The SGI Origin: A ccNUMA Highly Scalable Server

James Laudon and Daniel Lenoski
Silicon Graphics, Inc.
2011 North Shoreline Boulevard
Mountain View, California 94043
laudon@sgi.com lenoski@sgi.com

Abstract

The SGI Origin 2000 is a cache-coherent non-uniform memory access (ccNUMA) multiprocessor designed and manufactured by Silicon Graphics, Inc. The Origin system was designed from the ground up as a multiprocessor capable of scaling to both small and large processor counts without any bandwidth, latency, or cost cliffs.The Origin system consists of up to 512 nodes interconnected by a scalable Craylink network. Each node consists of one or two R10000 processors, up to 4 GB of coherent memory, and a connection to a portion of the XIO IO subsystem. This paper discusses the motivation for building the Origin 2000 and then describes its architecture and implementation. In addition, performance results are presented for the NAS Parallel Benchmarks V2.2 and the SPLASH2 applications. Finally, the Origin system is compared to other contemporary commercial ccNUMA systems.

1 Background

Silicon Graphics has offered multiple generations of symmetric multiprocessor (SMP) systems based on the MIPS microprocessors. From the 8 processor R3000-based Power Series to the 36 processor R4000-based Challenge and R10000-based Power Challenge systems, the cache-coherent, globally addressable memory architecture of these SMP systems has provided a convenient programming environment for large parallel applications while at the same time providing for efficient execution of both parallel and throughput based workloads.

The follow-on system to the Power Challenge needed to meet three important goals. First, it needed to scale beyond the 36 processor limit of the Power Challenge and provide an infrastructure that supports higher performance per processor. Given the factor of four processor count increase between the Power Series and Power Challenge lines, it was desired to have the next system support at least another factor of four in maximum processor count. Second, the new system had to retain the cache-coherent globally addressable memory model of the Power Challenge. This model is critical for achieving high performance on loop-level parallelized code and for supporting the existing Power Challenge users. Finally, the entry level and incremental cost of the system was desired to be lower than that of a high-performance SMP, with the cost ideally approaching that of a cluster of workstations.

Simply building a larger and faster snoopy bus-based SMP system could not meet all three of these goals. The second goal might be achievable, but it would surely compromise performance for larger processor counts and costs for smaller configurations.

Therefore a very different architecture was chosen for use in the next generation Origin system. The Origin employs distributed shared memory (DSM), with cache coherence maintained via a directory-based protocol. A DSM system has the potential for meeting all three goals: scalability, ease of programming, and cost. The directory-based coherence removes the broadcast bottleneck that prevents scalability of the snoopy bus-based coherence. The globally addressable memory model is retained, although memory access times are no longer uniform. However, as will be shown in this paper, Origin was designed to minimize the latency difference between remote and local memory and to include hardware and software support to insure that most memory references are local. Finally, a low initial and incremental cost can be provided if the natural modularity of a DSM system is exploited at a relatively fine granularity by the product design.

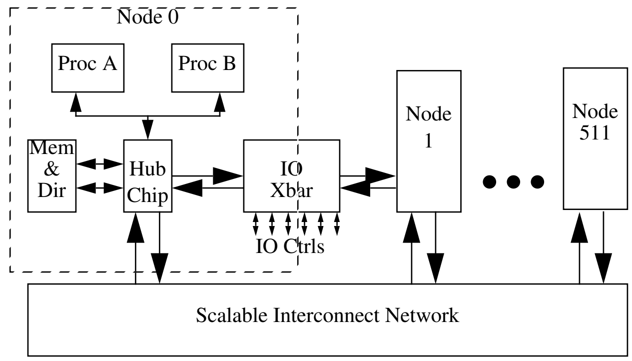

Figure 1 Origin block diagram

In the following section of this paper, the scalable shared-memory multiprocessing (S^2MP) architecture of the Origin is presented. Section 3 details the implementation of the Origin 2000. Performance of the Origin 2000 is presented in Section 4. Section 5 compares the Origin system with other contemporary ccNUMA systems. Finally, Section 6 concludes the paper.

2 The Origin S^2MP Architecture

A block diagram of the Origin architecture is shown in Figure 1. The basic building block of the Origin system is the dual-processor node. In addition to the processors, a node contains up to 4 GB of main memory and its corresponding directory memory, and has a connection to a portion of the IO subsystem.

The architecture supports up to 512 nodes, for a maximum configuration of 1024 processors and 1 TB of main memory. The nodes can be connected together via any scalable interconnection network. The cache coherence protocol employed by the Origin system does not require in-order delivery of point-to-point messages to allow the maximum flexibility in implementing the interconnect network.

The DSM architecture provides global addressability of all memory, and in addition, the IO subsystem is also globally addressable. Physical IO operations (PIOs) can be directed from any processor to any IO device. IO devices can DMA to and from all memory in the system, not just their local memory.

While the two processors share the same bus connected to the Hub, they do not function as a snoopy cluster. Instead they operate as two separate processors multiplexed over the single physical bus (done to save Hub pins). This is different from many other ccNUMA systems, where the node is a SMP cluster. Origin does not employ a SMP cluster in order to reduce both the local and remote memory latency, and to increase remote memory bandwidth. Local memory latency is reduced because the bus can be run at a much higher frequency when it needs to support only one or two processor than when it must support large numbers of processors. Remote memory latency is also reduced by a higher frequency bus, and in addition because a request made in a snoopy bus cluster must generally wait for the result of the snoop before being forwarded to the remote node[7]. Remote bandwidth can be lower in a system with a SMP cluster node if memory data is sent across the remote data bus before being sent to the network, as is commonly done in DSM systems with SMP-based nodes[7][8]. For remote requests, the data will traverse the data bus at both the remote node and at the local node of the requestor, leading to the remote bandwidth being one-half the local bandwidth. One of the major goals for the Origin system was to keep both absolute memory latency and the ratio of remote to local latency as low as possible and to provide remote memory bandwidth equal to local memory bandwidth in order to provide an easy migration path for existing SMP software. As we will show in the paper, the Origin system does accomplish both goals, whereas in Section 6 we see that all the snoopy-bus clustered ccNUMA systems do not achieve all of these goals.

In addition to keeping the ratio of remote memory to local memory latency low, Origin also includes architectural features to address the NUMA aspects of the machine. First, a combination of hardware and software features are provided for effective page migration and replication. Page migration and replication is important as it reduces effective memory latency by satisfying a greater percentage of accesses locally. To support page migration Origin provides per-page hardware memory reference counters, contains a block copy engine that is able to copy data at near peak memory speeds, and has mechanisms for reducing the cost of TLB updates.

Other performance features of the architecture include a high-performance local and global interconnect design, coherence protocol features to minimize latency and bandwidth per access, and a rich set of synchronization primitives. The intra-node interconnect consists of single Hub chip that implements a full four-way crossbar between processors, local memory, and the I/O and network interfaces. The global interconnect is based on a six-ported router chip configured in a multi-level fat-hypercube topology.

The coherence protocol supports a clean-exclusive state to minimize latency on read-modify-write operations. Further, it allows cache dropping of clean-exclusive or shared data without notifying the directory in order to minimize the impact on memory/directory bandwidth caused by directory coherence. The architecture also supports request forwarding to reduce the latency of interprocessor communication.

For effective synchronization in large systems, the Origin system provides fetch-and-op primitives on memory in addition to the standard MIPS load-linked/store-conditional (LL/SC) instructions. These operations greatly reduce the serialization for highly contended locks and barrier operations.

Origin includes many features to enhance reliability and availability. All external cache SRAM and main memory and directory DRAM are protected by a SECDED ECC code. Furthermore, all high-speed router and I/O links are protected by a full CRC code and a hardware link-level protocol that detects and automatically retries faulty packets. Origin's modular design provides the overall basis for a highly available hardware architecture. The flexible routing network supports multiple paths between nodes, partial population of the interconnect, and the hot plugging of cabled-links that permits the bypass, service, and reintegration of faulty hardware.

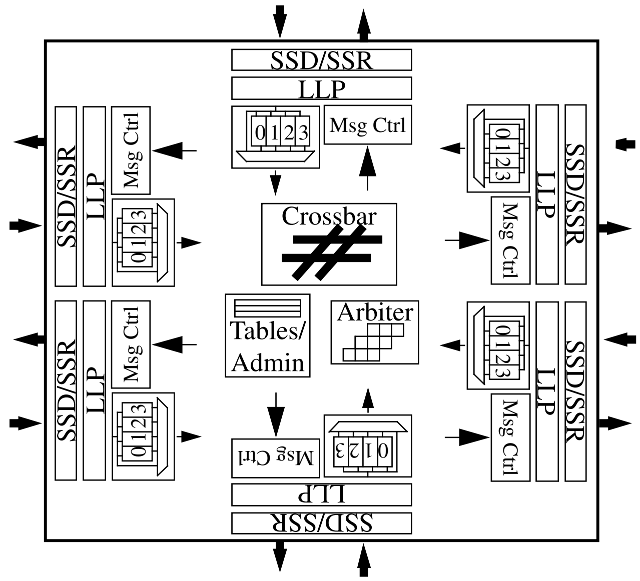

Figure 2 SPIDER ASIC block diagram

To address software availability in large systems, Origin provides access protection rights on both memory and IO devices. These access protection rights prevent unauthorized nodes from being able to modify memory or IO and allows an operating system to be structured into cells or partitions with containment of most failures to within a given partition[10][13].

3 The Origin Implementation

While existence proofs for the DSM architecture have been available in the academic community for some time[1][6], the key to commercial success of this architecture will be an aggressive implementation that provides for a truly scalable system with low memory latency and no unexpected bandwidth bottlenecks. In this section we explore how the Origin 2000 implementation meets this goal. We start by exploring the global interconnect of the system. We then present an overview of the cache coherence protocol, followed with a discussion of the node design. The IO subsystem is explored next, and then the various subsystems are tied together with the presentation of the product design. Finally, this section ends with a discussion of interesting performance features of the Origin system.

3.1 Network Topology

The interconnect employed in the Origin 2000 system is based on the SGI SPIDER router chip[4]. A block diagram of this chip is shown in Figure 2. The main features of the SPIDER chip are:

- six pairs of unidirectional links per router
- low latency (41 ns pin-to-pin) wormhole routing
- DAMQ buffer structures[4] with global arbitration to maximize utilization under load.
- four virtual channels per physical channel
- congestion control allowing messages to adaptively switch between two virtual channels

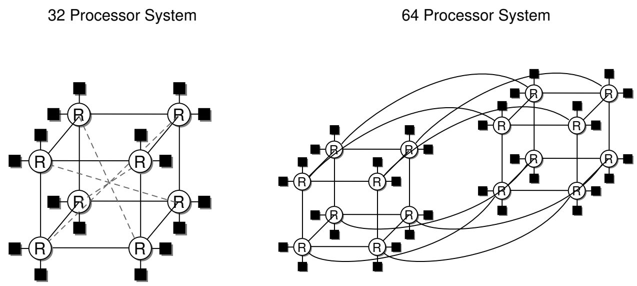

Figure 3 32P and 64P Bristled Hypercubes

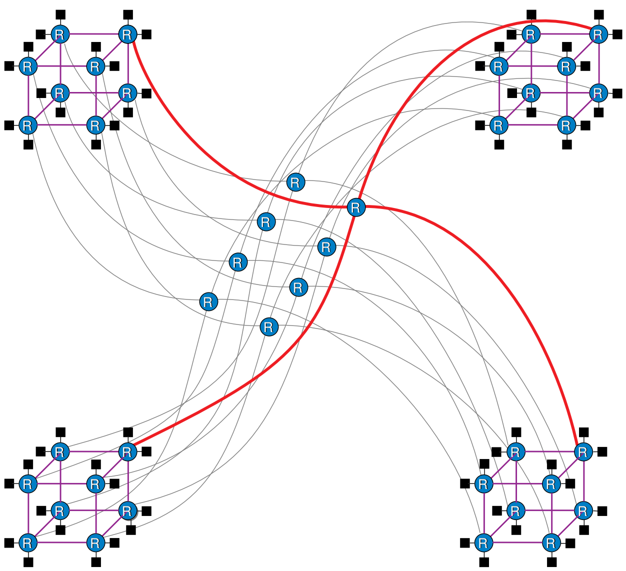

Figure 4 128P Heirarchical Fat Bristled Hypercube

- support for 256 levels of message priority with increased priority via packet aging
- CRC checking on each packet with retransmission on error via a go-back-n sliding window protocol
- software programmable routing tables

The Origin 2000 employs SPIDER routers to create a bristled fat hypercube interconnect topology. The network topology is bristled in that two nodes are connected to a single router instead of one. The fat hypercube comes into play for systems beyond 32 nodes (64 processors). For up to 32 nodes, the routers connect in a bristled hypercube as shown in Figure 3. The SPIDER routers are labeled using R, the nodes are the block boxes connecting to the routers. In the 32 processor configuration, the otherwise unused SPIDER ports are shown as dotted lines being used for Express Links which connect the corners of the cube, thereby reducing latency and increasing bisection bandwidth.

Beyond 64 processors, a hierarchical fat hypercube is employed. Figure 4 shows the topology of a 128 processor Origin system. The vertices of four 32-processor hypercubes are connected to eight *meta*-routers. To scale up to 1024 processors, each of the single meta-routers in the 128 processor system is replaced with a 5-D hypercubes.

3.2 Cache Coherence Protocol

The cache coherence protocol employed in Origin is similar to the Stanford DASH protocol[6], but has several significant performance improvements. Like the DASH protocol, the Origin cache coherence protocol is non-blocking. Memory can satisfy any incoming request immediately; it never buffers requests while waiting for another message to arrive. The Origin protocol also employs the request forwarding of the DASH protocol for three party transactions. Request forwarding reduces the latency of requests which target a cache line that is owned by another processor.

The Origin coherence protocol has several enhancements over the DASH protocol. First, the Clean-exclusive (CEX) processor cache state (also known as the exclusive state in MESI) is fully supported by the Origin protocol. This state allows for efficient execution of read-modify-write accesses since there is only a single fetch of the cache line from memory. The protocol also permits the processor to replace a CEX cache line without notifying the directory. The Origin protocol is able to detect a rerequest by a processor that had replaced a CEX cache line and immediately satisfy that request from memory. Support of CEX state in this manner is very important for single process performance as much of the gains from the CEX state would be lost if directory bandwidth was needed each time a processor replaced a CEX line. By adding protocol complexity to allow for the "silent" CEX replacement, all of the advantages of the CEX state are realized.

The second enhancement of the Origin protocol over DASH is full support of upgrade requests which move a line from a shared to exclusive state without the bandwidth and latency overhead of transferring the memory data.

For handling incoming I/O DMA data, Origin employs a write-invalidate transaction that uses only a single memory write as opposed to the processor's normal write-allocate plus writeback. This transaction is fully cache coherent (i.e., any cache invalidations/interventions required by the directory are sent), and increases I/O DMA bandwidth by as much as a factor of two.

Origin's protocol is fully insensitive to network ordering. Messages are allowed to bypass each other in the network and the protocol detects and resolves all of these out-of-order message deliveries. This allows Origin to employ adaptive routing in its network to deal with network congestion.

The Origin protocol uses a more sophisticated network deadlock avoidance scheme than DASH. As in DASH, two separate networks are provided for requests and replies (implemented in Origin via different virtual channels). The Origin protocol does have requests which generate additional requests (these additional requests are referred to as *interventions* or *invalidations*). This request-to-request dependency could lead to deadlock in the request network. In DASH, this deadlock was broken by detecting a potential deadlock situation and sending negative-acknowledgments (NAKs) to all requests which needed to generate additional requests to be serviced until the potential deadlock situation was resolved. In Origin, rather than sending NAKs in such a situation, a *backoff* intervention or invalidate is sent to the requestor on the reply network. The backoff message contains either the target of the intervention or the list of sharers to invalidate, and is used to signal the requestor that the memory was unable to generate the intervention or invalidation directly and therefore the requestor must generate that message instead. The requestor can always sink the backoff reply, which causes the requestor to then queue up the intervention or invalidate for injection into the request network as soon as the request network allows. The backoff intervention or invalidate changes the request-intervention-reply chain to two request-reply chains (one chain being the request-backoff message, one being the intervention-reply chain), with the two networks preventing deadlock on these two request-reply chains. The ability to generate backoff interventions and invalidations allows for better forward progress in the face of very heavily loaded systems since the deadlock detection in both DASH and Origin is conservatively done based on local information, and a processor that receives a backoff is guaranteed that it will eventually receive the data, while a processor that receives a NAK must retry its request.

Since the Origin system is able to maintain coherence over 1024 processors, it obviously employs a more scalable directory scheme than in DASH. For tracking sharers, Origin supports a bit-vector directory format with either 16 or 64 bits. Each bit represents a node, so with a single bit to node correspondence the directory can track up to a maximum of 128 processors. For systems with greater than 64 nodes, Origin dynamically selects between a full bit vector and coarse bit vector[12] depending on where the sharers are located. This dynamic selection is based on the machine being divided into up to eight 64 node *octants*. If all the processors sharing the cache line are from the same octant, the full bit vector is used (in conjunction with a 3-bit octant identifier). If the processors sharing the cache line are from different octants, a coarse bit vector where each bit represents eight nodes is employed.

Finally, the coherence protocol includes an important feature for effective page migration known as *directory poisoning*. The use of directory poisoning will be discussed in more detail in Section 3.6.

A slightly simplified flow of the cache coherence protocol is now presented for both read, read-exclusive, and writeback requests. We start with the basic flow for a read request.

1. Processor issues read request.
2. Read request goes across network to home memory (requests to local memory only traverse Hub).
3. Home memory does memory read and directory lookup.
4. If directory state is Unowned or Exclusive with requestor as owner, transitions to Exclusive and returns an exclusive reply to the requestor. *Go to 5a*.
   If directory state is Shared, the requesting node is marked in the bit vector and a shared reply is returned to the requestor. *Go to 5a*.
   If directory state is Exclusive with another owner, transitions to Busy-shared with requestor as owner and send out an intervention shared request to the previous owner and a speculative reply to the requestor. *Go to 5b*.
   If directory state is Busy, a negative acknowledgment is sent to the requestor, who must retry the request. QED

5a. Processor receives exclusive or shared reply and fills cache in CEX or shared (SHD) state respectively. QED

5b. Intervention shared received by owner. If owner has a dirty copy it sends an shared response to the requestor and a sharing writeback to the directory. If owner has a clean-exclusive or invalid copy it sends an shared ack (no data) to the requestor and a sharing transfer (no data) to the directory.

6a. Directory receives shared writeback or shared transfer, updates memory (only if shared writeback) and transitions to the shared state.

6b. Processor receives both speculative reply and shared response or ack. Cache filled in SHD state with data from response (if shared response) or data from speculative reply (if shared ack). QED

The following list details the basic flow for a read-exclusive request.

1. Processor issues read-exclusive request.
2. Read-exclusive request goes across network to home memory (only traverses Hub if local).
3. Home memory does memory read and directory lookup.
4. If directory state is Unowned or Exclusive with requestor as owner, transitions to Exclusive and returns an exclusive reply to the requestor. *Go to 5a*.
   If directory state is Shared, transitions to Exclusive and a exclusive reply with invalidates pending is returned to the requestor. Invalidations are sent to the sharers. *Go to 5b.*
   If directory state is Exclusive with another owner, transitions to Busy-Exclusive with requestor as owner and sends out an intervention exclusive request to the previous owner and a speculative reply to the requestor. *Go to 5c*.
   If directory state is Busy, a negative acknowledgment is sent to the requestor, who must retry the request. QED

5a. Processor receives exclusive reply and fills cache in dirty exclusive (DEX) state. QED

5b. Invalidates received by sharers. Caches invalidated and invalidate acknowledgments sent to requestor. *Go to 6a.*

5c. Intervention shared received by owner. If owner has a dirty copy it sends an exclusive response to the requestor and a dirty transfer (no data) to the directory. If owner has a clean-exclusive or invalid copy it sends an exclusive ack to the requestor and a dirty transfer to the directory. *Go to 6b.*

6a. Processor receives exclusive reply with invalidates pending and all invalidate acks. (Exclusive reply with invalidates pending has count of invalidate acks to expect.) Processor fills cache in DEX state. QED

6b. Directory receives dirty transfer and transitions to the exclusive state with new owner.

6c. Processor receives both speculative reply and exclusive response or ack. Cache filled in DEX state with data from response (if exclusive response) or data from speculative reply (if exclusive ack). QED

The flow for an upgrade (write hit to SHD state) is similar to the read-exclusive, except it only succeeds for the case where the directory is in the shared state (and the equivalent reply to the exclusive reply with invalidates pending does not need to send the memory data). In all other cases a negative acknowledgment is sent to the requestor in response to the upgrade request.

Finally, the flow for a writeback request is presented. Note that if a writeback encounters the directory in one of the busy states, this means that the writeback was issued before an intervention targeting the cache line being written back made it to the writeback issuer. This race is resolved in the Origin protocol by “bouncing” the writeback data off the memory as a response to the processor that caused the intervention, and sending a special type of writeback acknowledgment that informs the writeback issuer to wait for (and then ignore) the intervention in addition to the writeback acknowledgment.

1. Processor issues writeback request.
2. Writeback request goes across network to home memory (only traverses Hub if local).
3. Home memory does memory write and directory lookup.
4. If directory state is Exclusive with requestor as owner, transitions to Unowned and returns a writeback exclusive acknowledge to the requestor. *Go to 5a*.
   If directory state is Busy-shared, transitions to Shared, a shared response is returned to the owner marked in the directory. A writeback busy acknowledgment is also sent to the requestor. *Go to 5b*.
   If directory state is Busy-exclusive, transitions to Exclusive, an exclusive response is returned to the owner marked in the directory. A writeback busy acknowledgment is also sent to the requestor. *Go to 5b.*

5a. Processor receives writeback exclusive acknowledgment. QED

5b. Processor receives both a writeback busy acknowledgment and an intervention. QED

3.3 Node Design

The design of an Origin node fits on a single 16” x 11” printed circuit board. A drawing of the Origin node board is shown in Figure

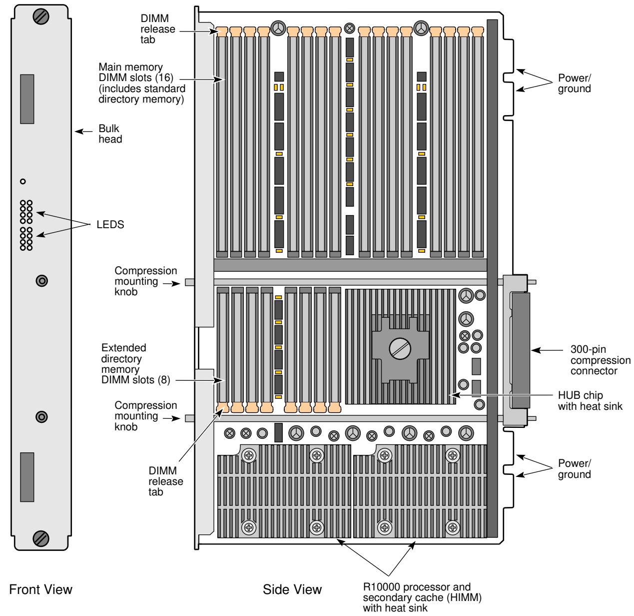

Figure 5 An Origin node board

5. At the bottom of the board are two R10000 processors with their secondary caches. The R10000 is a four-way out-of-order superscalar processor[14]. Current Origin systems run the processor at 195 MHz and contain 4 MB secondary caches. Each processor and its secondary cache is mounted on a horizontal in-line memory module (HIMM) daughter card. The HIMM is parallel to the main node card and connects via low-inductance fuzz-button processor and HIMM interposers. The system interface buses of the R10000s are connected to the Hub chip. The Hub chip also has connections to the memory and directory on the node board, and has two ports that exit the node board via the 300-pin CPOP (compression pad-on-pad) connector. These two ports are the Craylink connection to router network and the XIO connection to the IO subsystem.

As was mentioned in Section 3.2, a 16 bit-vector directory format and a 64 bit-vector format are supported by the Origin system. The directory that implements the 16-bit vector format is located on the same DIMMs as main memory. For systems larger than 32 processors, additional expansion directory is needed. These expansion directory slots, shown to the left of the Hub chip in Figure 5, operate by expanding the width of the standard directory included on the main memory boards. The Hub chip operates on standard 16-bit directory entries by converting them to expanded entries upon their entry into the Hub chip. All directory operations within the Hub chip are done on the expanded directory entries, and the results are then converted back to standard entries before being written back to the directory memory. Expanded directory entries obviously bypass the conversion stages.

Figure 6 shows a block diagram of the Hub chip. The hub chip is divided into five major sections: the crossbar (XB), the IO interface (II), the network interface (NI), the processor interface (PI), and the memory and directory interface (MD). All the interfaces communicate with each other via FIFOs that connect to the crossbar.

The IO interface contains the translation logic for interfacing to the XIO IO subsystem. The XIO subsystem is based on the same low-level signalling protocol as the Craylink network (and uses the same interface block to the XIO pins as in the SPIDER router of Figure 2), but utilizes a different higher level message protocol. The IO section also contains the logic for two block transfer engines (BTEs) which are able to do memory to memory copies at

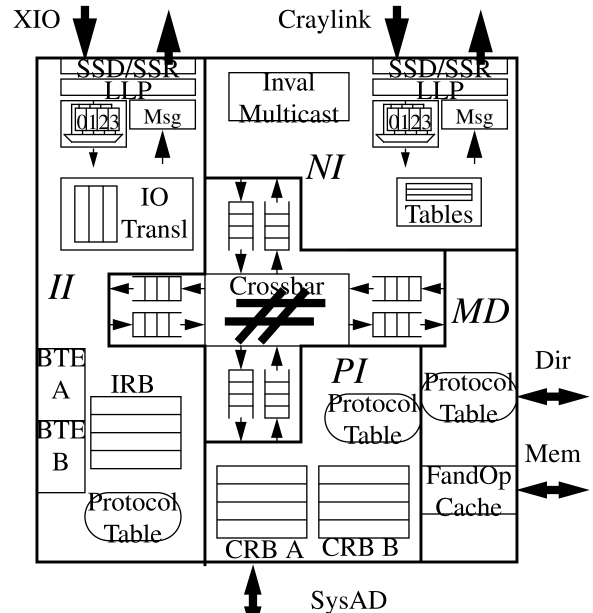

Figure 6 Hub ASIC block diagram

near the peak of a node's memory bandwidth. It also implements the IO request tracking portion of the cache coherence protocol via the IO request buffers (IRB) and the IO protocol table. The IRB tracks both full and partial cache line DMA requests by IO devices as well as full cache line requests by the BTEs.

The network interface takes messages from the II, PI, and MD and sends them out on the Craylink network. It also receives incoming messages for the MD, PI, II, and local Hub registers from the Craylink network. Routing tables for outgoing messages are provided in the NI as the software programmable routing of the SPIDER chip is pipelined by one network hop[4]. The NI also is responsible for taking a compact intra-Hub version of the invalidation message resulting from a coherence operation (a bit-vector representation) and generating the multiple unicast invalidate messages required by that message.

The processor interface contains the logic for implementing the request tracking for both processors. Read and write requests are tracked via a coherent request buffer (CRB), with one CRB per processor. The PI also includes the protocol table for its portion of the cache coherence protocol. The PI also has logic for controlling the flow of requests to and from the R10000 processors and contains the logic for generating interrupts to the processors.

Finally, the memory/directory section contains logic for sequencing the external memory and directory synchronous DRAMs (SDRAMs). Memory on a node is banked 4-32 way depending on how many memory DIMMs are populated. Requests to different banks and requests to the same page within a bank as the previous request can be serviced at minimum latency and full bandwidth. Directory operations are performed in parallel with the memory data access. A complete directory entry (and page reference counter, as will be discussed in Section 3.6) read-modify-write can be performed in the same amount of time it takes to fetch the 128B cache line from memory. The MD performs the directory portion of the cache coherence protocol via its protocol table and generates the appropriate requests and/or replies for all incoming messages.

The MD also contains a small fetch-and-op cache which sits in front of the memory. This fetch-and-op cache allows fetch-and-op variables that hit in the cache to be updated at the minimum net-

Port	SysAD	Mem	XIO	Craylink
GB/s	0.78	0.78	1.56	1.56

Table 1 Hub ASIC port bandwidths

Section	XB	IO	NI	PI	MD
K gates	246	296	56	133	77

Table 2 Hub ASIC gate count

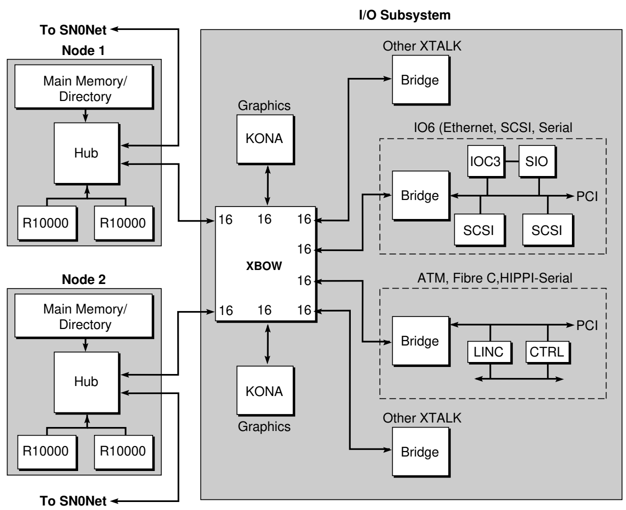

Figure 7 Example IO subsystem block diagram

work reply serialization rate of 41 ns instead of at the much slower SDRAM read-modify-write timing.

Note that all the protocol tables in the Hub are hard-wired. While programmable protocol engines can come close to achieving the performance of a hard-wired protocol state machine[5][9], we opted for hard-wiring the protocol to minimize latency and maximize bandwidth. We were also concerned about the variability in latency and bandwidth given the caching of directory information used by most programmable approaches. To ensure that the cache coherence protocol implemented in the tables was correct, we employed formal verification[3]. Formal verification worked extremely well; no bugs have been found in the Origin cache coherence protocol since the formal verification was completed.

The raw data bandwidth of the Hub chip ports is listed in Table 1. A summary of the sizes of the units is shown in Table 2. Note that most of the chip is allocated either to interfacing to the IO subsystem or in the crossbar itself, rather than in implementing global cache coherence.

3.4 IO Subsystem

Not too surprisingly, the Origin system also utilizes crossbars in its IO subsystem. Figure 7 shows one possible configuration of IO cards connected to two nodes. Using the same link technology as in the Craylink interconnect, each Hub link provides a peak of 1.56 GB/sec of bandwidth to the six XIO cards connected to it (actually limited to half this amount if only local memory bandwidth is considered). At the heart of the IO subsystem is the Crossbow (Xbow) ASIC, which has many similarities with the SPIDER router. The primary differences between the Xbow and the router is a simplification of the Xbow buffering and arbitration protocols given the chips more limited configuration. These simplifications reduce costs and permit eight ports to be integrated on a single chip. Some of the main features of the Xbow are:

- eight XIO ports, connected in Origin to 2 nodes and 6 XIO cards.
- two virtual channels per physical channel
- low latency wormhole routing
- support for allocated bandwidth of messages from particular devices
- CRC checking on each packet with retransmission on error via a go-back-n sliding window protocol

Board	Number of Ports
Base IO	2 Ultra SCSI, 1 Fast Enet, 2 serial
Ultra SCSI	4
10/100 Enet	4
HiPPI	1 serial
Fibre Channel	2 Cu loops
ATM OC3	4
Infinite Reality Gfx	1
Standard PCI Cage	3
VME Adapter	1

Table 3 Origin IO boards

The Crossbow has support in its arbiter for allocating a portion of the bandwidth to a given IO device. This feature is important for certain system applications such as video on demand.

A large number of XIO cards are available to connect to the Crossbow. Table 3 contains a listing of the common XIO cards. The highest performance XIO cards connect directly to the XIO, but most of the cards bridge XIO to an embedded PCI bus with multiple external interfaces. The IO bandwidth together with integration provide IO performance which is effectively added as a PCI-bus at a time versus individual PCI cards.

3.5 Product Design

The Origin 2000 is a highly modular design. The basic building block is the deskside module, which has slots for 4 node boards, 2 router boards, and 12 XIO boards. The module also includes a CDROM and up to 5 Ultra SCSI devices. Figure 8 shows a block diagram of the deskside module, while Figure 9 shows a rear-view perspective of a deskside system. The system has a central midplane, which has two Crossbow chips mounted on it. The 4 node and 12 XIO boards plug into the midplane from the rear of the system, while the 2 router boards, the power supply and the UltraSCSI devices plug into the midplane from the front of the system.

A module can be used as a stand-alone deskside system or two modules (without the deskside plastic skins) can be mounted in a rack to form a 16 processor system. In addition to the two modules, the rack also includes a disk bay for up to 8 additional disks. One of the modules can be replaced with an Infinite reality graph-

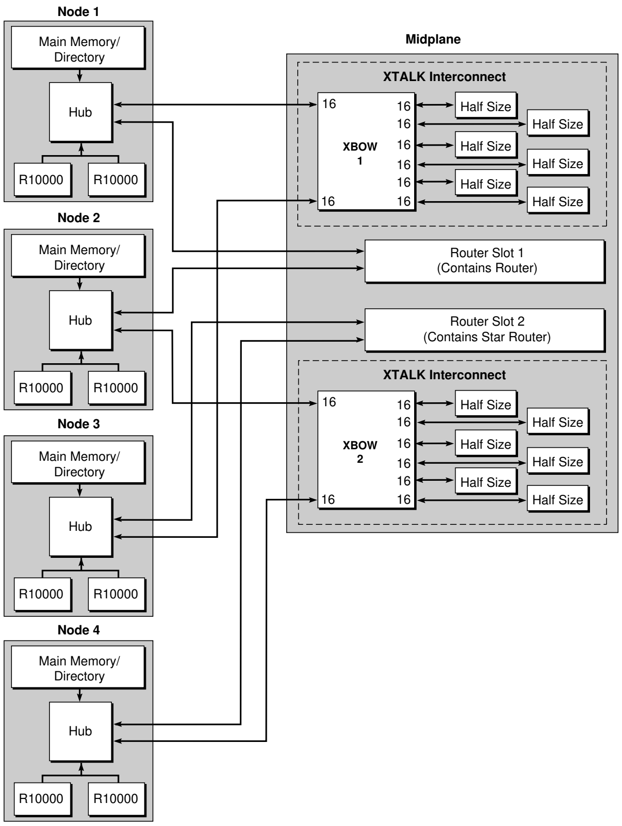

Figure 8 Deskside module block diagram

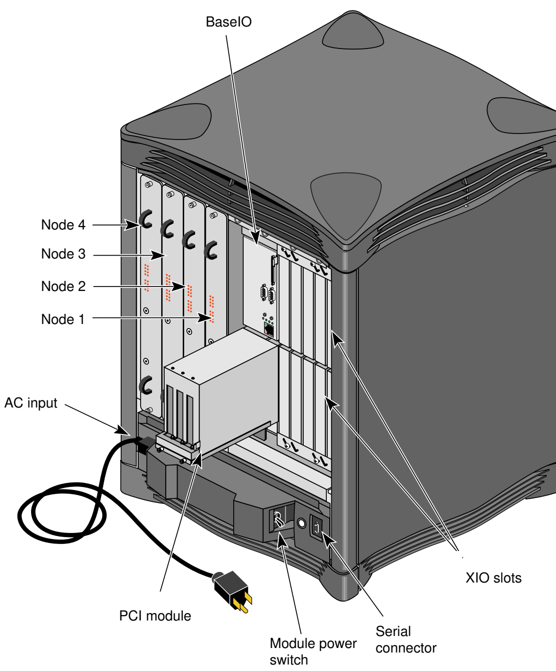

Figure 9 Deskside module, rear view

ics module or with 4 additional 8-disk bays. An Origin Vault which contains 9 8-disk bays in a single rack is also available. Figure 10 depicts a configured rack supporting 16 processors, 24 XIO boards, and 18 UltraSCSI devices.

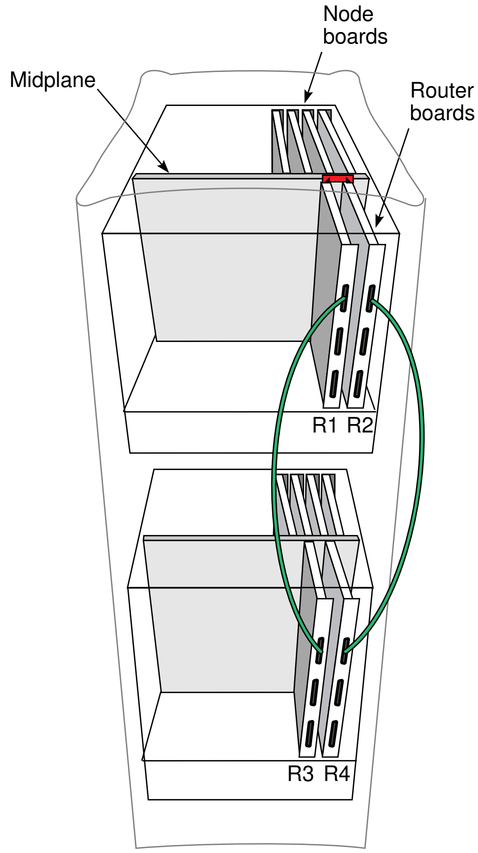

Figure 10 16 processor Origin system.

3.6 Performance Features

The Origin system has two features very important for achieving good performance in a highly scalable system. First, fetch-and-op primitives are provided as uncached operations that occur at the memory. Fetch-and-op variables are used for highly contended locks, barriers, and other synchronization mechanisms. The typical serialization rate (the rate at which a stream of requests can be serviced) for fetch-and-op variables is 41 ns. In Section 4.1 we will show how fetch-and-op variables can improve the performance of highly contended objects.

Second, Origin provides hardware and software support for page migration. Page migration is important for NUMA systems as it changes many of the cache misses which would have gone to remote memory to local misses. To help the OS in determining when and which page to migrate the Origin system provides an array of per-page memory reference counters, which are stored in the directory memory. This array is indexed by the nodes in a system (up to 64 nodes, beyond this 8 nodes share a single counter). When a request comes in, its reference counter is read out during the directory lookup and incremented. In addition, the reference counter of the home node is read out during the same directory lookup. The requestor's count and home count are compared and if the difference exceeds a software programmable threshold register (and the migration control bits stored with the requestor's reference counter says that this page is a candidate for migration), an interrupt is generated to the home node. This interrupt signals a potential migration candidate to the operating system.

When the operating system determines it does indeed want to migrate the page[13], two operations need to be performed. First, the OS needs to copy the page from its current location to a free memory page on the requestor's node. Second, the OS needs to invali-

Memory level	Latency (ns)
L1 cache	5.1
L2 cache	56.4
local memory	310
4P remote memory	540
8P avg. remote memory	707
16P avg. remote memory	726
32P avg. remote memory	773
64P avg. remote memory	867
128P avg. remote memory	945

Table 4 Origin 2000 latencies

date all the translations to the old page cached in processor's TLBs and then update the translation for the migrated page to point to the new page.

The block transfer engine allows a 16 KB page to be copied from one node's memory to another in under 30 microseconds. Unfortunately, in a very large Origin system, the cost to invalidate all the TLBs and update the translation using a conventional TLB shootdown algorithm can be 100 microseconds or more, removing much of the benefit of providing a fast memory to memory copy. Recent page migration research has also identified TLB shootdown as a significant cost of page migration[11]. To solve the TLB update problem, the directory supports a block transfer copy mode known as directory poisoning, which works as follows.

During the read phase of the poisoning block copy, in addition to reading the data, the directory protocol makes sure the latest copy of the data is written back to memory, and the directory is placed in the POISON state. Any access by a processor to a poisoned directory entry during the copy will result in a synchronous bus error. The bus error handler is able to quickly determine that the bus error was due to page migration, and the handler invalidates the processor's TLB entry for the page, and then has the process wait for the new translation to be produced.

Once the poisoning block copy has completed, the new translation is updated and all processors that took the poison bus error will load their TLB with the new translation. The poisoned page is now placed on a poisoned list to "age". The operating system invalidates one sequential TLB entry per scheduler tick, so after a time equal to the number of per-processor TLB entries times the period between scheduler ticks, the page can be moved off the poisoned list and onto the free list. This directory poisoning allows the TLB shootdown and page copy to proceed in parallel, and as a result the cost to migrate a page is much lower than if a standard TLB shootdown were invoked. This low cost of migration enables the operating system to be fairly aggressive in determining when to migrate a page.

4 Origin Performance

This section examines the performance of the Origin system using both microbenchmarks to measure latencies and bandwidths, and by using the NAS Parallel Benchmarks V2.2 and the SPLASH2 suite to measure performance of a set of parallel applications.

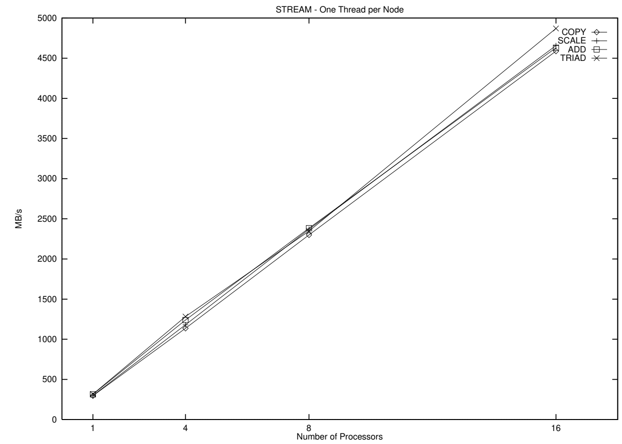

Figure 11 STREAM results - one thread per node

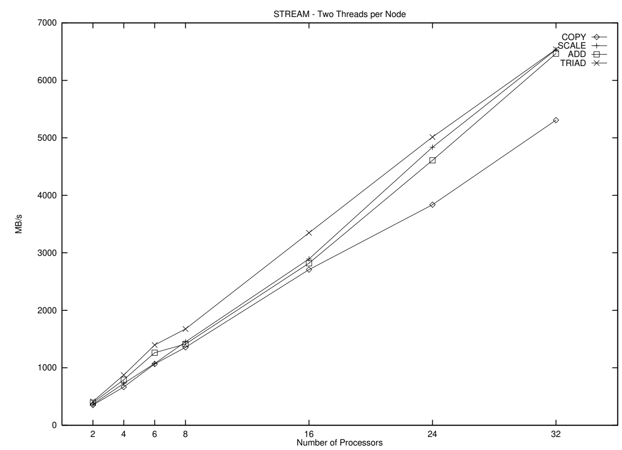

Figure 12 STREAM results - two threads per node

4.1 Microbenchmarks

The first microbenchmarks examine the latency and bandwidth of the Origin memory system. Table 4 shows the latency measured for a memory reference in isolation. This is the time from when the L1 cache is accessed until the instruction associated with the cache miss can graduate. The remote latency numbers for 16 and 32 processors assume that express links are employed.

The STREAM benchmark is the standard memory bandwidth benchmark for the high performance computing industry. STREAM measures the performance of four simple long vector kernels, which are sized to eliminate cache re-use, and reports the results in terms of sustained memory bandwidth. On parallel systems, STREAM is essentially completely parallel; the only communication required is for synchronization at the end of execution of each kernel.

On the Origin 2000, each processor can effectively utilize more than half of the memory bandwidth available on a node. Thus, we've included STREAM results in MB/s with only one thread running per node in Figure 11, and with two threads running per node in Figure 12.

Table 5 shows the effectiveness of the fetch-and-increment operations in implementing a globally shared counter. Note that LL/SC does much better for a single processor, at 6.9 million increments/second, since the counter variable and the lock surrounding it (which are allocated from the same cache line) stays loaded in the processor's cache, whereas the fetch-and-increment variable is always accessed via an uncached reference to local memory, and de-

M op/s	1 P	2 P	4 P	8 P	16 P	32 P
fch-inc	4.0	7.4	6.1	10.0	19.3	23.0
LL/SC	6.9	2.3	0.84	0.23	0.12	0.09

Table 5 Comparison of LL/SC and fetch-and-op for atomic increments

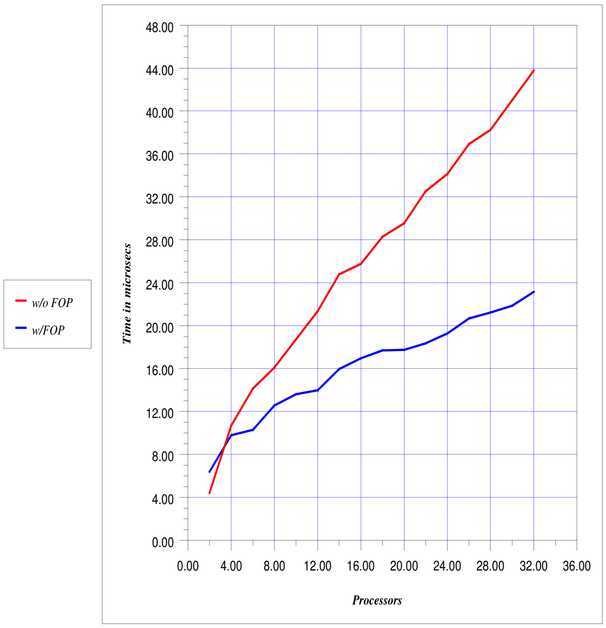

Figure 13 Comparison of LL/SC and fetch-and-op for a null doacross loop

livers only 4.0 million increments/sec. As more processors are added, however, the number of increments on the fetch-and-increment variable is able to increase to near the fetch-and-op cache throughput limit of 24.4 million/sec (one every 41 ns), while the number of increments on the LL/SC variable falls off dramatically, to under 100 thousand increments/second with 32 processors. Also note the drop off in fetch-and-op increments/second between two and four processors. With a small number of processors, the number of fetch-and-increments achievable per second is limited by the fetch-and-increment latency, since each R10000 processor can only have a single uncached read outstanding. Therefore with two processors, the fetch-and-increment can be allocated locally, whereas with four processors, only two of the processors can access the fetch-and-increment variable from local memory, and the other two processors must pay the longer remote latency.
Figure 13 shows the advantages of using fetch-and-op for barrier implementation. The graph shows the time to execute a null FORTRAN doacross statement. In addition to the barrier time, the null doacross includes the time to perform the work dispatch to the slaves and the execution of one iteration of an empty do-loop. Therefore the graph actually understates the performance benefits of fetch-and-ops in implementing the barrier itself.

4.2 Applications

In this section we examine the performance of the Origin system using the NAS Parallel Benchmarks V2.2 Class A and the

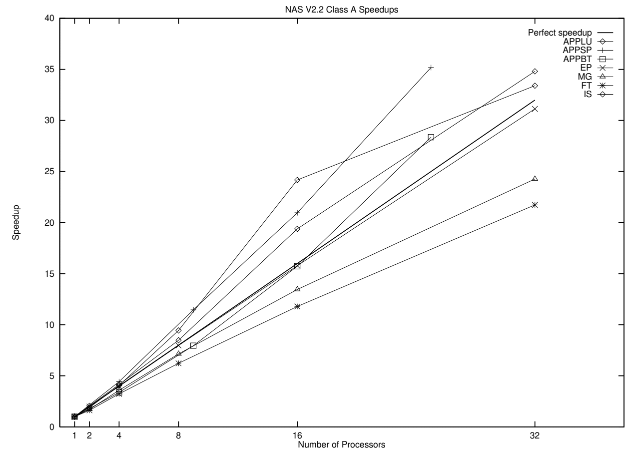

Figure 14 NAS Parallel V2.2 Speedups

Application	Command Line
radiosity	-batch -room
raytrace	balls4.env
lu	-n2048 -b16
ocean	-n 1026
barnes	< input.512

Table 6 SPLASH2 Applications

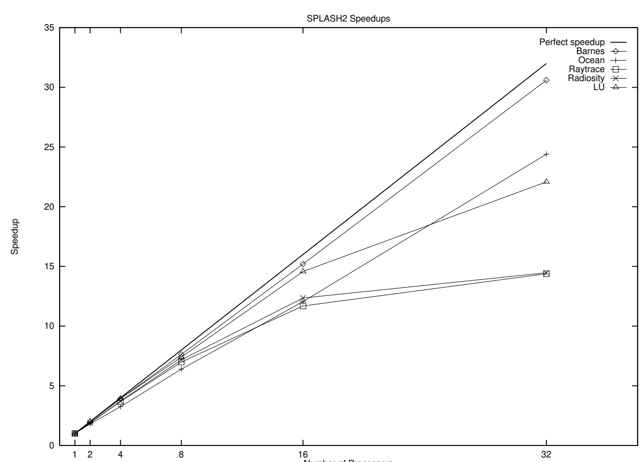

Figure 15 SPLASH2 Speedups

SPLASH2 suite. Application and operating system tuning is ongoing, so these results are a snapshot as of late February, 1997.
Figure 14 shows the performance of the NAS parallel benchmarks using the Class A datasets for up to 32 processors. Overall, speedup on the NAS benchmarks is very good. For several of the benchmarks, superlinear speedups are achieved due to the larger total cache size and memory bandwidth available as the number of processors (and therefore the number of nodes) increases.
Table 6 lists the applications run from the SPLASH2 suite along with their command line arguments. Figure 15 shows speedups for

the four SPLASH2 applications and one SPLASH2 kernel on the Origin system. Barnes and Ocean get good speedups to 32 processors, while Lu starts to roll off beyond 16 processors. Radiosity and Raytrace both begin to have speedup fall-off after 8 processors, and show very small increases between 16 and 32 processors. We have just started our investigation into the performance of the SPLASH suite, so we are not certain of the exact cause of the speedup roll-off for Lu, Radiosity, and Raytrace. Our benchmark machine had a limited amount of total memory, so the small data sets for these applications may be a contributor to the limited speedup at 32 processors.

5 Related Systems

The Origin system benefitted from the many lessons the authors learned in designing the Stanford DASH[7], so we start by discussing the major differences between Origin and DASH. We then contrast the Origin with three contemporary ccNUMA systems: the Sequent NUMAQ, the Data General NUMALiiNE, and the Convex Exemplar X.

5.1 Stanford DASH

The differences between the Origin and DASH coherence protocols was already explored in Section 3.2. The main architectural difference between the Origin and DASH systems is that DASH employed a four processor SMP cluster as its node[7], while Origin uses a two processor node where coherence between the processors in the node is handled by the directory-based protocol.

The major advantage of a SMP-based node is the potential for cache-to-cache sharing within the node. In [7] the 4-way intranode sharing of DASH is shown to produce small performance gains for 3 out of 4 SPLASH applications with only the Barnes application showing significant performance gains.

On the other hand, the disadvantages of the SMP-based nodes are three-fold. First, to get to the number of processors where intranode sharing will be significant the bus will most likely not be able to be on a single board. This causes local memory latency to be longer as the bus must run slower to support the large number of devices connected to it and more ASIC crossings will generally be between the processor and local memory. In addition, this makes the initial cost of the SMP node much higher since even a single processor node requires several boards. Second, remote latency also increases as requests to remote memory will generally have to wait for the results of the local processor snoops before being able to issue to the remote memory. Finally, as discussed earlier, the remote memory bandwidth for a SMP-based node is half the local memory bandwidth. In fact, in DASH, the remove memory bandwidth was reduced by a factor of three since each remote memory reference needed to traverse the local bus for the initial request, the home memory bus to get the data, and the local bus again to return the data to the processor.

While the DASH prototype did suffer increased latency on it's local memory access time due to the snoopy bus, it did manage to have a 3:1 best case (nearest-neighbor) remote to local latency, which as we will see when we examine some commercial contemporary systems is quite good for a SMP-based ccNUMA machine. Despite being much better than other SMP-based node solutions, this ratio is still less than the 2:1 best case (nearest-neighbor) remote to local latency achieved by Origin.

5.2 Sequent NUMAQ and DG NUMALiiNE

The Sequent NUMAQ consists of up to 63 nodes, where each node is a 4-processor Pentium Pro-based SMP referred to as a *quad*[8]. The nodes are connected together via an SCI-based ring. Both the low-level transport layers and the higher-level coherence protocol of the SCI are utilized in the NUMAQ. A programmable protocol engine is used to implement the SCI coherence protocol. A full board (the Lynx Board) implements the complete interface between the Pentium Pro based quad.

The Sequent NUMAQ is architecturally similar to the Stanford DASH, albeit with a different processor, a simpler network topology, and the SCI coherence protocol instead of the DASH coherence protocol. It has the same advantages and disadvantages of DASH: the local memory latency is good, around 250 ns, but the best case remote memory latency is around 8 times the local latency[8]. The choice of a ring with its low bisection bandwidth as the interconnect network causes large degradation in remote latency as the system interconnect becomes loaded.

The Data General NUMALiiNE is architecturally very similar to the Sequent NUMAQ. The node is a Pentium-Pro quad, and the nodes are connected via a SCI-based ring. As such it suffers from the same limitations in remote latency and network performance as the NUMAQ.

5.3 Convex Exemplar X

The Convex Exemplar X is similar to earlier Exemplar systems implementing a crossbar connected set of hyper-nodes that are then connected by parallel ring interconnects that implement a modified version of the SCI protocol[2]. In the X-class machines the hypernode size has increased from 8 to 16 processors and the four 1-D rings have been replaced by eight sets of 2-D rings. Initial configurations support 64 processors (4 hypernodes), but the machine can architecturally scale to 512 processors in an 8x4 torus configuration.

The use of a crossbar intra-connect for the hypernode does reduce the bandwidth penalty of using an SMP node compared with the NUMAQ or NUMALiiNE machines, but still adds to latency in comparison with the smaller, more integrated nodes in Origin. While no latency data has been published to date, the ratio of local to remote in the Exemplar is likely to be similar to previous machines (5:1 without loading on small configurations), whereas the base numbers in the Origin start at 2:1 in small configurations and grow very slowly.

Another major difference is the use of a third-level cluster cache in the Exemplar. This mechanism is in contrast to Origin's page migration mechanism. The cluster cache can adapt to capacity misses more quickly then Origin's migration mechanism, but does this at the cost of increased latency for communication misses and misses that result from conflicts in the cluster cache. The cluster cache also hurts remote bandwidth because it implies that at least three DRAM accesses be made per remote miss (one in the local cluster cache, one at the home, an additional one or two if the line is held dirty in another cluster cache, and a final access to allocate into the local cluster cache), leading to remote bandwidth being one-third of the local bandwidth.

5.4 Overall Comparison of DSM Systems

The major difference between the Convex, Sequent and DG machines and the Origin is that the Origin has a much more tightly integrated DSM structure with the assumption of treating local accesses as an optimization of a general DSM memory reference. The other commercial DSM machines add a DSM mechanism on top of a relatively large SMP node in which local communication and memory accesses are optimized over performance of the general DSM structure. Which one of these models is more appropriate depends upon a number of factors:

1. Is scaling the system down in size and cost important? The overhead of the SMP nodes sets a minimum on how effective such machines can be in small configurations (one to two processors).

2. Is the workload primarily throughput oriented, with only a small degree of parallelism? If so, the SMP solutions might achieve higher performance due to lower communication costs between processors within the same SMP node. Oppositely, if parallelism beyond the size of the SMP node is important, then the latency and bandwidth overheads are likely to be very high in comparison to Origin.
3. Is I/O structured as a global resource or must nodes treat I/O as a local resource similar to a distributed memory system? If the goal is global accessibility then the tight DSM integration of Origin would also be preferred.

6 Conclusions

The Origin 2000 is a highly scalable server designed to meet the needs of both the technical and commercial marketplaces. This is accomplished by providing a highly modular system with a low-entry point and incremental costs. A bristled fat hypercube network is used to provide a high bisection bandwidth, low-latency interconnect. Low latency to local memory and a low remote to local memory latency ratio allow the existing application base to easily migrate their applications from the uniform access of the existing SMP Challenge and Power Challenge systems to the NUMA Origin systems. Origin also includes several features to help the performance of these applications including hardware and software support for page migration and fast synchronization.

7 Acknowledgments

The Origin system design resulted from the very hard work of a top-notch team of chip, board, and system engineers. Major contributors on the core logic design/verification team besides the authors included: Michael Anderson, John Andrews, Rick Bahr, John Burger, John Carlson, Hansel Collins, Pat Conway, Ken Choy, Asgeir Eiriksson, Paul Everhardt, Mike Galles, Sameer Gupta, Gary Hagensen, Dick Hessel, Roger Hu, Lee Jones, George Kaldani, John Keen, Ron Kolb, Yuval Koren, Waidy Lee, Viranjit Madan, John Manton, Greg Marlan, Dawn Maxon, David McCracken, Ali Moyedian, Bob Newhall, Kianoosh Naghshineh, Chuck Narad, Ron Nikel, Steve Padnos, Dave Parry, Ed Priest, Azmeer Salleh, Ken Sarocky, Chris Satterlee, Alex Silbey, Doug Solomon, Jim Smith, Tuan Tran, Swami Venkataraman, Rich Weber, Eric Williams, Mike Woodacre, and Steve Yurash.

The authors would also like to thank Jaswinder Pal Singh and Dongming Jiang for their help in getting the SPLASH performance numbers on Origin, John McCalpin for providing the NAS Parallel Benchmark and STREAM results, and the anonymous referees for their comments which helped to improve the final version of this paper.

References

[1] Anant Agarwal, Ricardo Bianchini, David Chaiken, Kirk L. Johnson, David Kranz, John Kubiatowicz, Beng-Hong Lim, Kenneth Mackenzie, and Donald Yeung. The MIT Alewife machine: Architecture and Performance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 2-13, June 1995.

[2] Tony Brewer and Greg Astfalk. The evolution of the HP/Convex Exemplar. In *Proceedings of COMPCON Spring '97: Forty-Second IEEE Computer Society International Conference*, pages 81-86, February 1997.

[3] Asgeir Th. Eiriksson and Ken L. McMillan, Using formal verification/analysis methods on the critical path in system design: A case study. In *Proceedings of Computer Aided Verification Conference*, Liege Belgium, LNCS 939, Springer Verlag, 1995.

[4] Mike Galles. Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER chip. In *Hot Interconnects '96.*

[5] Mark Heinrich, Jeffrey Kuskin, David Ofelt, John Heinlein, Joel Baxter, Jaswinder Pal Singh, Richard Simoni, Kourosh Gharachorloo, David Nakahira, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, and John Hennessy. The performance impact of flexibility in the Stanford FLASH multipocessor. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 274-285, October 1994.

[6] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 148-159, May 1990.

[7] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, and John Hennessy. The DASH prototype: Logic overhead and performance. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):41-61, January 1993.

[8] Tom Lovett and Russell Clapp, STiNG: A CC-NUMA computer system for the commercial marketplace. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 308-317, May 1996.

[9] Steven K. Reinhardt, James R. Larus, and David A. Wood. Tempest and Typhoon: User-level shared memory. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 325-336, April 1994.

[10] Mendel Rosenblum, John Chapin, Dan Teodosiu, Scott Devine, Tirthankar Lahiri, and Anoop Gupta. Implementing efficient fault containment for multiprocessors. *Communications of the ACM*, 39(3):52-61, September, 1996.

[11] Ben Verghese, Scott Devine, Anoop Gupta, and Mendel Rosenblum. Operating system support for improving data locality on CC-NUMA compute servers. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 279-289, October 1996.

[12] Wolf-Dietrich Weber, *Scalable Directories for Cache-Coherent Shared-Memory Multiprocessors*. Ph.D.thesis, Stanford University, Stanford, California, January 1993.

[13] Steve Whitney, John McCalpin, Nawaf Bitar, John L. Richardson, and Luis Stevens. The SGI Origin software environment and application performance. In *Proceedings of COMPCON Spring '97: Forty-Second IEEE Computer Society International Conference*, pages 165-170, February 1997.

[14] Kenneth Yeager, The MIPS R10000 Superscalar Microprocessor, *IEEE Micro*, 16(2):28-40, April, 1996.