White Paper

The SGI® Altix™ 3000 Global Shared-Memory Architecture

Michael Woodacre, Derek Robb, Dean Roe, and Karl Feind

Abstract

This paper describes the global shared-memory architecture and benefits of the SGI Altix 3000 family of servers and superclusters. Memory size and scaling, cache organization and coherency, nodes and internode memory access, bandwidths and latencies, and RAS features are all discussed. The paper is written from a perspective that may be useful to an applications developer or a system administrator. It describes the current implementation of the Altix 3000 system communication infrastructure.

1.0 Introduction

SGI Altix 3000 is a cache-coherent, shared-memory multiprocessor system. It is based on the proven SGI® NUMAflex™[1] system architecture used in SGI® Origin® 3000 systems, with enhancements to provide new levels of performance. While Origin® systems are based on the MIPS® microprocessor and the IRIX® operating system, the Altix 3000 family combines NUMAflex with industry-standard components—the Intel® Itanium® 2 microprocessor and the Linux® operating system.

The Altix 3000 family enables new capabilities and time-to-solution breakthroughs that neither traditional Linux OS-based clusters nor competitive SMP architectures can tackle, by holding more complex job geometries and complete workflows in memory. Altix 3000 systems dramatically reduce the time and resources required to run technical applications by managing extremely large data sets in a single, system-wide, shared-memory space called global shared memory.

Global shared memory means that a single memory address space is visible to all system resources, including microprocessors and I/O, across all nodes. Systems with global shared memory allow access to all data in the system's memory directly and extremely quickly, without having to go through I/O or networking bottlenecks. Systems with multiple nodes without global shared memory instead must pass copies of data, often in the form of messages, which can greatly complicate programming and slow down performance by significantly increasing the time processors must wait for data. Global shared memory requires a sophisticated system memory interconnect like NUMAlink™ and application libraries that enable shared memory calls, such as Message Passing Toolkit (MPT) and XPmem from SGI.

2.0 NUMAflex

NUMAflex uses an SGI® NUMA (cache-coherent, nonuniform memory access) protocol implemented directly in hardware for performance and a modular packaging scheme. NUMAflex gets its name from the flexibility it has to scale independently in the three dimensions of processor count, memory capacity, and I/O capacity. The key to the NUMAflex design of Altix is a controller ASIC, referred to as the SHUB, that interfaces to the Itanium 2 front side bus, to the memory DIMMs, to the I/O subsystem. and to other NUMAflex components in the system.

Altix 3000 is built from a number of component modules, or bricks, most of which are shared with Origin 3000. The C-brick (compute brick) is the module that customizes the system to a given processor architecture. The Altix C-brick consists of four processors connected to two SHUBs and up to 32GB of memory implemented on two equal "node" boards in a 3U brick. A schematic diagram of the C-brick is shown in figure 1.

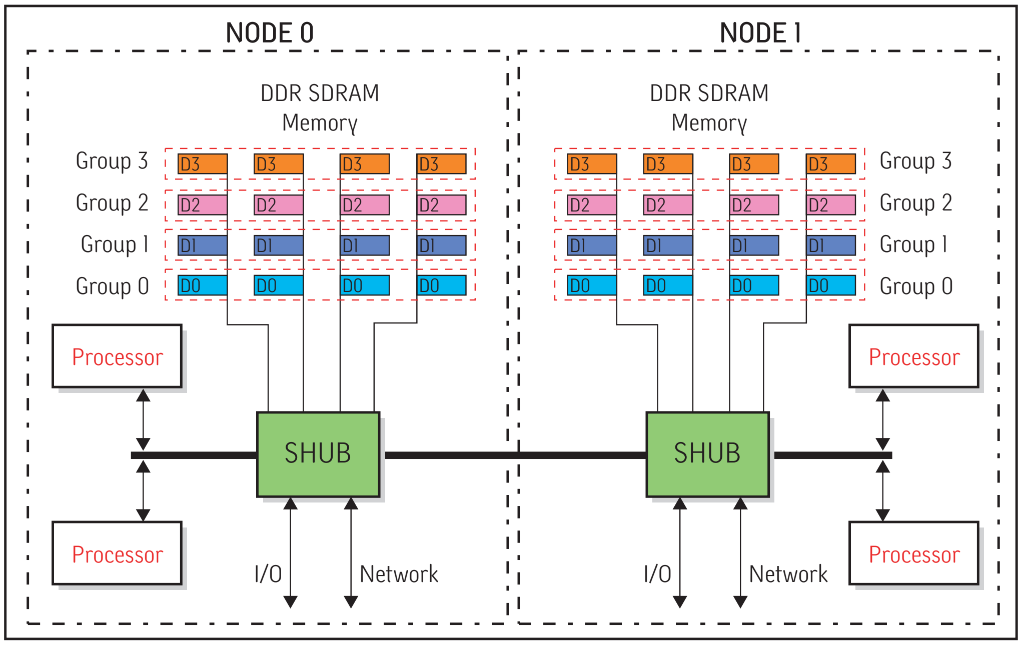


Fig. 1. Altix 3000 C-brick schematic

The M-brick (memory brick) is essentially a C-brick without the processors. An M-brick can be placed in any location in the interconnect fabric that could be occupied by a C-brick and thus allows the system to scale memory without adding processors up to multi-terabyte shared memory domains for a single node. The remaining components of the system are the R-brick (an 8-port NUMAlink™ 3 router brick), which is used to build the interconnect fabric between the C-bricks and M-bricks; the IX-brick (the base IO brick) and the PX-brick (a PCI-X expansion brick), which attach to the C-brick via the I/O channel; and the D-brick2 (a second-generation JBOD brick). SGI supplies a variety of networking, Fibre Channel SAN, RAID, and offline storage products to complete the Altix 3000 offering.

3.0 Intel Itanium 2 Microprocessor

The Intel Itanium 2 64-bit microprocessor brings industry-leading uniprocessor performance levels to the Altix 3000 family. The NUMAflex architecture allows the system to scale applications performance up to 512 processors, all working together in a cache-coherent manner. Intel is providing a family of socket-compatible processors in the Itanium 2 family[2], so shortly after Intel delivers a new member of the processor family, it is expected to be made available on the Altix 3000 platform, with the intent that customers will remain at the leading edge of microprocessor performance. Itanium 2 has a processor bus that supports 6.4GB per second of bandwidth into the system.

Altix 3000 has been optimized for the demands of high-performance applications. While the Itanium 2 processor supports up to four processors being placed on a bus, Altix 3000 places just two processors on a bus, so it can provide twice the memory bandwidth of other systems on a per-processor basis.

When the data required by the processor is not in one of the on-die caches shown in figure 2, the processor sends a request for a cache line of data from the global shared memory. When two or more processors are operating independently on the same data, the Altix 3000 hardware will keep the data coherent without software intervention.

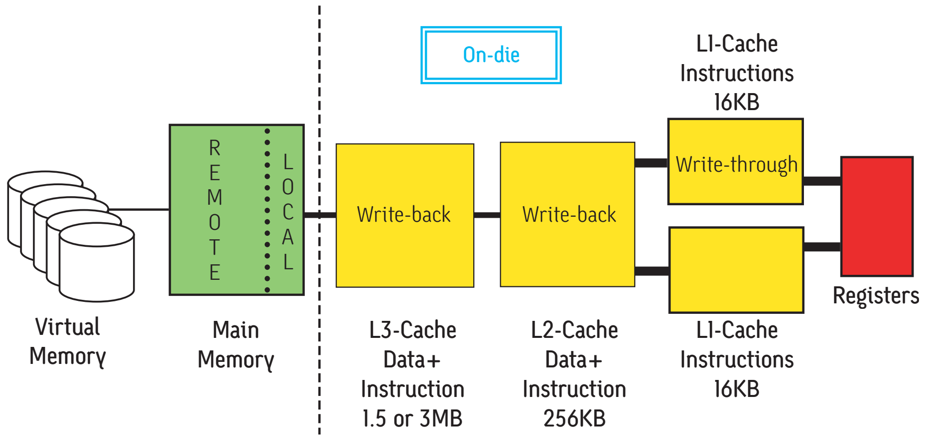

On-die
L1-Cache Instructions 16KB
Write-through
R E M O T E
L O C A L
Write-back
Write-back
Registers
Virtual Memory
Main Memory
L3-Cache Data+ Instruction 1.5 or 3MB
L2-Cache Data+ Instruction 256KB
L1-Cache Instructions 16KB

Fig. 2. Itanium 2 memory and cache hierarchy diagram

4.0 Cache Coherency

The cache-coherency protocol on the Altix 3000 family is implemented in the SHUB ASIC, which interfaces to both the snooping operations of the Itanium 2 processor and the directory-based scheme used across the NUMAflex interconnect fabric. If a processor request, can be satisfied by the contents of its neighbors cache, data will flow directly from one processor cache to the other processor without the extra latency of sending the request to memory.

With snoop-based systems, every transaction must be made visible to all the processors in the system to maintain cache coherence, and so system overhead becomes an increasing problem for large numbers of processors. However, with directory-based cache-coherent systems, only the processors that are currently playing an active role in the usage of a given cache line need to be informed about an operation to that cache line. This reduces the amount of information that is needed to flow around the system to maintain cache coherence, resulting in lower system overhead, reduced latency, and higher delivered bandwidth for actual data operations.

In the past decade, “Beowulf” cluster systems have become popular because of the attraction of their low up-front hardware costs and the availability of open-source software. In such cluster systems, cache coherence is the responsibility of the user, typically requiring copying of data over commodity interconnects with the associated performance issues that involves. Altix 3000 enables the user to get direct access to cache-coherent load/store semantics to share data across the high-performance NUMAflex network.

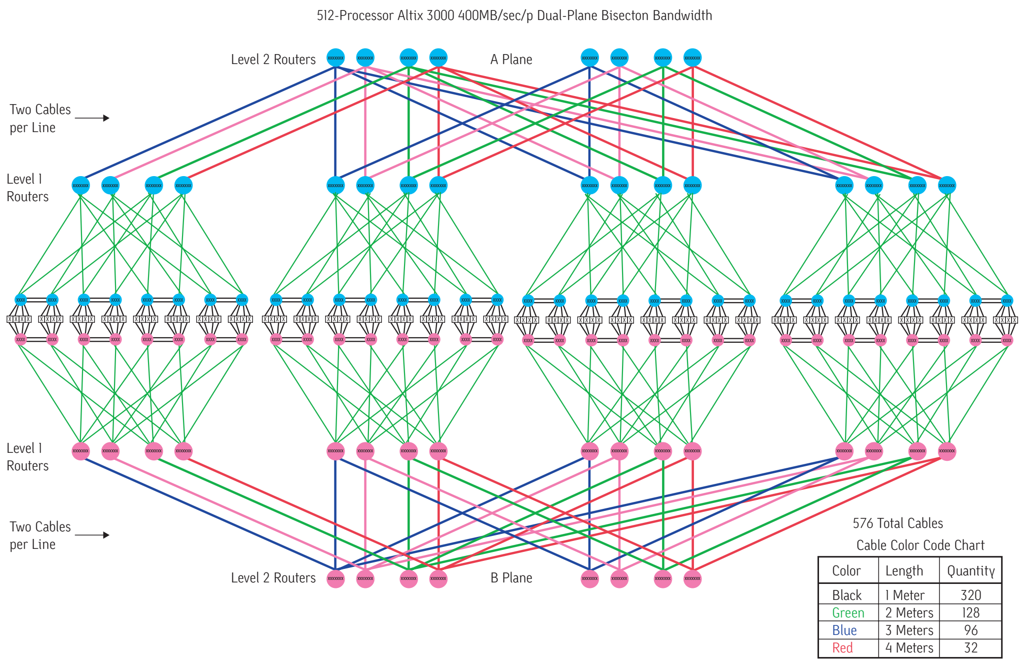
5.0 Interconnection Network

Altix 3000 makes use of a new advance in NUMAflex technology, the NUMAlink™ 4 communications channel. Prior generations of NUMAlink have been employed in SGI® scalable systems. NUMAlink 4 builds on this low-latency, high-bandwidth heritage. NUMAlink 4 provides double the bandwidth of NUMAlink 3 while maintaining compatibility with NUMAlink 3 physical connections. NUMAlink 4 is able to achieve this performance boost by employing advanced bidirectional signaling technology.

The NUMAflex network for Altix is configured in a fat tree topology. Figure 3 shows this topology for a 512-processor configuration. The circles in the figure represent R-bricks, the lines represent NUMAlink cables, and the 128 small squares across the center of the diagram represent C-bricks.

The fat tree topology enables the system performance to scale well by providing a linear increase in bisection bandwidth as the systems increase in size. Altix 3000 also provides dual-plane or parallel networks for increased bisection bandwidth. This dual-plane configuration is made possible by providing two

NUMAlink ports on the Altix C-brick. With Altix initially being deployed using the NUMAlink 3 router brick, the system will be able to double the bisection bandwidth when the NUMAlink 4 router brick becomes available, allowing the systems' capabilities to grow along with the demands of new generations of Itanium 2 family microprocessors.



Color	Length	Quantity
Black	1 Meter	320
Green	2 Meters	128
Blue	3 Meters	96
Red	4 Meters	32

Fig. 3. 512-processor dual "fat tree" interconnect topology

The main memory characteristics of Altix 3000 are given in table 1. The two numbers in the bandwidth-per-processor column correspond to NUMAlink 3 and NUMAlink 4. The two numbers in the maximum local memory column correspond to using 512MB and 1GB DIMMs.

Because memory DIMMs can be added to the system using M-bricks, the memory can be scaled independent of processors. Hence it is entirely possible to build a system with 16 processors and 4TB of shared cache-coherent memory.

Table 1. System and Local Memory Characteristics of Altix 3000

# of SHUBs or nodes	Maximum # of processors	NUMAlink 3 Bandwidth MB/sec/p	NUMAlink 4 Bandwidth MB/sec/p	Maximum Local Memory w/512MB DIMMs*	Maximum Local Memory w/1GB DIMMs*	Maximum Local Memory w/2GB DIMMs*	# of Routers	Maximum # of Router Hops
8	16	800	1600	64GB	128GB	256GB	2	3
16	32	800	1600	128GB	256GB	512GB	4	4
32	64	400	800	256GB	512GB	1TB	8	4
64	128	400	800	512GB	1TB	2TB	20	5
128	256	400	800	1TB	2TB	4TB	40	5
256	512	400	800	2TB	4TB	8TB	112	7
512	1024	400	800	4TB	8TB	16TB	288	10
1024	2048	400	800	8TB	16TB	32TB	576	10

*M-bricks can be used to scale memory independently up to 128GB per CPU.

The physical address space is split into a memory address of 36 bits (architectural limit of 64GB per SHUB), and compute node address of 10 bits (1K nodes, or 2K processors). While the initial system is capable of coherently sharing cache lines among up to 512 processors, it is possible to build a single NUMAflex network with globally upgradeable memory of up to 2,048 processors. Communications among the four 512-processor sharing domains on the NUMAflex network use coherent I/O semantics for moving data between the sharing domains, while still utilizing the low-latency, high-bandwidth characteristics of NUMAlink. Future enhancements may increase the size of the coherence limit to thousands of processors.

Previous generations of NUMAflex systems used proprietary memory DIMMs to provide data storage and the additional directory storage for tracking cache coherence within the system. The Altix 3000 family breaks away from this limitation and embraces industry-standard DIMM technology, providing the economic benefits of this standard technology without sacrificing performance.

The memory subsystem of Altix 3000 uses PC-style double data rate (DDR) SDRAM DIMMs, shown in table 2. Each SHUB ASIC in a C-brick supports four DDR buses. Each DDR bus may contain up to four DIMMs (each DIMM is 72 bits wide, 64 bits of data and 8 bits of ECC). The four memory buses are independent and can operate simultaneously to provide up to 12.8GB per second of memory bandwidth.

Table 2. DDR SDRAM Memory used on Altix 3000

DIMM Type	DDR Speed	DIMM Size	Local Memory Bandwidth
PC2100	266 MHz	512MB, 1GB or 2GB	8.5GB/sec
PC2700	320 MHz	512MB, 1GB or 2GB	10.2GB/sec
PC3200	400 MHz	512MB, 1GB or 2GB	12.8GB/sec

The system supports the use of PC2100 (DDR—266 MHz), PC2700 (DDR—320 MHz), and PC3200 (DDR—400 MHz) DIMMs. The aggregate memory bandwidths using these DIMMs are 8.5GB per second and 10.2GB per second respectively. Using 256Mb DRAM technology yields a per-DIMM capacity of 512MB. This gives each node a base capacity of 8GB, 4GB per processor. One-gigabyte DIMM technology will provide a capacity of 32GB per C-brick or 8GB per processor. Different DIMM sizes may be mixed in a C-brick but must be added in sets of eight DIMMs at a time. The system is designed to accommodate 2GB DIMMs when they become commercially available.

Each SHUB ASIC contains a directory cache for the most recent cache-coherency state information. This allows for efficient utilization of the memory subsystem for performing data operations, by minimizing the amount of memory bandwidth that is needed to look up cache-coherency state information.

When the system is booted, it sets aside approximately 3% of the memory space to store the cache-coherency directory information (in comparison, typical memory structures set aside 12% of memory space to store error-detection and correction codes) for the system. This directory space is used to store directory information that is not being actively used in the directory cache. The directory information is stored in parallel with the cache-line data, but on a different DIMM bus. So if the directory state for a memory reference is not currently available in the on-chip directory cache, the directory information can be read from memory at the same time as the data (which resides on a different DIMM bus). This optimized storage scheme ensures that the system can achieve maximum delivered data bandwidth by minimizing bus usage conflicts.

While the local processor bus has a peak bandwidth of 6.4GB per second, the local memory subsystem has enough bandwidth to fully saturate the local processor demands while leaving available bandwidth to service remote processor and I/O memory requests.

In addition to providing support for cacheable memory, the memory subsystem also supports atomic in-memory operations (AMO) that may be used to provide fast, scalable communication primitives such as locks and barriers. These AMOs don't require a full cache line of data to ping around between different processor caches; such pinging around causes high contention that in turn leads to low performance.

6.0 Reliability, Availability, and Serviceability (RAS)

The Altix 3000 family was designed to provide a robust operating environment. The design protects data flowing around the system using a number of techniques.

Memory is protected by an error-correcting code that can correct all single-bit errors, detect multiple-bit errors, and provide chip-kill error detection.

The NUMAlink channels protect the messages flowing across the channels with a CRC protection code. If a CRC error is detected at the receiving end of a NUMAlink channel, the message is resent, enabling the system to provide reliable communications. The dual-plane NUMAlink interconnect fabric also provides enhanced availability since the system is designed to remain fully operational if one of the planes fails.

The Itanium 2 microprocessor has ECC protection on caches and ECC is used on the processor bus to detect and correct single-bit errors and detect multiple-bit errors.

The system can run multiple nodes of the Linux kernel, so even if one node suffers a fatal error, other nodes may continue operating while the failed node is repaired or rebooted.

The Altix 3000 packaging provides N+1 hot-swap fans and N+1 redundant power supplies on each of the C-bricks, R-bricks, and I/O bricks in the system.

7.0 Peer I/O

The cache-coherence protocol has been designed to accommodate future generations of I/O bricks directly in the NUMAlink fabric. This "peer-I/O" capability will enable NUMAflex systems to scale I/O independently of processing capabilities and thus configure to the exact requirements that a user may need, without the cost of surplus infrastructure.

8.0 Linux Scaling

The standard Linux operating system has been scaled up to the new peak of 64 processors within a single system image on the Altix 3000 family. In a traditional cluster, each host must be provisioned with the maximum memory that any process may ever need. In addition, there is also the need for enough memory to run a copy of the operating system on each host. If a process wants more memory than has been provisioned on a single host, the code needs to be reworked to spread that load across multiple hosts (if that is even possible). With a large host size, a larger pool of memory is available to each individual process running on that host. For example, a 64-processor system, running a single 64-processor Linux kernel will have up to 1TB of memory available for a single process to use. This gives application developers an extraordinarily large sandbox to work in, so they can concentrate on the demands of their applications without the need to worry about arbitrary node-configuration limits.

Another major benefit of the Altix 3000 system is the ability to share user memory for load/store access across the entire system, comprising multiple hosts all connected to the same NUMAlink interconnect. Using interfaces like SHMEM and XPMEM that provide sharing of memory segments across hosts, a user application can approach work-sharing and parallelism problems with direct memory load/store access methods previously available only on small SMP systems with System V and similar shared-memory constructs.

9.0 Internode Access (XPMEM and XPNET)

SGI NUMAflex architectures have the capability of having multple nodes that are independent systems, each running its own copy of the operating system. The physical memory of an Altix 3000 system can be separated via firewalls, which can be raised or lowered to prevent or allow memory, CPU, and I/O accesses to be made across the node boundary by processes on the other side.

SGI NUMAflex architectures also contain block transfer engines (BTEs) which can be viewed as cache-coherent DMA engines. BTEs reside on the SHUB ASICs and are used to copy data from one physical memory range to another at very high bandwidth. Once initiated, BTE data transfers do not require processor resources, and they can be performed across node boundaries.

Internode memory access allows users to access memory belonging to processes on the same Altix 3000 system. This memory can reside within the same or on a separate node. Memory can be accessed by data copies utilizing the BTE or by directly sharing the underlying physical memory. The software stacks that allow internode shared-memory (XPMEM) and networking (XPNET) software layers to be built for SGI Linux™ are shown in figure 4.

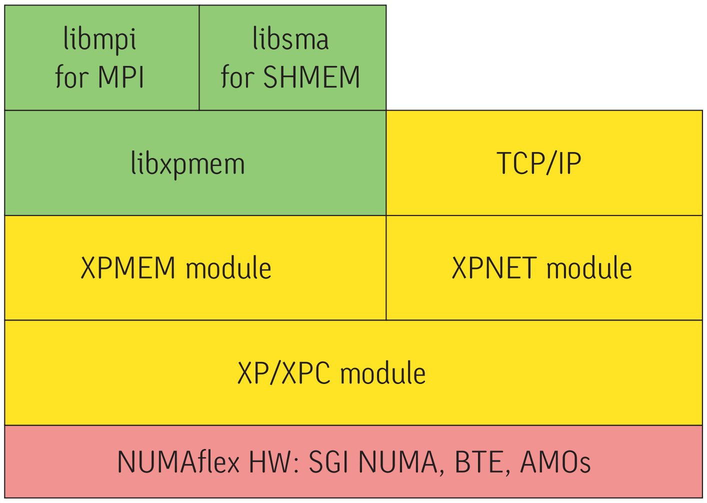


Fig. 4. Software stacks for XPMEM and XPNET

The XP and XPC kernel modules provide a reliable and fault-tolerant internode communication channel that is used to transfer data over the NUMAlink interconnect. XPNET utilizes the NUMAlink interconnect to provide high-speed TPC and UDP protocols for applications.

Internode memory access is provided to user applications via the libxpmem user library and the XPMEM kernel module. XPMEM allows a source process to make regions of its virtual address space accessible to processes within or across node boundaries. The source process can define a permission requirement to limit which processes can access its underlying memory. Other processes can then request access to this region of memory and are allowed to attach to it if they satisfy the permission requirement. Once the underlying physical memory is attached and faulted, the remote process operates on it via cache-coherent loads and stores just as the source process does.

XPMEM locks physical pages shared across node boundaries in memory so they cannot be swapped. This is done dynamically by the XPMEM kernel module and only when a physical page is first used across node boundaries. Prior to that, the physical page is not required to be locked in memory.

The SGI® Message Passing Toolkit[3] (MPI + SHMEM) is optimized to use XPMEM via the process-to-process interfaces outlined above. Future enhancements for XPMEM may include interfaces similar to existing System V shared-memory interfaces. Use of XPMEM interfaces across cache-coherence domains (via non-cached shared memory, high-speed memory-to-memory data copies, etc.) is also being explored.

Open-source message-passing library implementations, such as MPICH[4] and LAM-MPI[5], that are based on the IP protocol would build on top of XPNET.

10.0 The XPMEM User API

The XPMEM API consists of a library (libxpmem) that abstracts the underlying kernel module implementation from the end user.

The xpmem_make() library function provides the user process with a unique handle that represents a user-specified segment of its address space. It is the responsibility of the user process to share its handle with other processes if external access to that address segment is desired.

Once a process has knowledge of a handle, it must ensure that it has the proper credentials required to access the address segment represented by the handle. If the xpmem_get() library function succeeds, an ID is returned to the process that can then be used to access the segment. The xpmem_get() operation is performed before, and separate from, the actual access operations, so that credential checks need only be done once and not on every access operation.

Once the necessary ID(s) are known, the process can use various library functions such as xpmem_copy() and xpmem_attach() to perform the desired memory-access operation. xpmem_copy() can be used to copy data from one address segment to another, utilizing the BTE when possible. xpmem_attach() can be used to attach data from another process' address segment to the calling process' address space so that the underlying physical memory is actually shared between the processes.

The XPMEM API provides a convenient low-level mechanism on which to build higher-level APIs that provide the end user with memory-sharing capabilities. In addition to optimized MPI send/receive, MPI and SHMEM put/get, and SHMEM remote pointers (see below), users could explicitly share access to System V segments or distributed-memory regions mapped into a single contiguous range of virtual memory.

11.0 Code Sample for Shared-Memory Pointers

The following simple working program shows how to use shmem_ptr to access a data structure that resides in a remote process' memory. This is accomplished by using XPMEM from within libsma (SHMEM). In this program, MPI process 0 stores 100 integers directly into the bigd array that resides in MPI process 1, even if process 1 is running on a different host in the same Altix 3000 system. A virtual address pointing to the remote bigd array is returned by shmem_ptr(). Then the process simply stores the data at that address. A barrier synchronization is performed before process 1 reads the data that was delivered. The power and ease of use of shmem_ptr() makes it an interface worth investigating by code developers.

```
  #include <mpi.h>
 #include <mpp/shmem.h>
 main(ing argc, char **argv)
 {
        static int bigd[100];
        int *ptr;
        int i;

        MPI_Init(&argc, &argv);

        if (_my_pe() == 0) {     /* MPI rank 0 */
               /* initialize PE 1's bigd array */
               ptr = shmem_ptr(bigd, 1);
               for (i=0; i<100; i++)
                      *ptr++ = i+1;
        }

        shmem_barrier_all();

        if (_my_pe() == 1) {     /* MPI rank 1 */
               printf("bigd on PE 1 is:\n");
               for (i=0; i<100; i++)
                      printf(" %d",bigd[i]);
               printf("\n");
        }
MPI_Finalize();
 }
```

12.0 Conclusion

Altix 3000 is a powerful new server and supercluster architecture for HPC applications. With a fast, extensible, globally shared, cache-coherent memory and an array of APIs, users will be able to solve computational problems of greater mathematical complexity at finer resolution in a shorter time.

13.0 References

1. Intel press announcement, January 16, 2003: *www.infoworld.com/articles/hn/xml/03/01/16/030116hnitanium.xml?s=IDGNS*
2. MPI Programmer's Manual, MPT Release Notes, mpi man page and intro_shmem man page: *http://techpubs.sgi.com*
3. *www-unix.mcs.anl.gov/mpi/mpich*
4. *www.lam-mpi.org/*