Cray Cascade: a Scalable HPC System based on a Dragonfly Network

Greg Faanes, Abdulla Bataineh†, Duncan Roweth, Tom Court, Edwin Froese,
Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins and James Reinhard
Cray Inc, Chippewa Falls, Wisconsin, USA

***Abstract*— Higher global bandwidth requirement for many applications and lower network cost have motivated the use of the Dragonfly network topology for high performance computing systems. In this paper we present the architecture of the Cray Cascade system, a distributed memory system based on the Dragonfly [1] network topology. We describe the structure of the system, its Dragonfly network and the routing algorithms. We describe a set of advanced features supporting both mainstream high performance computing applications and emerging global address space programing models.**

We present a combination of performance results from prototype systems and simulation data for large systems. We demonstrate the value of the Dragonfly topology and the benefits obtained through extensive use of adaptive routing.

Keywords: Interconnect technologies, parallel and scalable system architectures, performance evaluation and measurement of real systems.

I. INTRODUCTION

The Cray Cascade system was developed as part of Cray's DARPA High Productivity Computing System program. Cascade is a distributed memory architecture, its network provides programmers with global access to all of the memory of their parallel application. Aries[1] is the Cray network ASIC, developed as part of Cascade to provide substantial improvements on all of the standard network performance metrics for high performance computing: bandwidth, latency, message rate and scalability.

In addition, the Cascade design addresses one of the key challenges in high performance networking today: providing scalable global bandwidth at a reasonable cost. The ability to efficiently execute complex communication patterns, such as all-to-all, broadens the range of applications that will perform well on Cascade systems. The combination of low latency and high global bandwidth makes Cascade ideally suited for strong scaling problems that scale poorly on current systems.

The Cascade system has a number of attractive attributes. The system can be configured to match the users' global bandwidth requirements and cost constraints by reducing the number of expensive optical connections. The adaptive routing technology within Aries is designed to efficiently utilize the reduced number of connections. Cascade extends the small address space of general purpose processors to efficiently access all the physical memory in the system. Aries provides a suite of communication and synchronization mechanisms: block transfer engines for asynchronous pipelined memory copies, atomic memory operations, barriers, destination-side synchronization, message delivery doorbells and collective combining trees. The network protocol is designed to transfer 8-byte messages at very low overhead. Finally, Cascade provides a rich set of static and dynamic adaptive routing algorithms that can be utilized at the packet level.

The Cray software stack uses the advanced features of the Cascade network to provide highly optimized implementations of standard programming environments such as MPI and SHMEM, together with novel Partitioned Global Address Space (PGAS) programming models such as CoArray FORTRAN (CAF), Universal Parallel C (UPC) and Chapel. A single Cray Linux Environment (CLE) release supports both the Cray XE6 system and the new Cascade system. System specific features are hidden behind standard application interfaces ensuring a high degree of compatibility which makes application porting between systems straightforward.

In this paper, we present the architecture of the Cray Cascade system. In Section II we describe the node organization, network topology, system packaging and system configurations. Section III describes routing. Section IV describes memory architecture and address translation. Section V outlines network RAS features. Section VI describes the Cray software stack available on the Cascade system. Section VII highlights prior related work. Section VIII presents preliminary performance results. Finally, Section 0 summarizes the key attributes of the Cascade system architecture.

II. SYSTEM OVERVIEW

This section describes the node organization, network topology, system packaging, and system configurations.

A. Node Organization

The Cascade system is constructed from blades that contain four compute nodes and a single Aries interconnect ASIC (Fig. 1). The Aries ASIC provides the network connectivity for all four nodes on the blade. Aries uses a system-on-a-chip design that combines four high performance

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001.

† Abdulla Bataineh is also affiliated with Jordan University of Science and Technology, Irbid, Jordan.

[1] ARIES is a trademark owned by Intel, and is used by permission of Intel.

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

Network Interface Controllers (NICs) and a 48-port high radix router.

Figure 1. A Cascade blade which consists of 8 Xeon sockets, organized as 4 dual-socket nodes, and a single Aries ASIC.

Each of the four Aries NICs provides an independent PCI-Express Gen3 x16 host interface. In the first Cascade blade design these interfaces connect to four independent dual socket Xeon nodes. Each node has a pair of Sandy Bridge or Ivy Bridge processors and one SouthBridge. There are 8 DDR-3 memory channels per node, each with a single DIMM. Memory capacity is up to 128 GBytes per node (Fig. 2). Future Cascade blade designs might include other x86 processors and different types of accelerators.

The node issues network requests by writing them across the host interface to the NIC. The NIC packetizes these requests with virtual global addresses and issues the packets to the network. Packets are routed across the network to a destination NIC. The destination NIC executes the operation specified in the request packet and returns a response to the source.

Support for I/O is provided by an I/O blade comprising two single-socket nodes and a single Aries ASIC. Each node supports two PCI-Express Gen 3 x8 low-profile cards. An I/O blade is the same form factor as a compute blade. A chassis can be populated with a mix of compute and I/O blades.

Cabinet cooling is implemented using side-to-side air flow, with air moving in series through all cabinets in a row. Heat is extracted via water coils embedded within each cabinet. Air flow is accomplished by six horizontally mounted fans in separate enclosures placed after every pair of cabinets.

B. *Network Topology*

The Cascade network design builds on the Cray BlackWidow system [8], one of the first systems to take advantage of high radix routers [5]. The Cray Dragonfly network design provides scalable global bandwidth while minimizing the number of expensive optical links. Ideally, a router would have a sufficient number of ports to connect to all other routers in the system, with every node one hop away from every other node - a network diameter of one. This is not practicable for large systems, but if a group of routers, acting in concert as a single very-high radix router, pool their links, then the group as a whole has enough links to be directly connected to all other groups in the system. This is the key idea behind the Dragonfly network.

The Aries router is an evolution of the tiled design used in the Cray BlackWidow system [6] and the Cray XE6 system [3]. Using a tile-based micro-architecture simplifies implementation, since each tile is identical, and produces a regular structure for physical implementation. The router tiles are arranged in a 6×8 matrix. Forty of the tiles, referred to as *network tiles*, connect to the SerDes which connect devices together. Eight of the tiles, called *processor tiles* connect to the four Aries NICs. Four of the processor tiles are shared between NIC0 and NIC1, the other four are shared between NIC2 and NIC3. Traffic is multiplexed over the processor tiles on a packet-by-packet basis. This mechanism distributes the traffic according to load, allowing NICs to exceed their fair share of bandwidth on non-uniform communications patterns such as gather and scatter.

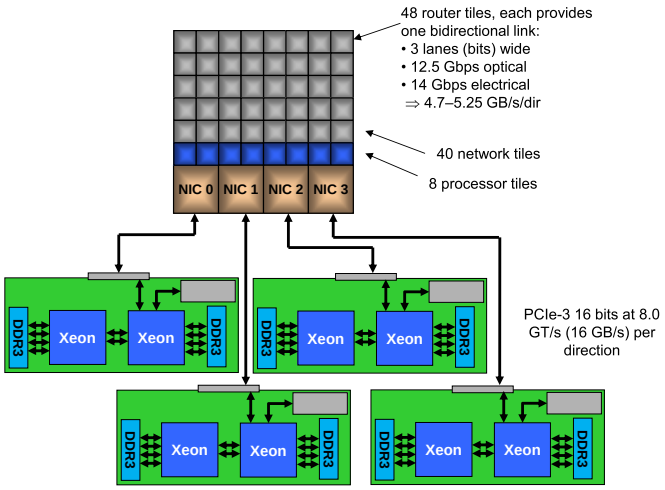

Figure 2. A single Aries system-on-a-chip device provides network connectivity for the four nodes on a Cascade blade.

C. *Network Packaging*

Each router provides both *intra-group* links that connect it to other routers in its group and *inter-group* links (also known as global links) that connect it to the other groups. The group size controls the scalability of the system and was influenced by many factors. To reduce cost, only electrical links are used within a group. The maximum group size is governed by the maximum reach of the electrical cables, which is in turn limited by the signaling rate. Restrictions on the dimension of the mechanical infrastructure limit the number of Aries routers per chassis. The number of ports on the Aries router and the balance between inter-group and global bandwidth also influenced the choice of group size.

High speed, low cost electrical cables are used to construct groups containing up to 384 nodes housed within a pair of Cascade cabinets. These inter-group connections operate at 14 Gbps per lane and require a maximum cable length of 2.3 metres. Optical cables are used for the long links between groups. The Cascade design has no external switches and half the number of optical links as an equivalent fat-tree.

There are three chassis (Fig. 3) within a Cascade cabinet, each containing 16 blades, for a total of 192 nodes per cabinet. A two-dimensional all-to-all structure is used within each two cabinet group (Fig. 4). The chassis backplane provides connectivity in the first dimension, referred to as the Green

dimension. The backplane connects each Aries to each of the other 15 Aries within that chassis.

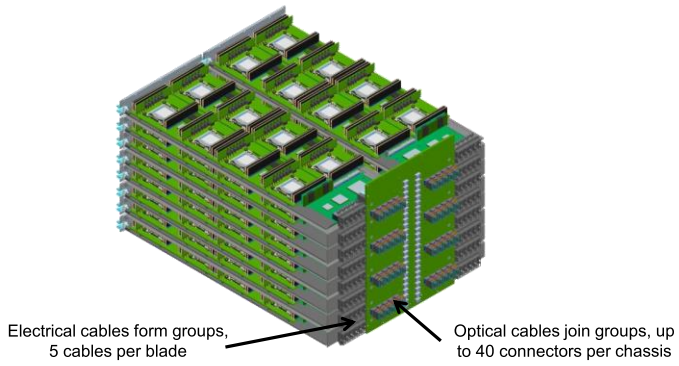

Figure 3. A Cascade chassis comprises 16 four-node blades with an Aries per blade. The chassis backplane provides all-to-all connectivity between the blades. Each blade provides 5 electrical connectors used to connect to other chassis within the group. Each Aries also provides 10 global links. These links are taken to connectors on the chassis backplane.

The second dimension, referred to as the Black dimension, consists of three electrical links from each Aries within a chassis to its peer in each of the other chassis within that group.

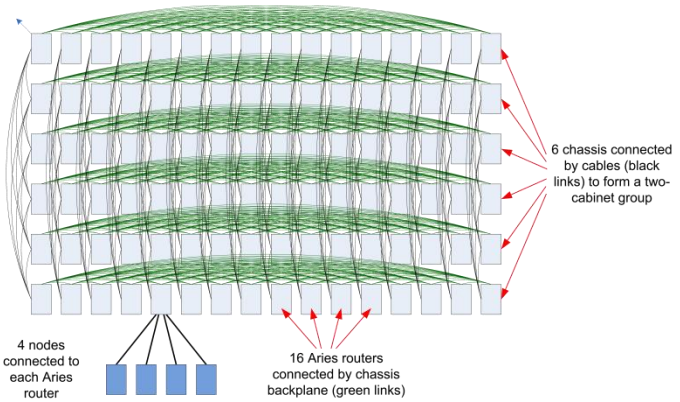

Figure 4. Structure of a Cascade electrical group. Each row represents the 16 Aries in a chassis, with 4 nodes attached to each, and connected by the chassis backplane (green links). Each column represents an Aries in one of the six chassis of a two-cabinet group, connected by electrical cables (black links).

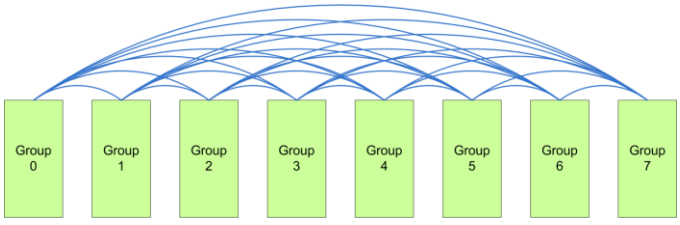

Figure 5. The global (blue) links connect Dragonfly groups together. In a large system these links are active optical cables.

Groups are connected via optical links, referred to as Blue links (Fig. 5). Each Aries provides 10 of these links for a total of 960 per group. Blue links are combined into sets of 4 links, which restricts the maximum size of the system to 241 groups (92,544 nodes). The blue links use 12X Active Optical Cables (AOCs) operating at 12.5 Gbps per lane. For small systems electrical cables are used for the blue links.

The 4 NICs on each Aries connect to eight router ports for packet injection into the network. When intra-group traffic is uniformly distributed over one dimension, a portion will stay local to each router: one part in 16 for the Green dimension and one part in 6 for the Black dimension. The design of the Cascade group (Fig. 4) ensures that the ratio of intra-group to inter-group bandwidths meets or exceeds the factor of two that is desirable for Dragonfly networks. With 10 optical ports per Aries the global bandwidth of a full network exceeds the injection bandwidth – all of the traffic injected by a node can be routed to other groups.

D. System Configurations

Cascade systems are constructed from a number of 2-cabinet groups, each containing 384 nodes. The global links from each group are divided into bundles of equal size, with one bundle from each group connecting to each of the other groups. The number of inter group cables per bundle is given by:

$$inter\ group\ cables \leq \frac{global\ cables\ per\ group}{groups - 1}$$

This number may be reduced substantially in systems that do not require full global bandwidth. One advantage of reducing the number of inter-group cables is that it makes the systems easier to upgrade. For example, a six group (12-cabinet) system might be initially configured with bundles of 12 optical cables, using 60 of the maximum of 240 ports available. If the system is upgraded to eight groups (16-cabinets) then new cable bundles are added, connecting the new groups together and connecting each of the existing groups to the two new ones. These cables can be added to those in place without the need to rewire the system. The total number of optical cables required is given by:

$$inter\ group\ cables \times (groups - 1) \times groups\ /\ 2$$

Small and medium size systems can be upgraded with single chassis granularity. All groups except the last must be full. Larger systems must be comprised of a whole number of cabinets or a whole number of groups.

III. CASCADE ROUTING

In a Cascade network request packets are routed from source node to destination node. Response packets are independently routed back to the source node; there is no requirement for request and response traffic to follow the same path. Packets are routed deterministically or adaptively along either a minimal or non-minimal path. Minimal routing within a group will always take at most one Green and one Black hop. Minimal routing between groups will route minimally in both the source and target groups, and will take exactly one global (Blue) link. Note that minimal routing implies a direct route between a source and a target, not the minimal number of hops required. Minimal paths may differ in hop count if, for instance, one routing path does not require a Green and/or Black hop in the source and/or destination groups due to placement of the Blue global link used between groups.

Non-minimal routing in Cascade is an implementation of Valiant's routing algorithm [6]. It is used to avoid congestion and to spread non-uniform traffic evenly over the set of available links in the system. Non-minimal routes within a group can be thought of as routing minimally from the source Aries to a randomly selected intermediate Aries known as the *root*, and then minimally to the target Aries. Non-minimal routes within a group can take up to two Green hops and two Black hops. A global non-minimal path routes "up" to an intermediate Aries anywhere in the system, and then "down" to the destination node using a minimal path. The maximum path length for a global non-minimal path is ten hops. Most global non-minimal paths are shorter than this (six or seven hops) in all but the largest systems.

To avoid deadlock and allow for routing flexibility, the Aries router provides four Virtual Channels (VCs) for requests and four for responses. Each time a packet crosses into another group (crosses a Blue link) it requires an additional VC to avoid deadlock from cycles within the network. An additional VC is required for non-minimal routing.

Aries supports packet by packet adaptive routing. The routing pipeline selects up to four possible routes at random, two minimal and two non-minimal. Link loads are computed using a combination of downstream link load, estimated far-end link load and near-end link load following the methods introduced in [1]. Downstream load information is propagated from router-to-router at high frequency so as to ensure that routing decisions are made on the basis of up-to-date information. The downstream, far-end, and near-end link loads are combined to produce a single load value per tile. The load on each of the selected paths is compared and the path with the lightest load is selected. Load metrics can be biased towards minimal or non-minimal routing using information supplied by the application. We are working on tuning adaptive routing at both system and application levels. At the system level, we tune the adaptive routing policy based on the packet's path. The network API allows the programmer to select from amongst several adaptive routing algorithms.

Each network packet specifies a routing control mode. Where the runtime requires ordered packet delivery, the route is selected using a deterministic hash computed on fields from the packet header. In general Cascade uses adaptive routing. Aries' method for tracking transactions is not sensitive to the order in which individual packets are delivered. Furthermore, every request includes address information; consequently, re-order buffers are not required.

IV. Memory Architecture and Programming Models

The Cascade system is designed to run distributed memory applications that use MPI, SHMEM and PGAS programming models. The system is cache coherent, but allows only memory from the local node to be cached. All remote references are performed as Gets/Puts and Atomic Memory Operations (AMOs), they are never cached by the initiator. This reduces overhead for the expected case of explicit communication between nodes. A Put, for example, causes data to flow directly across the network to the target node. If the target line is in the cache, the Cascade system is capable of updating the cache directly via hint bits, avoiding any access to main memory by either the Put or subsequent load.

A. Address Translation

Each network request includes a global virtual address which consists of an 18-bit Processing Element (node) identifier (*PE)*, a 12-bit field associated with a memory segment registered at the remote node, and a 40-bit *offset* into this segment. The 70-bit global address extends the physical address space of the x86 node, enabling global access to all of the memory in the system. The 12-bit field is combined with the offset to generate a user virtual address in the target process. This virtual address is translated to a physical address in the target node.

Aries provides virtual to physical address translation for incoming network requests, fetching network responses, local reads generated by the NIC, and completion event writes. It supports Linux 4 Kbyte pages using 2-level translation, and huge page sizes of up to 64 Gbytes using a single level translation. The translation unit includes a deep pipeline to hide the latency of fetching Page Table Entries (PTEs) on a PTE cache miss. There is no requirement to use large pages on Aries; their use is a performance optimization as it is on general purpose processors. Fetching PTEs will consume significant host bandwidth on random access patterns. This load can be reduced by using large pages. Large pages are used by default for the SHMEM symmetric heap. The Cray programming environment makes it straightforward to use large pages for other segments.

B. Remote Memory Access

Aries provides two remote memory access mechanisms Fast Memory Access (FMA) and the Block Transfer Engine (BTE).

When using FMA user processes generate network transactions consisting of Puts, Gets and AMOs by writing directly to an FMA window within the NIC. The FMA unit translates I/O writes by the processor into fully qualified network requests. FMA supports scattered accesses by allowing the user library to select which bits of the offset into the FMA window determine the remote address and which the remote PE. Having set the FMA descriptor appropriately one can, for example, store 64, 128, or 256 bytes of data to each process in a parallel job by simply storing a contiguous block of data to the FMA window. FMA provides both low latency and high issue rate on small transfers.

FMA supports source-side synchronization methods for tracking when Put requests have reached a *globally ordered* point at the destination node and when responses to Get requests have completed at the local node. It is also possible to issue Puts that generate destination-side synchronization events, enabling a process on the remote node to be notified of new data or to poll a completion queue for its arrival.

The BTE supports asynchronous transfer between local and remote memory. Library software writes block transfer descriptors to a queue and the Aries hardware performs the transfers asynchronously. The BTE supports memory operations (Put/Get) where the user specifies a local address, a network address and a transfer size. In addition, the BTE

supports channel operations (Send) where the user specifies a local address and a remote PE, but no destination address. Channel semantics require the user to have pre-posted a receive buffer with the target BTE. By default, there is no guarantee of completion ordering amongst block transfers issued by a given NIC. Fence operations are used where necessary to ensure that one transfer is completed before another starts.

In general FMA is used for small transfers and BTE for large. FMA transfers are lower latency but require the involvement of the CPU. BTE transfers take longer to start, but once running can transfer large amounts of data (up to 4GB) without CPU involvement. Aries provides support for user-space launch of block transfer requests, reducing the issue latency and hence the minimum efficient block transfer size.

Aries includes a novel fast path, *FMA Launch*, which optimizes the issue rate for single word remote memory operations. Once an FMA window has been initialized, and a transaction has been opened, a remote memory operation can be issued with just a single 128-bit write from host to NIC and an x86 sfence instruction. Each of these remote memory operations can target an arbitrary remote PE and offset. A completion event is generated once the transaction has been closed and all pending remote memory operations have been completed.

Aries synchronization is based on a concept of transactions. All operations that access remote data occur within the bounds of a transaction. When using FMA, software explicitly defines the bounds of each transaction. With BTE, a transaction consists of one or more block transfers. For each transaction, Aries hardware counters track its constituent requests and their corresponding responses. In this way Aries is able to determine when each transaction has completed at both the local and the remote nodes and is able to deliver an event to a completion queue at the local node signaling software that the transaction is globally complete. Aries is also able to notify remote node software of completion at the remote node. The transaction concept is also used to assist error containment and recovery.

C. Reductions and Barriers

The Aries Collective Engine (CE) provides hardware support for reduction and barrier operations. The CE is optimized for latency sensitive, single word operations that limit scalability in many large HPC applications. CE supports logical operations and barriers, together with min, max, and sum operations on integer and 32/64-bit floating point types. The min and max operations return the location as well as the value supporting MPI_MINLOC and MPI_MAXLOC.

V. Network Reliability, Availability, and Serviceability

Aries offers sophisticated Reliability, Availability, and Serviceability (RAS) capabilities with comprehensive hardware error correction coverage, resulting in fewer retransmissions, higher availability, and fewer maintenance events. A combination of both hardware and software techniques is used to avoid a broad class of well-understood faults such as single-bit memory errors and transmission errors on network links. However, when a fault does manifest as an error, the hardware and software layers identify and contain the fault, recover from the error if possible, and prevent data corruption. Aries provides support for identification of the first error within a sequence of errors occurring in a short interval of time. This mechanism simplifies root cause analysis of problems.

Although improved semiconductor fabrication processes have improved yield and reduced the extent of manufacturing defects, or *hard* errors, in the silicon, these high-density, low-voltage devices are increasingly susceptible to influences from electrical noise, process variation, and natural radiation interference. The resultant errors from these effects are referred to as *soft* errors since they can corrupt the state, but generally do not result in any permanent damage to the underlying circuitry. Soft errors are detected and possibly corrected by mechanisms such as parity, ECC, and CRC. Within Aries, major memories, and much of the data path are protected using ECC.

Packet level CRC checks in the network transport layer provide immediate indication of corrupt data from end to end. Errors are reported as precisely as possible: data payload errors are reported directly to the user, but control errors cannot always be associated with a particular transaction. In all cases, either the operating system or the Hardware Supervisory System (HSS) are notified of the error. Router errors are reported at the point of the error with the packet possibly being discarded. In the case of a discarded packet, the initiating NIC times out, reporting a timeout error to the user process.

The Aries network link-layer protocol provides reliable packet delivery across each link. The Link Control Block provides automatic retransmission in the event of an error. A 20-bit CRC code protects 60 bytes of data distributed across 3 SerDes lanes. For a 25,000 node system, with a bit error rate of 10^{-7} on all SerDes lanes, the expected rate of undetected error is less than one every 7250 years. Measured bit error rates on correctly functioning links are better than 10^{-12}.

By incorporating error detection and handling into the communication protocol, the status of every transaction can be passed to the originating process. User level libraries, MPI in particular, use this information to ensure that applications are resilient to transient network failures.

The Synchronization Sequence Identification (SSID) mechanism provides functionality to assist in the identification of software operations and processes impacted by errors. This aids error recovery, and improves system resiliency, by enabling the overall system to survive error events even though remedial action may need to be applied to those processes actually impacted by the errors. The SSID provides this functionality by monitoring errors detected at the source NIC as well as errors returned from the destination NIC in response packets. These errors are included in the associated completion event.

Adaptive routing helps mitigate the effects of link failure. The adaptive routing hardware spreads packets over the correctly functioning links according to load. If a link fails, the adaptive routing hardware automatically stops sending packets in that direction. In the event of lost connectivity, such as blade failures or hardware removal, it is necessary to route around the problem. The Hardware Supervisory System (HSS) performs this task.

Cascade systems are monitored via the HSS. Daemons running on the blade level controllers are responsible for monitoring the MMRs which indicate the health of the network. Fatal errors can be configured to generate interrupts to HSS. HSS reacts to failed links or routers by computing updates to the routing tables in order to avoid the problem. Where necessary, HSS can request that network traffic be quiesced, so that route table changes can be deployed safely. Applications pause briefly while this operation is in progress with traffic released once the operation completes.

PCI-Express traffic between Aries and the CPU is protected using a 32-bit link layer CRC per packet with automatic retry in case of error. PCI-Express includes support for Advanced Error Reporting (AER) so that link level errors can be reported to the host processor (and/or HSS) to assist in fault isolation and recovery. Correctable AER errors are recovered by the PCI-Express protocol without the need for software intervention, and without any risk of data loss. Uncorrectable errors are passed to the operating system.

VI. Software Stack

Cascade supports both kernel communication, through a Linux device driver, and direct user space communication, where the driver is used to establish communication domains and handle errors, but is bypassed for data transfer. Parallel applications typically use a library such as Cray MPI or SHMEM in which the programmer makes explicit communication calls. Alternatively they may use a programming model such as CAF, UPC, or Chapel in which the compiler automatically generates the inter-process communication calls.

Standard communications libraries are layered over the user level Generic Network Interface (uGNI) and/or the Distributed Memory APPlications (DMAPP) library which performs the Cray network specific operations (Fig. 6). Kernel modules such as the Lustre Network Driver (LND) communicate via the kernel Generic Network Interface (kGNI).

Security is maintained by the use of communication domains. Each such communication domain defines a set of processes or kernel threads that are allowed to communicate. Creating a communications domain is a privileged operation. Any attempt to communicate with a process outside of the communication domain is blocked by hardware and generates an error.

Program initialization follows a sequence of steps in which the placement scheduler (ALPS) first creates a communication domain and then starts the user processes. The processes sign on to the communication domain and create their completion queues and register memory with the communication domain. Having completed this sequence of steps, the processes in a parallel job can initiate either Put/Get or Send/Receive style communication without operating system intervention. These operations are asynchronous, with completion events being generated when an operation or sequence of operations has been completed. This approach promotes programming models in which communication is initiated early in order to hide latency. Aries optimizes for this, supporting both large numbers of pending operations and high issue rates.

Cray MPI uses the MPICH2 distribution from Argonne. The MPI library uses a Nemesis driver for Aries layered over uGNI [7]. Use of FMA gives MPI applications the ability to pipeline large numbers of small, low latency transfers; an increasingly important requirement for strong scaling on multi-core nodes. Where space is available, intermediate size messages are sent eagerly to pre-allocated system buffers. Large messages are transferred using a rendezvous protocol in which bulk data transfer occurs after matching of a message header and a receive request. MPI message matching is progressed by each call, or in the case of large messages, using an optional helper thread. The Aries block transfer engine is used to provide high bandwidth and overlap of computation and communication. Implementation of latency sensitive collectives, MPI_Allreduce and MPI_Barrier, is optimized using the Aries collective engine.

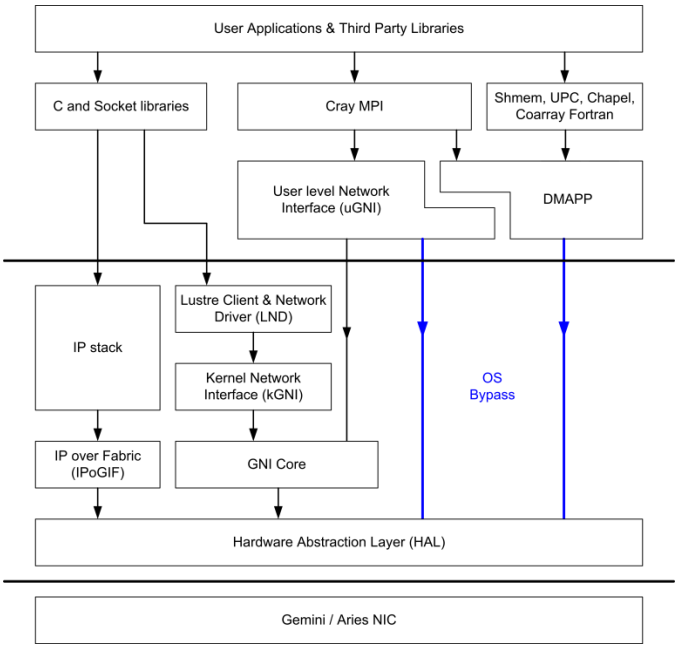

Figure 6. Cray network software stack

Cray SHMEM [9] provides an explicit one-sided communication model. Each process executes in its own address space but can access segments of the memory of other processes, typically the static data segment and the symmetric heap, through a variety of put and get calls, AMO calls, and collectives. Cray systems since the T3D have supported SHMEM. Its implementation for Aries provides the application programmer with fine grain control of communication with a minimum of overhead.

Aries is designed to provide efficient support of emerging PGAS programming models. Cray's CoArray Fortran, UPC, and Chapel compilers use a common runtime implemented with DMAPP. The DMAPP library includes blocking, non-blocking and indexed variants of put and get together with scatter/gather operations and AMOs. This interface is also provided for third parties developing their own compilers, libraries and programming tools. Compiler generated communication typically results in large numbers of small irregular transfers. The Aries FMA mechanism minimizes the issue overhead and its high packet rate maximizes performance.

The Aries versions of GNI and DMAPP preserve the application interfaces developed for Cray XE6 systems, ensuring compatibility of higher levels of software. Support for new Aries hardware features is encapsulated within the Aries versions of GNI and DMAPP.

VII. Related Work

Cray XT and XE [2,3] are examples of parallel systems with network connections between neighboring nodes, organized as a 3-D torus. The Blue Gene Q [10] and Fujitsu K computer [11] use higher dimension tori. The IBM PERCS system [12] uses a hierarchy of all-to-all networks. It is interesting to note that Cascade and PERCS were both developed under the DARPA program to meet similar user requirements.

The torus routers used in the Cray XE6 system have five times as much routing bandwidth as injection bandwidth. Even with this much bandwidth, performance will degrade when each packet takes an average of more than five hops. The bandwidth that each node can sustain on complex traffic patterns, such as all-to-all, falls as the system size increases. However, applications which are dominated by nearest neighbor, or local, communication patterns can be executed efficiently on a parallel system such as Cray XE. One of the objectives of the Cascade program is to make Cray systems more productive by increasing their programmability, performance and efficiency. By increasing the global bandwidth we enable more efficient execution of a wider range of applications.

Another class of network known as fat trees [4] is widely used in constructing small and medium size HPC systems. These networks can have bi-section bandwidth equal to injection bandwidth, although they are frequently tapered reducing cost and bandwidth. The advantage that a Dragonfly network has over a fat-tree depends on the percentage of packets which are minimally routed. With 100% minimal routing, in a Dragonfly network each packet takes at most one optical hop rather than the two of a fat-tree network, giving Dragonfly a 2:1 advantage. If all traffic is non-minimally routed then the load on the optical links is equal to that in a full bandwidth fat-tree. Traffic patterns that generate high volumes of global traffic are well suited to minimal routing provided there is good diversity in the range of destinations each node is communicating with at any given point in time. Regular traffic patterns with low path diversity generate a higher proportion of non-minimal routing, but most packets remain local to each group.

VIII. Performance

This section describes our preliminary results on prototype hardware and using detailed simulation models of the Cascade system. Peak raw bandwidth of the PCI-Express Gen3 x16 interface is 16 GBytes/sec. However, due to PCI-Express overhead the payload rate is approximately 10 GBytes/sec for a 64-byte message. This is the peak payload injection rate that can be achieved by user processes. With symmetric traffic there is additional overhead on both the host interface and the links, as requests and responses share the same data path. This reduces bandwidth on symmetric data transfer to approximately 7.5 GBytes/sec in each direction.

The bisection bandwidth for a fully configured multi-group system is given by:

$$\frac{G}{2(G-1)} \times L \times global\ links\ per\ group \times G$$

where G is the number of groups and L is the link bandwidth. The ratio G/2(G-1) reflects the proportion of links connected to the other half of the system, falling to ½ as the system size increases. The maximum number of optical cables per group is 240. For reduced bandwidth systems or systems in which the number of optical ports per group does not divide evenly by the number of other groups, the number of optical cables per group, and hence the global bandwidth, is reduced.

Raw global bandwidth is 11.7 GBytes/sec per node for a full network. Taking into consideration packet and link overheads, symmetric payload efficiency is 64%, which equates to a maximum symmetric global payload bandwidth of 7.5 GBytes/sec. For systems with a large number of groups, the per node bisection bandwidth is approximately one half the global bandwidth.

Measured end-to-end latencies on the prototype hardware for user-space communication are 0.8 µs for an 8-byte Put, 1.6 µs for an 8-byte Get and approximately 1.3 µs for an 8-byte MPI message (Fig. 7). With more software overheads, the MPI latencies depend on the type and speed of CPU. In each case there is little additional cost for transferring 64 or 128 bytes of user data rather than just 8 bytes, the per transfer setup costs dominate. For a quiet network, each router-to-router hop adds approximately 100 ns of latency, giving a maximum of 500 ns for minimally routed packets – packets will be minimally routed when the network is quiet.

Bandwidths measured between user processes on different Intel Sandy Bridge nodes reach approximately 10 GB/sec for unidirectional traffic. Bidirectional bandwidths are approximately 15 Gbytes/sec, 7.5 GBytes/sec in each direction (Fig. 8). With multiple issuing processes per node half peak bandwidth is obtained for Puts of size 128 bytes. Each NIC supports up to 1024 outstanding packets, ensuring that bandwidths can be maintained between widely separated nodes.

A single node can issue Put requests at rates of up to 120 million per second using the FMA Launch mechanism. The issue rate for a single thread/process depends upon the CPU core - rates of up to 7.5 million requests per second per core have been measured to date.

The Aries AMO unit is capable of one operation every two cycles. Rates of up to 400 million AMOs per second can be obtained when many processes update a single cached variable. AMO rates to random variables that miss in the AMO cache depend on the performance of the host memory system and the load generated by user processes. Rates in the range of 50 to 120 million per second are possible depending on the size of the array being accessed. Large page sizes are required in order to maintain Put or AMO rates over large target arrays.

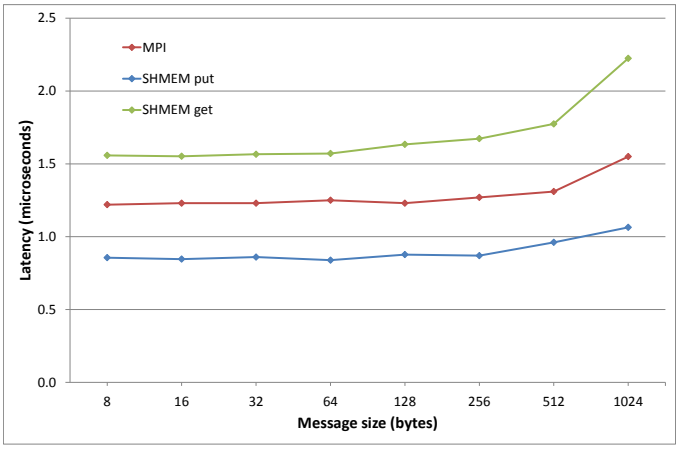

Figure 7. Aries latencies as a function of transfer size. Results are for MPI Send/Recv, SHMEM Put and SHMEM Get, on a prototype Cascade system.

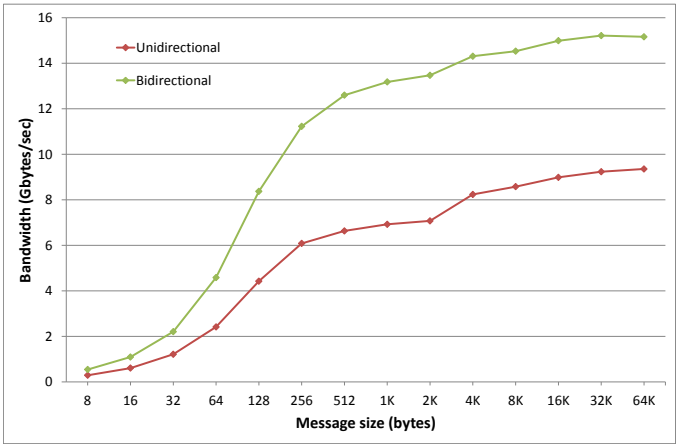

Figure 8. Aries point to point Put bandwidth as a function of transfer size.

In Fig. 9 we show bandwidth results from a cycle-accurate simulation of an all-to-all benchmark with uniform random destinations as a function of message size using a range of routing modes. The all-to-all algorithm uses a rotating shift communication pattern to avoid end-point contention. For small messages we see good performance with minimal routing because the traffic is uniform with a high degree of path diversity. For large messages path diversity is low and performance of the minimal routing modes begins to degrade while the performance of the other routing modes remains relatively constant.

There is not a single routing mode that performs well for all message sizes or all traffic patterns. Aries provides mechanism to tune the adaptive routing algorithms, details of which we intend to publish when systematic tests on large systems have been completed. In Fig. 10 we show early results that indicate that these tuning mechanisms almost eliminate the performance drop-off for large all-to-all messages.

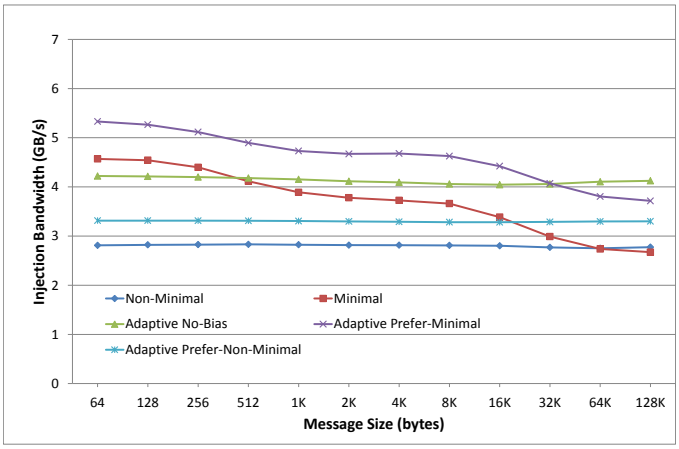

Figure 9. Random All-to-All PUTs on an 8 Group, 6 chassis per group, system (3072 nodes), bandwidth is shown for different routing algorithms.

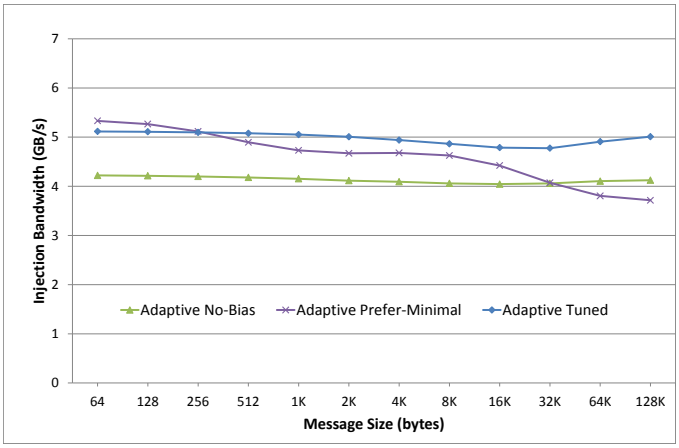

Figure 10. Random All-to-All PUTs, Tuned Adaptive versus other adaptive routing modes, on an 8 Group, 6 chassis per group, system (3072 nodes)

Another attribute of the Cascade system is the ability to execute a parallel application efficiently with little regard to job placement. For example, if an application requires a small portion of a large system it can be placed anywhere on the system without significant degradation in performance. Simulation of a 384 node all-to-all benchmark running on a 3072 node system (Fig. 11) shows little variation in performance for different placements on a system where the nodes not assigned to the job are idle. Fig. 12 shows the results for the same placements on a system where all the nodes are busy.

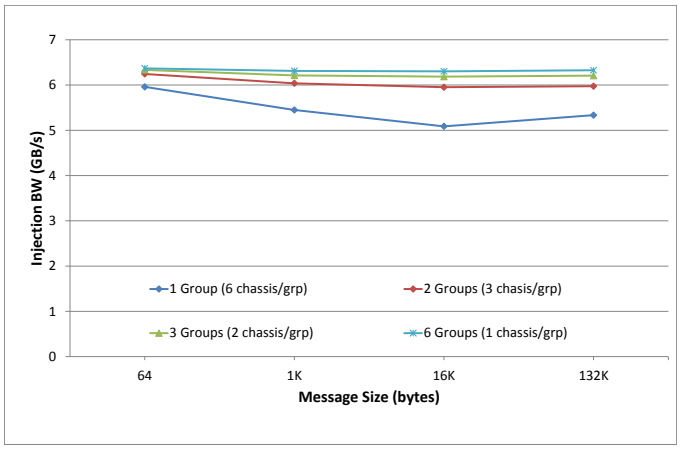

Figure 11. Small All-to-All benchmark (384 nodes), of message size 64 bytes to 132K bytes, placed across 1, 2, 3 and 6 groups on a 3072 node *idle* system.

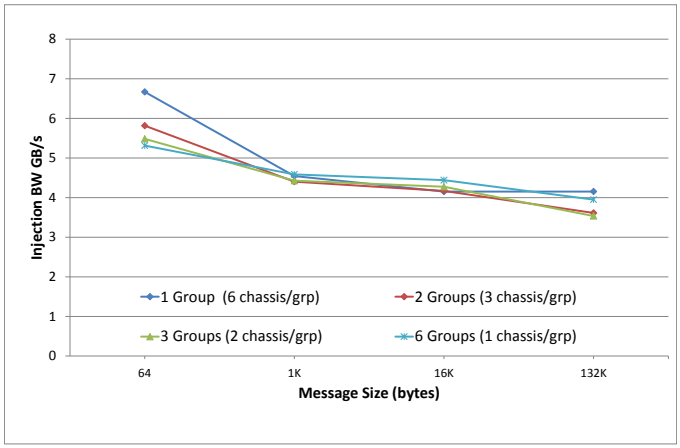

Figure 12. Small All-to-All benchmark (384 nodes), of message size 64 bytes to 132K bytes, placed across 1, 2, 3 and 6 groups on a 3072 node *busy* system.

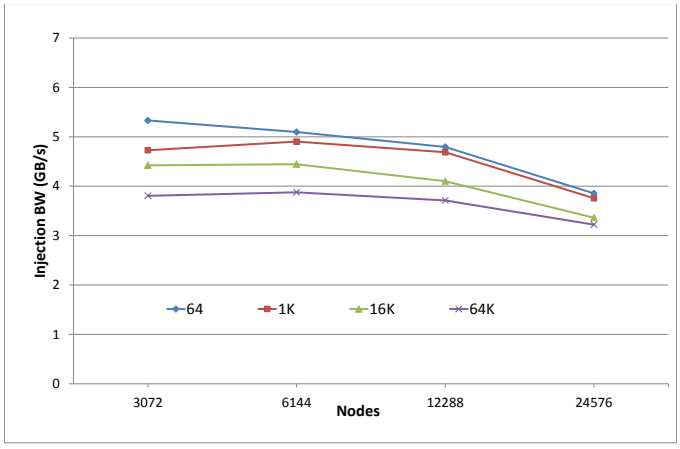

Figure 13. System-wide All-to-All benchmark with message sizes of 64 bytes to 64K bytes, running on various size systems.

One of the key design goals of the Cascade system is the ability to provide scalable global bandwidth for complex communication patterns such as All-to-All. In Fig. 13 we show that the Cascade system is able to sustain good per-node injection bandwidth on systems as large as 24,576 nodes (49,152 sockets). This makes Cascade ideally suited for strong scaling problems that scale poorly on current systems.

IX. Conclusions

The Cascade network provides the essential components necessary to build the communication system for Cray's next generation of supercomputers. In addition to standard programming environments such as MPI and SHMEM it supports emerging global address space programming models that require fine-grain single-sided Put, Get, and AMO operations, and light-weight synchronization methods. The Dragonfly network used by Cascade provides cost-efficient scalable global bandwidth. This together with advanced RAS features allows Cascade systems to scale in excess of one million cores.

X. Acknowledgements

Only a small number of the many Cray employees that developed Cascade were involved in the preparation of this paper. The authors would like to acknowledge the enormous effort made by the entire team.

XI. References

[1] J. Kim, W.J. Dally, S.Scott and D.Abts, "Technology-Driven, Highly-Scalable Dragonfly Topology", Proc. of the International Symposium on Computer Architecture (ISCA), pages 77-88.

[2] R. Brightwell, K. Predretti, K. Underwood, and T. Hudson. "Seastar Interconnect: Balanced Bandwidth for ScalablePerformance", Micro, IEEE, 26(3):41–57, 2006

[3] Bob Alverson, Duncan Roweth and Larry Kaplan, Cray Inc. "The Gemini System Interconnect", High-Performance Interconnects Symposium, 83–87 (2010).

[4] C. Leisserson, "Fat-trees: Universal networks for hardware efficient supercomputing", IEEE Transactions on computer, C-34(10):892-901, October 1985.

[5] S. Scott, D. Abts, J. Kim, and W. J. Dally. "The BlackWidow High-radix Clos Network", Proc. of the International Symposium on Computer Architecture (ISCA), pages 16–28, 2006.

[6] L. G. Valiant, "A scheme for fast parallel communication", SIAM Journal on Computing, 11(2):350–361, 1982.

[7] H. Pritchard, I. Gorodetsky, and D. Buntinas, "A uGNI based MPICH2 Nemesis Network Module for the Cray XE", Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface, EuroMPI'11, pages 110–119, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] Dennis Abts, Abdulla Bataineh, Steve Scott, Greg Faanes, Jim Schwarzmeier, Eric Lundberg, Tim Johnson, Mike Bye, Gerald Schwoerer, "The Cray BlackWidow: A Highly Scalable Vector Multiprocessor", SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1-12.

[9] Cray Research, Inc.: SHMEM Technical Note for C, SG-2516 2.3. (1994).

[10] Dong Chen, Eisley, N.A.; Heidelberger, P.; Senger, R.M.; Sugawara, Y.; Kumar, S.; Salapura, V.; Satterfield, D.; Steinmacher-Burow, B.; Parker, J., "The IBM Blue Gene/Q Interconnection Fabric", IEEE Micro, Jan.-Feb. 2012, Volume: 32, Issue: 1, pg 32.

[11] Takumi Maruyama "SPARC64(TM) VIIIfx: Fujitsu's New Generation Octo Core Processor for PETA Scale computing",. Proceedings of Hot Chips 21. IEEE Computer Society, 2009.

[12] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni and R. Rajamony, "The PERCS High-Performance Interconnect", Proceedings of 18th Symposium on High-Performance Interconnects (Hot Interconnects 2010), IEEE, Aug. 2010).