications from the SPLASH benchmark suite were used, including Water(modified), Jacobi, TSP, Quicksort and ILlink. Figure 9 shows the speedups obtained on these programs with TreadMarks implemented over the ATM LAN using the AAL 3/4 protocol. Overall, good speedups were achieved on these benchmarks, except in the case of Water, which requires a great deal of communication. Further investigation into the execution characteristics of these applications reveals the disadvantages of the TreadMarks software DSM system.

In order to achieve good performance with a system such as TreadMarks, the amount and frequency of communication must be minimized. Other than Water, the applications chosen[22] exhibit very low communication to computation ratios during their execution. Even Water achieves a decent speedup only because the authors implemented a modified version with relaxed consistency models that reduces the volume of communication

significantly compared to the standard implementation of this benchmark. Since most real shared-memory applications tend to be communication intensive, Water was selected to examine the performance difference when three different communication substrates were used. Figure 10 shows Water's execution time breakdown according to computation, Unix overhead, TreadMark overhead and idle times when run over ATM, using both the standard UDP/IP protocol and AAL3/4 protocol, and also over standard 10-Mbps Ethernet using the UDP/IP protocol. Computation time refers to the time spent actually executing code, while idle time refers to the time wasted waiting for communication to complete across the network. Similarly, Unix time is the time spent executing kernel and library code and TreadMarks time is the time spent executing the TreadMarks library code. From the figure, it can be seen that the AAL3/4 protocol over ATM performs best. The computation and TreadMarks overhead times remain constant across the three platforms, while the Unix overhead increases slightly due to increasing cost of the Unix kernel and library code execution for the different communication protocols. The increasing idle time from the AAL3/4 to the UDP/IP protocols over ATM corresponds to the increased protocol overhead in processing the network packets, while the large increase shown on Ethernet is due to network saturation.

3.1.3 Comparison between TreadMarks and a Tightly Coupled DSM

To better gauge the difference between the TreadMarks implementation on a loosely coupled architecture and a tightly coupled architecture, the same benchmarks were also run on an SGI 4D/480 multiprocessor [24]. This is an 8-processor shared memory machine that uses the same 40MHz MIPS R3000 processors that were used in the workstation cluster mentioned previously. Since the processor and memory hierarchies are the same on both the TreadMarks cluster of workstations and the SGI machine, this allowed the authors to compare the difference in their communication mechanisms. Their results showed that for the four benchmarks with low communication to computation ratios, the performance of TreadMarks was fairly close to that of the SGI machine. For Water, the differences in performance were drastic, since this application uses a great deal of communication, even in its optimized version. For communication intensive applications, the hardware DSM clearly performed better due to its dedicated, high-performance network. To try and reduce unnecessary software overhead, the TreadMarks library was given a kernel-level implementation. With some of the software overheads minimized, speedups for Water went from 3.96 in the user-level implementation to 5.6 in the kernel level implementation, while the SGI still had a speedup of 7.17. Moving the communication layer of TreadMarks closer to the hardware level showed a good level of improvement in its performance as a DSM.

For parallel applications exhibiting a small communication to computation ratio, TreadMarks provides very good speedup at a much lower cost than a tightly coupled DSM, with the help of a high-performance commodity network. It is easily implemented on top of existing Unix Workstations, and its programming environment is almost identical to typical shared-memory environments, making TreadMarks relatively easy to use. Unfortunately, it still performs

worse on communication intensive applications for several reasons. Many of the less expensive commodity networks provide much worse throughput and higher latencies than interconnection networks designed for use in a hardware DSM. Also, the overhead required by the software communication layer in TreadMarks is still very expensive. Lower overhead user-level communication interfaces on workstation platforms, or kernel level implementations of the communication protocols are required to improve performance. The TreadMarks system is a step in the right direction for loosely coupled DSM architectures, providing low-cost parallel performance while sacrificing communication speed.

3.2 The Brazos System

Another promising system for implementing a shared memory multiprocessor using clusters of workstations can be found in the Brazos system [25]. Brazos is similar to TreadMarks, but has several key differences. It also is a software DSM system, but is designed to run on x86 multiprocessor workstations using Windows NT 4x. It has been implemented on a cluster of four Compaq Proliant 1500 servers connected by a 100Mbps Ethernet LAN. Each Proliant machine has two 200 MHz Pentium Pro processors with 192MB of main memory. Brazos makes use of selective multicast, multithreading, a software-only implementation of scope consistency and several adaptive runtime performance tuning mechanisms. Each of these features helps Brazos approach the performance of a tightly coupled DSM.

3.2.1 Brazos Implementation Details

The decision to design Brazos for the x86 architecture was made for a few reasons. Recent improvements in commodity networks and processors have made networks of multiprocessor PC workstations an inexpensive alternative to traditional tightly coupled DSM multiprocessors. The Windows NT operating system natively supports true preemptive multithreading, includes the TCP/IP transport protocol with multicast support, and is compatible with symmetric multiprocessor machines, such as the Compaq Proliant 1500. Brazos was designed as a multithreaded system, allowing the overlap of computation with the long communication latencies typically associated with software DSM systems. Multithreaded user-code execution is also supported, lettings programs take advantage of the tightly-coupled shared memory available on the local multiprocessor PC machine while transparently interacting with remote virtually shared memory on the other servers in the cluster. Just like any other DSM implementation, Brazos relies on its interconnection network for performance, and employs its own optimizations to reduce communication delays. Brazos was designed for a time-multiplexed network environment with fast, multicast support, such as Ethernet. By specifying multiple recipients for each message, a large reduction in the number of messages sent and data transferred to maintain cache coherence can be achieved. Brazos also makes use of the scope consistency shared memory model, which attempts to further reduce the amount of coherence traffic seen compared to other models such as release consistency.

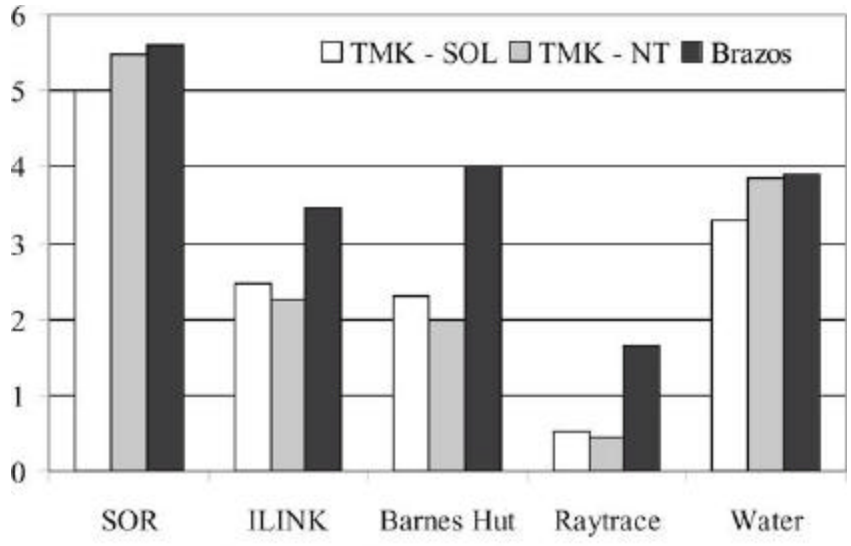

Figure 11: Performance of Brazos vs TreadMarks on 8 Processors

3.2.2 Performance Comparison Between Brazos and TreadMarks

To compare the effectiveness of these Brazos design decisions, it was compared against the TreadMarks system using the same applications mentioned previously [25]. TreadMarks was run on both a Solaris Unix system and was also ported to Windows NT. Both versions of TreadMarks and Brazos were used on the 8-processor Compaq Proliant cluster discussed for the benchmark applications. Figure 11 shows the relative speedups for the 3 platforms, where TMK-SOL is TreadMarks on the Solaris system, and TMK-NT is the Windows NT port of TreadMarks. It should be noted that speedup is measured relative to the uniprocessor case for each platform, and that results from different platforms can not be related directly. The results showed that the Windows NT TCP/IP calls generally involved more operating system overhead than on Unix systems, due to the Windows API for network functions. In general, Brazos' use of multithreading and selective multicast operations allows it to outperform TreadMarks despite its implementation atop an operating system with less efficient network system calls. In a similar study, it was also shown that integrating the use of multithreading and multicast operations into the TreadMarks system improved its performance by an average of 20%, illustrating the effectiveness of these techniques in a software DSM design [26].

4. Conclusion

In the near future, the complexity of computer simulations will grow to demand more powerful platforms for them to run on. Distributed shared memory style parallel machines have developed a considerable amount of raw processing power compared to uniprocessor machines, and have the advantage of providing a similar programming environment to ease the development of parallel software. Tightly coupled architectures show several performance

advantages due to their custom, high-speed interconnection networks. Both software and hardware optimizations exist to maximize the benefit from such a network, and reduce overall system latency. However, the high performance of tightly coupled architectures comes at the cost of increased development time and high financial expense. In an attempt to leverage existing technology and reduce hardware costs, loosely coupled architectures have gained some popularity. These software-coordinated DSM machines make use of much less expensive clusters of workstations connected through a sufficiently fast commercial networking technology, such as Ethernet or ATM. They offer parallel performance at a fraction of the cost that tightly coupled architectures can incur. Unfortunately, even by using several high-speed commercial components, their lack of tight integration is reflected by their poor overall performance on communication intensive parallel applications. Some optimizations have been implemented to try and bridge the gap between the two extreme ends of the DSM spectrum, and the future promises better, cheaper parallel performance as a result.

References

[1] Almasi, G.S. and A. Gottlieb. "*Highly Parallel Computing*". Benjamin/Cummings Publishers. Redwood City, CA. 1989.

[2] Holt, C., J.P. Singh, and J. Hennessy. "*Application and Architectural Bottlenecks in Large Scale Distributed Shared Memory Machines*", in Proceedings of the 23rd Annual International Symposium on Computer Architecture, pp.134-145, 1996.

[3] Judge, A., et. all, "*Overview of distributed shared memory*", Dept. CS, Trinity College, Ireland, Oct. 1998.

[4] Laudon, J. and D. Lenoski, "*The SGI Origin: a ccNUMA Highly Scalable Server*", In Proceedings of the 24th International Symposium on Computer Architecture (ISCA). pp. 241-251. Jun. 1997

[5] Galles, M. "*Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip*", Proc. Symp. High Performance Interconnects (Hot Interconnects 4), Aug. 1996.

[6] Ammon, J. "*Hypercube Connectivity within ccNUMA Architectures*", SGI Origin Team, Mountain View, CA, 1998.

[7] Dally, W.J. "*Performance Analysis of k-ary -cube Interconnection Networks*", IEEE Transactions on Computers, Vol.39, No.6, pp.775-785, 1990.

[8] Dai, D. and D.K. Panda, *"How Can We Design Better Networks for DSM Systems*", Workshop on Parallel Computer Routing and Communication, Atlanta, Georgia, 1997.

[9] Jiang, D. and J.P. Singh, *"A Methodology and an Evaluation of the SGI Origin 2000*", In Proceedings of 1998 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 171-181, June, 1998.

[10] Dally, W.J. "*Wire-Efficient VLSI Multiprocessor Communication Networks*", Proceedings of the Stanford Conference on Advanced Research in VLSI, MIT Press, 1987, pp.391-415.

[11] Dally, W. J. "*Virtual-Channel Flow Control*", Proc. 17th IEEE International Symposium on Computer Architecture, pp.60-68, May 1990.

[12] Dally, W.J. "*Network and Processor Architecture for Message-Driven Computing*", VLSI and Parallel Processing , Morgan Kaufmann, pp. 140-222, 1989.
[13] Tanenbaum, A.S. "*Computer Networks*", Prentice-Hall, Englewood Cliffs, NJ, 1996.
[14] Dally, W.J., "*Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*", IEEE Transactions on Computers, May 1987, p.547-553.
[15] Woo, S.C et. all., "*The Splash-2 Programs: Characterization and Methodological Considerations*", In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pp.24-36, 1995.
[16] SGI website, "*http://www.sgi.com/origin/2000/awards.html*", 1999.
[17] Kuskin, J. et. all, "*The Stanford FLASH Multiprocessor*", 21st Annual Int'l. Symposium on Computer Architecture, pp 302--313, April 1994.
[18] Agarwal, A. et. all, "*The MIT Alewife Machine: Architecture and Performance*", In Proceedings of the 22nd International Symposium on Computer Architecture, pp.2-13 1995.
[19] Lovett, T. and R. Clapp, "*STiNG: A CC-NUMA Computer System for the Commercial Marketplace*", Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996.
[20] Grbic, A. et. all, "*Design and Implementation of the NUMAchine Multiprocessor*", In Proceedings of the 35rd DAC, pp. 66-69, 1998.
[21] Oi, H and N. Ranganathan, "*Performance Analysis of the Bidirectional Ring-Based Multiprocessor*", in Proceedings of ISCA 10th International Conference on Parallel & Distributed Computing Systems, pp. 397-400, October 1997.
[22] Kelcher, P. et. all, "*TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems*", In Proceedings of the 1994 Winter Usenix Conference, pp. 115-131, January 1994.
[23] Gharachorloo, K. et. all, "*Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors*", In Proceedings of the 17th Annual International Symposium on Computer Architecture, pp.15-26, May 1990.
[24] Cox, A.L. et. all, "*Software Versus Hardware Shared-Memory Implementation: A Case Study*", in Proceedings of the 21st Annual International Symposium on Computer Architecture, pp. 106-117. IEEE, April 1994.
[25] Speight, E. and J.K. Bennet, "*Brazos: A Third Generation DSM System*", In First Usenix-NT Workshop, pp. 95-106, Aug. 1997.
[26] Speight, E. and J.K. Bennett. "*Using multicast and multithreading to reduce communication in software DSM systems*", In Proceedings of the Fourth International Symposium on High-Performance Computer Architecture, Feb. 1998.