2010 18th IEEE Symposium on High Performance Interconnects

The PERCS High-Performance Interconnect

Baba Arimilli *, Ravi Arimilli *, Vicente Chung *, Scott Clark *, Wolfgang Denzel †, Ben Drerup *, Torsten Hoefler ‡, Jody Joyner *, Jerry Lewis *, Jian Li †, Nan Ni * and Ram Rajamony †
* *IBM Systems and Technology Group, 11501 Burnet Road, Austin, TX 78758*
† *IBM Research (Austin, Zurich), 11501 Burnet Road, Austin, TX 78758*
‡ *Blue Waters Directorate, NCSA, University of Illinois at Urbana-Champaign, Urbana, IL 61801*
E-mail: arimilli@us.ibm.com, rajamony@us.ibm.com, htor@illinois.edu

***Abstract*—The PERCS system was designed by IBM in response to a DARPA challenge that called for a high-productivity high-performance computing system. A major innovation in the PERCS design is the network that is built using Hub chips that are integrated into the compute nodes. Each Hub chip is about 580 mm^2 in size, has over 3700 signal I/Os, and is packaged in a module that also contains LGA-attached optical electronic devices.**

The Hub module implements five types of high-bandwidth interconnects with multiple links that are fully-connected with a high-performance internal crossbar switch. These links provide over 9 Tbits/second of raw bandwidth and are used to construct a two-level direct-connect topology spanning up to tens of thousands of POWER7 chips with high bisection bandwidth and low latency. The Blue Waters System, which is being constructed at NCSA, is an exemplar large-scale PERCS installation. Blue Waters is expected to deliver sustained Petascale performance over a wide range of applications.

The Hub chip supports several high-performance computing protocols (e.g., MPI, RDMA, IP) and also provides a non-coherent system-wide global address space. Collective communication operations such as barriers, reductions, and multi-cast are supported directly in hardware. Multiple routing modes including deterministic as well as hardware-directed random routing are also supported. Finally, the Hub module is capable of operating in the presence of many types of hardware faults and gracefully degrades performance in the presence of lane failures.

***Keywords*-interconnect, topology, high-performance computing**

I. Introduction

In 2001, DARPA called for the creation of high-performance highly productive, commercially viable computing systems. The forthcoming system from IBM called PERCS (Productive Easy-to-use Reliable Computing System) is in direct response to this challenge. Compared to state-of-the-art high-performance computing (HPC) systems in existence today, PERCS has very high performance and productivity goals and achieves them through tight integration of computing, networking, storage, and software.

Although silicon technologies (e.g., multi-core dies, 45nm) continue to improve generation after generation [6], surrounding technologies in HPC systems such as the interconnect bandwidth, memory densities and bandwidths, power packaging and cooling, and storage densities and bandwidths do not scale accordingly. For instance, while High Performance Linpack performance [5], [10] shows a steady improvement over time, interconnect-intensive metrics such as G-RandomAccess and G-FFTE [5] show very little improvement.

The challenge of building a high-performance, highly productive, multi-Petaflop system forced us to recognize early on that the entire infrastructure had to scale along with the microprocessor's capabilities. A significant component of our scaling solution is a new switchless interconnect with very high fanout organized into a two-level direct connect topology. Using this interconnect technology enables us to build a full system with no external switches and half the physical interfaces and cables of an equivalent fat-tree structure with the same bisection bandwidth.

The rest of this paper is organized as follows. We describe the PERCS compute node in Section II. The IBM Hub chip is the gateway to the interconnect as well as the routing switch in the system. We describe the Hub chip in Section III and the interconnect topology in Section V. The Hub chip has several components that permit it to offer high value as well as high performance. We describe these components in Section IV. The two-tiered full-connect graph typology allows for several routing innovations which we describe in Section VI. We conclude with a description of the Blue Waters Sustained Petascale System in Section VII.

II. System Overview

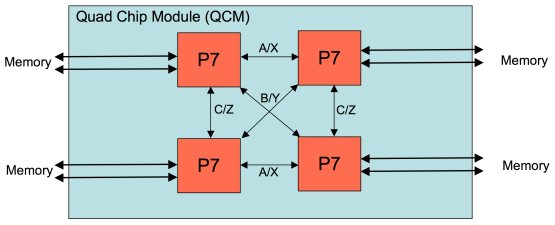

Figure 1. Compute node structure

Figure 1 shows the abstract structure of a compute node in a PERCS system. There are four POWER7 chips in a node with a single operating system image that controls resource

978-0-7695-4208-9/10 $26.00 © 2010 IEEE
DOI 10.1109/HOTI.2010.16

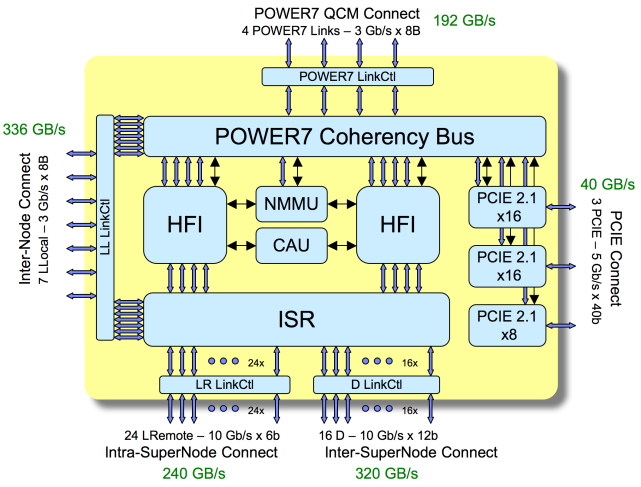

Figure 2. IBM Hub chip overview

allocation. Applications executing on the compute node can utilize 32 cores, 128 SMT threads, eight memory controllers, up to 512 GB of memory capacity, over 900 GFLOPS of compute capacity and over 500 GB/s of memory bandwidth. The four POWER7 chips are cache coherent and are tightly coupled using three pairs of buses.

The IBM Hub chip completes the compute node, providing network connectivity to the four POWER7 chips. The Hub chip participates in the cache coherence protocol within the node and serves not only as an interconnect gateway to the four POWER7 chips that connect to it, but also as a switch that routes traffic between other IBM hub chips. A PERCS system therefore requires no external network switches or routers with considerable savings in the switching components, cabling, and power.

III. Hub Chip

The main purpose of the IBM Hub Chip is to interconnect tens of thousands of compute nodes and to provide I/O services. The Hub design provides ultra-low latencies at high bandwidth, dramatically improving the scalability of applications written using such varied programming APIs as MPI, sockets, and PGAS languages [2].

The Hub chip also improves the performance and cost of an HPC storage subsystem by requiring no FCS Host Bus Adapters, no external switches, no storage controllers and no direct attached storage within the compute nodes. The Hub chip also obviates the need for external PCI-Express controllers by integrating them on-chip.

Key functions used by software are accelerated in hardware by the Hub Chip. The Collective Acceleration Unit (CAU) in the Hub chip speeds up collective (including synchronization) operations that are often a big scalability impediment to high-performance computing applications. The Hub Chip also employs a memory management unit that is kept consistent with the TLBs on the compute cores. This enables an application running on one compute node to use program-level effective addresses to operate upon data located on another compute node. Finally, the Hub Chip also has special facilities to enable certain operations to be atomically performed in the compute node's memory without involving any of the compute node's cores.

The Hub chip is implemented using 45 nm lithography Cu SOI technologies. The chip is 582 mm^2 in size with 440M transistors and 13 levels metal. There are over 3700 signal I/O and over 11,000 total I/O pins. The Hub chip is integrated along with 12X optics modules into a 58 cm^2 glass ceramic LGA module.

Figure 2 shows an overview of the Hub chip.

IV. Hub Chip Details

The different components of the Hub chip are described in greater detail below.

A. PowerBus Interface

The PowerBus interface enables the Hub chip to participate in the coherency operations taking place between the four POWER7 chips in the compute node. The Hub chip is a first-class citizen in the coherence protocol and has visibility to coherence transactions taking place in the node, including TLB-related coherence operations.

B. Host Fabric Interface

The two HFI units in the Hub chip manage communication to and from the PERCS interconnect. The HFI was designed to provide user-level access to applications. The basic construct provided by the HFI to applications for delineating different communication contexts is the "window". The HFI supports many hundreds of such windows each with its associated hardware state.

An application invokes the operating system and thus the hypervisor to reserve a window for its use. The reservation procedure maps certain structures of the HFI into the application's address space with window control being possible from that point onwards through user-level reads and write to the HFI mapped structures.

The HFI supports three APIs for communication:

- General packet transfer: This can be used for composing unreliable protocols as well as reliable protocols such as needed for MPI through higher levels of the software stack.
- Global address space operations and active messaging: This can be used by user-level codes to directly manipulate memory locations of a task executing on a different compute node. The Nest Memory Management Unit provides support for these operations.
- Direct Internet Protocol (IP) transfers

The HFI can extract data that needs to be communicated over the interconnect from either the POWER7 memory or

directly from the POWER7 caches. The choice of source is transparent with the data being automatically sourced from the faster location (caches can typically source data faster than memory). In addition to writing network data to memory, the HFI can also inject network data directly into a processor's L3 cache, lowering the data access latency for code executing on that processor.

Five primary packet formats are supported: Immediate sends, FIFO send/receive, IP, remote DMA (RDMA), and Atomic updates.

A new PowerPC instruction, ICSWX, is used to implement immediate sends [7]. This instruction forces a cache line directly to the HFI for interconnect transmission and is the lowest latency (at the expense of bandwidth) communication mechanism for sending packets that are less than a cache line in size.

The FIFO send/receive mode permits an application to use a staging area for both sending and receiving data. An application can pre-reserve a portion of its address space to serve as circular First-In-First-Out buffers. After composing packets in the send FIFO, the application "triggers" the HFI by writing an 8-byte value to a per-window trigger location. In this mode, incoming packets are written to the receive FIFO by the HFI and can then be processed by the application. An 8-byte write to another location informs the HFI of the space that it can reuse in the receive FIFO.

The HFI supports two forms of IP transfers. IP packets can be transferred to and from the FIFO (see above). IP packets can also be described with scatter/gather descriptors with the HFI assembling/dissembling data.

A variety of RDMA mechanisms are supported. In addition to traditional memory-to-memory transfers, the HFI also supports transfers between the FIFO and memory. Since these are asynchronous operations, completion notifications permit an application to implement read and write fences.

A final packet format permits an application to specify atomic updates to remote memory locations. Fixed-point operations such as ADD, AND, OR, XOR, and Cmp & Swap with and without Fetch for multiple data sizes (8-, 16-, 32-, 64-bits) are supported. Sequence numbers are used to ensure proper reliable operation of all atomic updates, with an optimized mode permitting up to four operations to be packed per cache line at a coarser reliability granularity.

Collective packets are also supported and the operation is described in more details in Section IV-D.

C. Integrated Switch Router (ISR)

The ISR implements the two-tiered full-graph network described in Section V. It is organized as a 56×56 full crossbar that operates at up to 3 GHz. In addition to the forty-seven L and D ports described previously, the ISR also has eight ports to the two local Host Fabric Interfaces, and one service port.

The ISR uses both input and output buffering with a packet replay mechanism to tolerate transient link errors. This feature is especially important since the D links can be several tens of meters in length. The ISR operates in units of 128-byte FLITs with a maximum packet size of 2048 bytes. Messages are composed of multiple packets with the packets making up a message being potentially delivered out of order.

High-performance computing applications benefit from having access to a single global clock across the entire system. The ISR implements a global clock feature whereby a clock onboard is globally distributed across the interconnect and kept consistent with the clocks on other Hub chips.

Deadlock prevention is achieved through virtual channels, each corresponding to a hop in the L-D-L-D-L worst case route.

More details of the ISR as it pertains to routing are described in Section V below.

D. Collectives Acceleration Unit (CAU)

Many HPC applications perform collective operations with the application being able to make forward progress not only after every compute node has completed its contribution to the collective operation, but also after the results of the collective are disseminated back to every compute node (e.g. barrier synchronization or a global sum). The Hub Chip provides specialized hardware to accelerate frequently used collective operations.

Specialized ALU logic within the CAU implements multicast, barriers and reduction operations. For reductions, the ALU supports the following operations and data types:

- Fixed point: NOP, SUM, MIN, MAX, OR, AND, XOR (signed and unsigned)
- Floating point: MIN, MAX, SUM, PROD (single and double precision)

Software organizes the CAUs in the system into collective trees. Each tree is set up so that it "fires" when data on all of its inputs are available with the result being fed to the next "upstream" CAU. There is one CAU in each Hub chip and a link in the CAU tree could map to a path in the network made up of more than one link. A multiple-entry content addressable memory structure per CAU supports multiple independent trees that can be concurrently used by different applications, for different collective patterns within the same application, or some combination.

Reliability and pipelining are afforded using sequence numbers and a retransmission protocol. Each tree has exactly one participating HFI window on any involved node. The tree can be set up such that the order in which the reduction operations are evaluated is preserved from one run to another. Programming models such as the Message Passing Interface (MPI) [8], which permit programmers to require collectives to be executed in a particular order, can benefit from this feature.

E. Nest Memory Management Unit (NMMU)

A key facility for high-performance global address space languages such as UPC [3], CAF [9], and X10 [2] is a low-overhead mechanism for user-level code to operate upon the address space of processes executing on the compute nodes. The NMMU in the Hub chip facilitates such operations.

A process executing on a compute node can register its address space, permitting interconnect packets to directly manipulate the registered region. Registering a portion of the address space results in the NMMU being able to reference a page table mapping table that maps effective addresses to real memory. A cache of the mappings is also maintained within the Hub chip and can map the entire real memory of most installations.

Incoming interconnect packets that reference memory such as RDMA packets and packets that perform atomic operations contain both an effective address as well as information pinpointing the context in which to translate the effective address. This greatly facilitates global address space languages by permitting such packets to contain easy-to-use effective addresses.

F. IO connectivity

The Hub chip has three PCI-E ports. Two of the ports are $\times 16$ and support $\times 16$, $\times 8$, $\times 4$, and $\times 1$ connections. The third port is $\times 8$ and supports $\times 8$, $\times 4$, and $\times 1$ connections. The ports are all backwards compatible up to Generation 1.1a. The Hub chip supports "Hot plug" capability.

V. PERCS Topology

Two key design goals for PERCS were to dramatically improve bisection bandwidth (over other topologies such as fat-tree interconnects) and to eliminate the need for external switches. With these goals in mind, the Hub chip was designed to support a large number of links that connect it to other Hub Chips. These links are classified into two categories "L", and "D", that permit the system to be organized into a two-level direct-connect topology. Figures 3 and 4 illustrates these concepts.

Every Hub chip has thirty-one L links that are used to fully connect thirty-two Hub chips into a star topology. Within this group of thirty-two Hub chips, every chip has a direct communication link to every other chip. The Hub chip implementation further divides the L links into two categories: seven electrical LL links with a combined bandwidth of 336 GB/s and twenty-four optical LR links with a combined bandwidth of 240 GB/s. The L links bind thirty-two compute nodes into a supernode.

Every Hub chip also has sixteen D links that are used to connect to other supernodes with a combined bandwidth of 320 GB/s. The topology maintains at least one D link between every pair of supernodes in the system, although smaller systems can employ multiple D links between supernode pairs.

Since the Hub chip being connected to the POWER7 chips in the compute node at a bandwidth of 192 GB/s and has 40 GB/s of bandwidth for general I/O, the peak switching bandwidth of the Hub chip exceeds 1.1 GB/s. An interesting metric is the ratio of the injection bandwdith to/from the compute POWER7 chips and the network bandwidth. When all links are populated and operate at peak bandwidths, the injection bandwidth to network bandwidth ratio is 1:4.6. Note though that by performing the dual roles of switch and interconnect gateway, the majority of traffic through the Hub chip will typically be destined for other compute nodes.

The topology used by PERCS permits routes to be made up of very small numbers of hops. Within a supernode, any compute node can communicate with any other compute node using a distinct L link. Across supernodes, a compute node has to employ at most one L hop to get to the "right" compute node within its supernode that is connected to the destination supernode (recall that every supernode pair is connected by at least one D link). At the destination supernode, at most one L hop is again sufficient to reach the destination compute node.

VI. Routing

The above-described principles form the basis for direct routing in the PERCS system. A direct route employs a shortest path between any two compute nodes in the system. Since a pair of supernodes can be connected together by more than one D link, there can be multiple shortest paths between a given set of compute nodes. With only two levels in the topology, the longest direct route L-D-L can have at most three hops made up of no more than two L hops and at most one D hop.

PERCS also supports indirect routes to guard against potential hot spots in the interconnect. An indirect route is one that has an intermediate compute node in the route that resides on a different supernode from that of the source and destination compute nodes. An indirect route must employ a shortest path from the source compute node to the intermediate one, and a shortest path from the intermediate compute node to the destination compute node. The longest indirect route L-D-L-D-L can have at most five hops made up of no more than three L hops and at most two D hops. Figure 5 illustrates direct and indirect routing within the PERCS system.

A specific route can be selected in three ways when multiple routes exist between a source-destination pair. First, software can specify the intermediate supernode but let the hardware determine how to route to and then from the intermediate supernode. Second, hardware can select amongst the multipe routes in a round robin manner for both direct and indirect routes. Finally, the Hub chip also provides support for route randomization whereby the hardware can pseudo-randomly pick one of the many possible routes between a source-destination pair. Hardware-directed randomized route

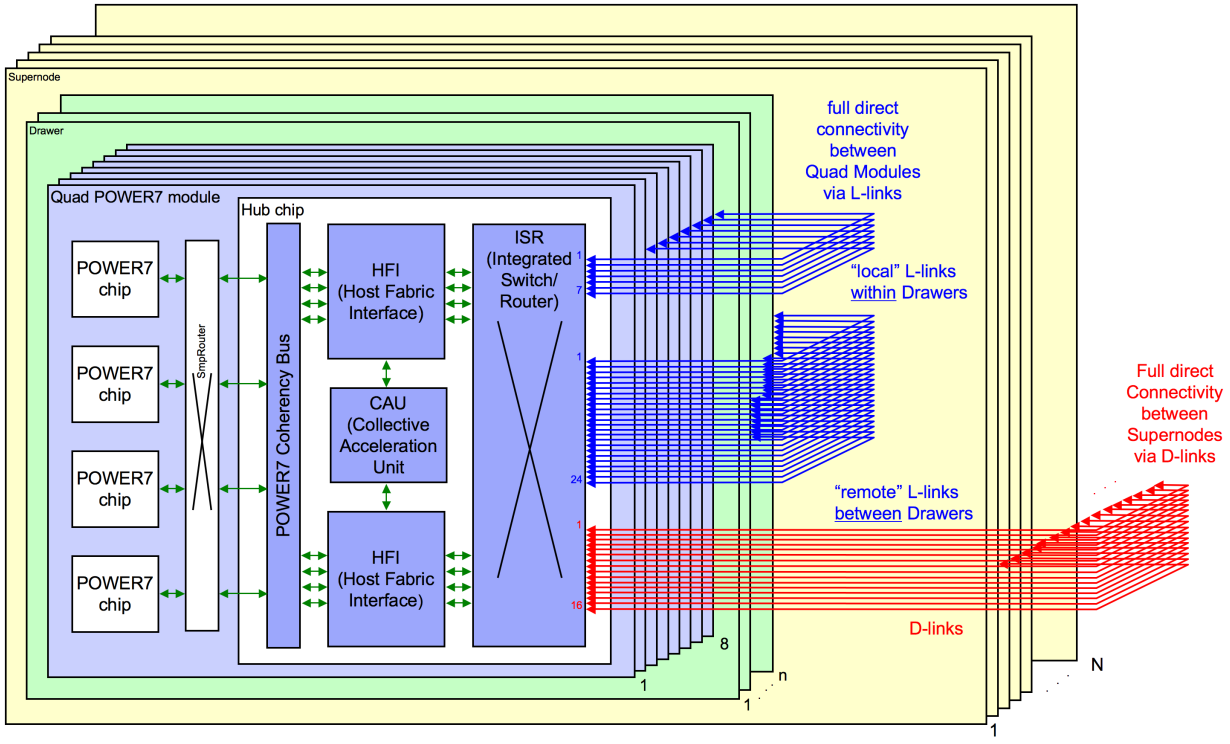

Figure 3. IBM Hub chip structure and interconnections

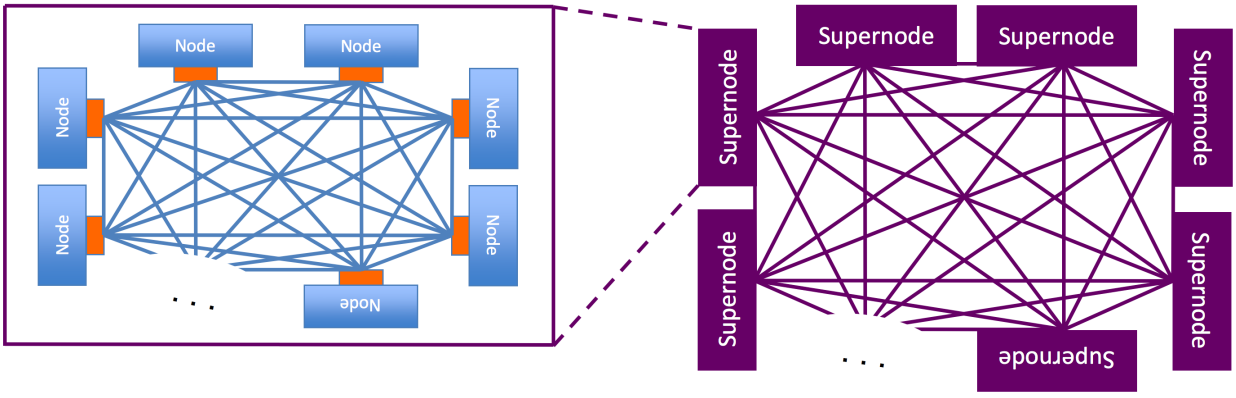

Figure 4. Direct connections between nodes in a supernode and supernodes in the system

selection is available only for indirect routes. These routing modes can be specified on a per-packet basis.

The right choice between the use of direct versus indirect route modes depends on the communication pattern(s) used by applications. Direct routing will be suitable for communication patterns where each node has to communicate with a large number of other nodes as with spectral methods. Communication patterns that involve small numbers of compute nodes will benefit from the extra bandwidth offered by the multiple routes with indirect routing.

Routing is accomplished using static route tables placed in the routers (ISR). These route tables are set up during system initialization and are dynamically adjusted as links go down or come up during operation. Packets are injected into the network with a destination identifier and the route mode. Route information is picked up from the route tables along the route path based on this information. Packets injected into the interconnect by the HFI employ source route tables. Per-port route tables are used to route packets along each hop in the network. Separate route tables are used for inter-

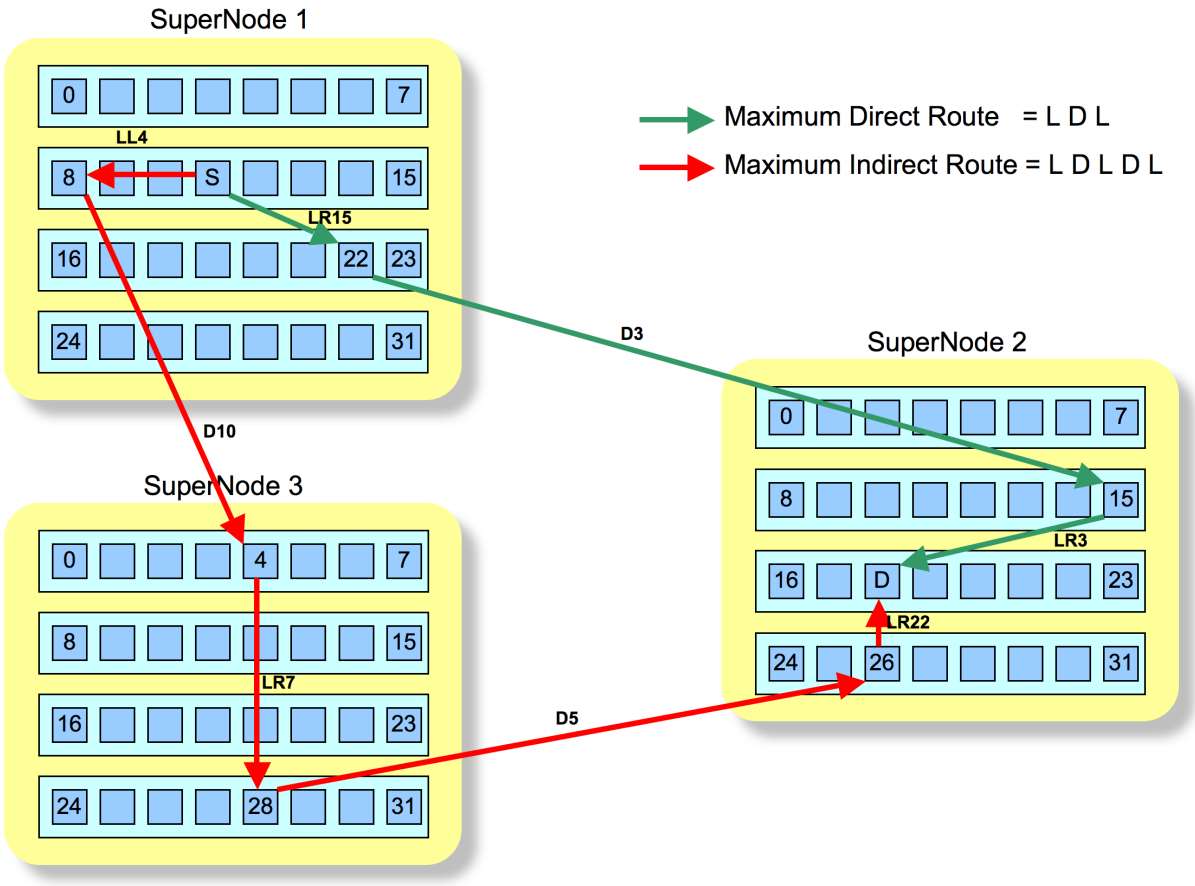

Figure 5. Direct and Indirect routes in PERCS

supernode and intra-supernode routes.

Virtual channels (VCs) are used to prevent deadlocks. Rather than use priorities, we use the position of the current hop within the full route to select which VC to use. Based on the worst case route in the system being L-D-L-D-L, there are three VCs assigned to L links and two VCs assigned to D-links.

Figure 6 shows how data flows within the PERCS system. The Integrated Switch Router (ISR) within the Hub chip employs cut-through and wormhole routing [4] with 128-byte FLITs. FLITs are assembled into packets which is the largest unit for which the hardware makes an ordering guarantee: all FLITs of a packet will be delivered in order. No ordering guarantees are provided between packets in a message. Thus even packets sent from the same source window (see Section IV) and node to the same destination window and node may reach that destination in a different order.

Figure 6 shows two Host Fabric Interfaces (HFIs) cooperating to move data from the POWER7s attached to one PowerBus to the POWER7s attached to another PowerBus through the interconnect. Note that the path between any two HFIs may be indirect, requiring multiple hops through intermediate ISRs.

In addition to the direct and indirect route tables, the ISR also has multicast route tables for replicating and forwarding IP multicast packets. All of the route tables are set up during system initialization by network management software. In the event of link or other failures, network management software is alerted and intervenes to reroute the system.

VII. Blue Waters—A Large-Scale Example

IBM and NCSA are working on constructing Blue Waters, a machine expected to achieve sustained Petascale performance for a large set of applications. Blue Waters will comprise more than 300.000 POWER7 cores, more than 1 PiB memory, more than 10 PiB disk storage, more than 0.5 EiB archival storage, and achieve around 10 PF/s peak performance. More information on Blue Waters is available at the Blue Waters project office [1].

A possible configuration could consist of several hundred supernodes (SN) with thousands of hub chips. Since the number of D links in an SN may not be an integral multiple of the number of other SNs in the system, the Hubs in an SN

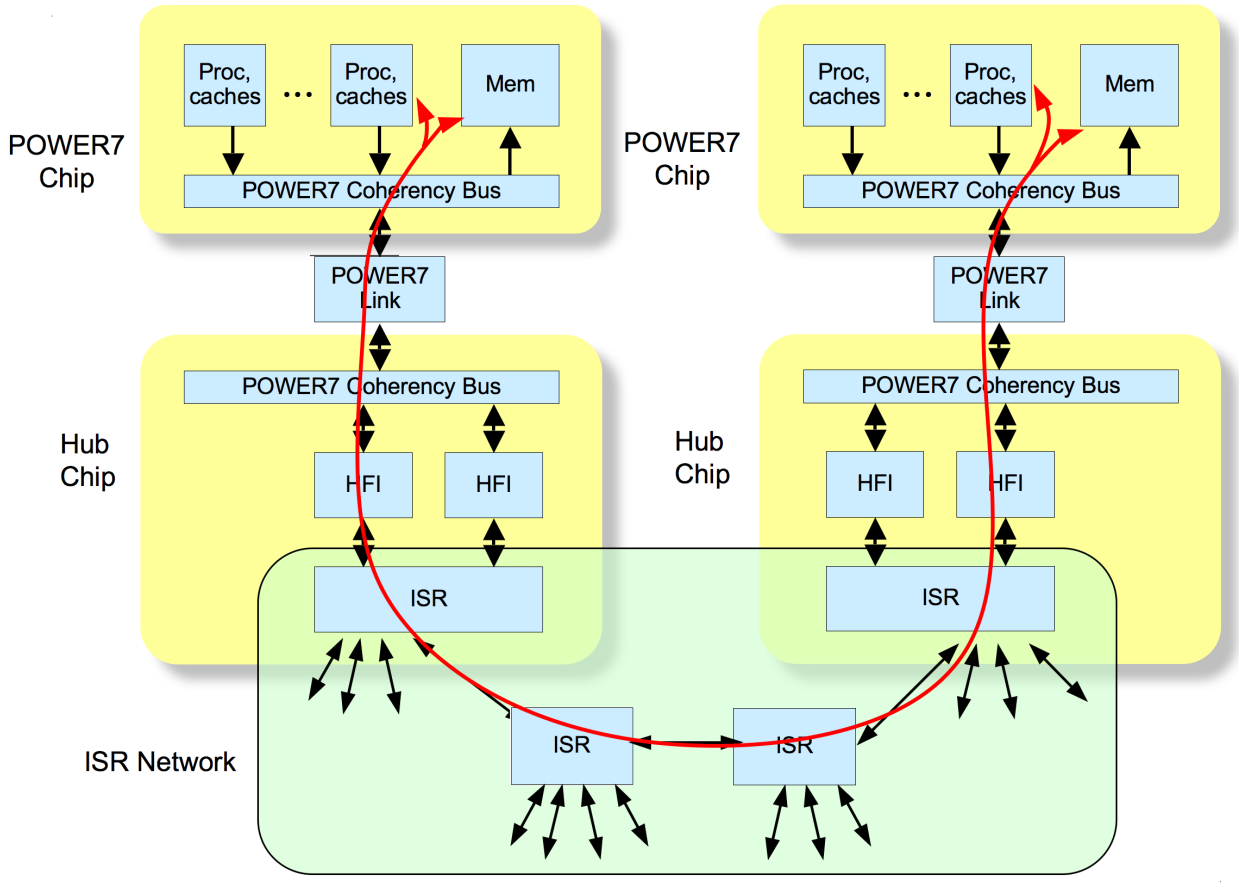

Figure 6. Packet flow in PERCS between two compute nodes. Note that data can both originate from and be written to caches on the source and destination compute nodes.

can differ in their D-link connections by one. For the number of SNs in Blue Waters, the ratio of injection bandwidth is expected to be close to that outlined in Section V.

A. *Effective Alltoall Bandwidth*

Alltoall is an important operation in parallel computing and imposes a high load onto the network. In this section, we derive a model for the effective alltoall bandwidth of the Blue Waters System. We derive an upper bandwidth bound with a simple counting argument assuming all communications happen simultaneously. First, we lead the argument for shortest-path static routing. From a single source, each compute node (CN) can be reached through a series of LL, LR and D links. We use only paths P that do not includes more than one D link or LL-LR, LR-LL, LL-LL, LR-LR connections. We denote $e(P)$ as the number of CNs that can be reached from one node through P. We assume that each Hub Chip is connected with d D-links to d distinct supernodes. Thus, $e(\text{LL}) = 7$, $e(\text{LR}) = 24$, $e(\text{D}) = d$, $e(\text{LL-D}) = e(\text{D-LL}) = 7d$, $e(\text{LR-D}) = e(\text{D-LR}) = 24d$, $e(\text{LL-D-LL}) = 49d$, $e(\text{LL-D-LR}) = e(\text{LR-D-LL}) = 168d$, $e(\text{LR-D-LR}) = 576d$, and $\sum_e(P) = 31 + 1024d$.

We can now count $c(L)$, the number of paths that lead through each LL, LR, and D link: $c(\text{LL}) = (e(\text{LL}) + e(\text{LL-D}) + e(\text{D-LL}) + 2e(\text{LL-D-LL}) + e(\text{LL-D-LR}) + e(\text{LR-D-LL}))/7 = 1+64d$, $c(\text{LR}) = (e(\text{LR})+e(\text{LR-D})+e(\text{D-LR})+e(\text{LR-D-LL})+e(\text{LL-D-LR})+2e(\text{LR-D-LR}))/24 = 1+64d$, and $c(\text{D}) = (e(\text{D})+e(\text{D-LL})+e(\text{LL-D})+e(\text{D-LR})+e(\text{LR-D}) + e(\text{LL-D-LL}) + e(\text{LL-D-LR}) + e(\text{LR-D-LL}) + e(\text{LR-D-LR}))/d = 1024$. This results in an effective bandwidth $b(L)$ per channel: $b(\text{LL}) = \frac{24\,GiB/s}{1+64d}$, $b(\text{LR}) = \frac{5\,GiB/s}{1+64d}$, and $b(\text{D}) = \frac{10\,GiB/s}{1024}$. The D-links seem to be the bottleneck of the alltoall for $d < 8$ while the LR-links are a bottleneck for $d \geq 8$. For $d \geq 8$, the effective alltoall bandwidth (limited by the slowest link) with shortest path routing would thus be $\sum_e(P) \cdot \frac{5\,GiB/s}{1+64d} = \frac{155+5120d}{1+64d}$ per CN. This is close to the injection bandwidth of a CN ($4{\cdot}24\,GiB/s = 96\,GiB/s$). We thus showed that the PERCS network topology with direct routing enables high-bandwidth alltoall communication on Blue Waters.

Indirect random routing would essentially half the alltoall bandwidth because it would perform a logical alltoall to

reach the intermediate supernode and then perform a second deterministic alltoall. Random routing is interesting because it can improve the worst-case congestion for other communication patterns. For example, communication from all 32 CNs in SN A want to communicate to different CNs in SN B would cause a congestion of 32 on the D-link between A and B. With random routing, each connection would bounce at a random SN which, with high probability, does not cause congestion on the D-link. Similar discussions can be lead for other communication patterns. We expect that the ideal routing scheme differs for different application classes. More detailed analyses are subject of current research.

B. Network Requirements of Petascale Applications

In the following, we briefly describe typical challenging requirements of Petascale applications and provide rough performance estimations. This short discussion shows how the described features of the PERCS network architecture are most critical to achieve Petaflop performance. Applications to be run on the system include Lattice QCD with a grid-size of $84^3 \cdot 144$ and a homogeneous isotropic turbulence code in a triply periodic box of size 12288^3.

For a high-performance implementation of Lattice QCD, the code running at full scale is expected to perform a global sum (allreduce) of two double complex values every 25-100μs. The CAU is capable of enabling Lattice QCD to be solved at this performance level and the offloaded global operation would even allow the application to hide the communication latency.

A three-dimensional Fast Fourier Transformation (3d FFT) is the most critical part in the Turbulence code. The 3d FFT is decomposed in two dimensions and requires alltoall communication along both dimensions. An example decomposition of a 8192^3 system into a $x \times y = 256 \times 32$ processor grid leaves 8192 pencils per CN. The computation would require 32 parallel alltoalls of size 4 MiB among 256 CNs and 256 parallel alltoalls of size 32 MiB bytes among 32 CNs. This mapping would map the y dimension into a SN while distributing the x dimension across SNs. The large-message communication would then use all 24 LR and 7 LL links per source simultaneously with the bandwidth of the slower LR links ($31 \cdot 5\,GiB/s$) which saturates the link bandwidth of the connections between the Hub chip and the POWER7 chips (assuming peak bandwidths). The small-message communication would use 8 D, 7 LL, and 24 LR-links to inject data simultaneously. Each SN communicates $32 \cdot 4\,MiB$ Bytes with each other SN over 256 D-links resulting in a bandwidth of $\frac{256 \cdot 10\,GiB/s}{32} = 80\,GiB/s$ per CN (the transfer is limited by the W links). Better mapping strategies for different layouts and sizes are subject of active research.

ACKNOWLEDGMENT

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002. This work is also supported by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of Illinois.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

The authors would like to thank Marc Snir, Bill Kramer, Jeongnim Kim, and Greg Bauer for helpful comments and discussions to improve Section VII.

REFERENCES

[1] Blue Waters Sustained Petascale Computing, Project Office. http://www.ncsa.illinois.edu/BlueWaters/, 2010. accessed July 2010.

[2] P. Charles, C. Grothoff, V. A. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) 2005*, pages 519–538, 2005.

[3] U. Consortium. UPC Language Specifications, v1.2. Technical report, Lawrence Berkeley National Laboratory, 2005. LBNL-59208.

[4] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[5] J. Dongarra and P. Luszczek. Introduction to the HPCChallenge Benchmark Suite. Technical report, ICL Technical Report, 10 2005. ICL-UT-05-01.

[6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, third edition, 2003.

[7] IBM. Power Instruction Set Architecture, 2009.

[8] MPI Forum. MPI: A Message-Passing Interface Standard. Version 2.2, September 4th 2009.

[9] R. W. Numrich and J. Reid. Co-array fortran for parallel programming. *SIGPLAN Fortran Forum*, 17(2):1–31, 1998.

[10] TOP500. http://www.top500.org/, 2006. accessed July 2010.