RC25366 (AUS1303-004) March 7, 2013
Computer Science

IBM Research Report

The Power 775 Architecture at Scale

R. Rajamony, M. W. Stephenson, W. E. Speight
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758
USA

This material is based upon work supported by the U. S. Defense Advanced Research Projects Agency (DARPA) under its Agreement No. HR0011-07-9-0002.

The Power 775 Architecture at Scale

Abstract

We describe the IBM Power 775, a supercomputing system that was designed to provide high performance at very large scales. The system recently attained world record performance numbers for three important, communication-heavy supercomputing benchmarks: RandomAccess, PTRANS, and FFT [1]*. At the heart of the Power 775's performance is the "hub module", which is a high-radix router containing forty-seven copper and optical links with a switching capacity of over 1.1 Tbyte/second. For the scale of the systems we have achieved, such bandwidth is unprecedented. As a result, we were forced to develop a complete software stack to fully leverage the communication capabilities of the system. In this paper we evaluate the Power 775 server at scales up to 2 Petaflops (63,360 POWER7 cores), discuss hardware and software tradeoffs considered during the design process, and finally present some post facto lessons learned.*

1 Introduction

The design of the Power 775 (henceforth referred to as P775) began in 2001 as part of a DARPA program called PERCS [21], and centered around the premise that high-performance communication would be the significant challenge for large-scale computing systems.

It is well known that several real-world analytics and supercomputing problems are communications bound. These problems spend more time moving data than they do operating on data. Indeed, we can see this effect today even on relatively small GPU systems with high-bandwidth connections to memory [16]. The problem of operand delivery is exacerbated as we increase the size of the system under consideration.

Consider an operation as conceptually simple as matrix transposition on which many algorithms rely. A parallel implementation of transpose may require a large fraction of the system to communicate, including potentially requiring every process to communicate with every other process. This is clearly a situation that becomes more expensive as the number of processes increase, and the maximum spatial distance between any two processes increases.

Much of the design effort for the P775 focused on the *hub module*, which is the heart of the system's communication fabric. The hub module is a high-radix router that switches data between sixteen 10 Gbyte/s optical links, twenty-four 5 Gbyte/s optical links, and seven 24 Gbyte/s copper links, with each link delivering the stated capacity in each direction. The hub chip switches *raw* data at over 1.1 Tbyte/second.

As demonstrated in November 2012, the P775 is over $3\times$ faster than its nearest competitor in parallel matrix transposition, and over $4\times$ faster on RandomAccess, a global memory performance benchmark [9]. These differences are even starker when the comparison is normalized and done on a per unit FLOP basis. On Global FFT, the P775 is only $1.6\times$ worse than the K computer while being $5.5\times$ smaller in peak capacity.

In this paper we describe the design of the P775 and evaluate it on three communication-heavy HPCC benchmarks [9] commonly used to gauge supercomputers. More specifically, we make the following contributions:

- We evaluate the P775 server "at scale" on workloads that traditionally perform poorly. This is the first paper to evaluate the P775 system at large scales.
- We describe how we attained world record performances for the communication heavy HPCC benchmarks: RandomAccess, PTRANS, and Global FFT.
- We learned many lessons through the course of this decade-long project. Of particular importance to this community, we describe system architectural features that we found useful, that we found detrimental, and that we wished we had.

The next section walks the reader through the system level design, starting with the microarchitecture. We also describe the compute node, other building blocks, and culminate with a description of an at-scale P775 server. Section 3 describes the communication-heavy, high-performance-computing workloads we use to analyze the P775, and we perform the analysis in Section 4. Section 5 is a retrospective discussion of lessons learned. We review related work in Section 6, and finally, Section 7 concludes the paper.

2 The Architecture

This section provides a bottom-up view of the system architecture. We start by describing the main compute engine of the P775, the POWER7 chip. Then we introduce the *compute node*, which is a collection of tightly interconnected POWER7s and a *hub chip*. We finally describe the P775's interconnection topology and routing schemes.

The entire system, including the processors, memories, and hub chips, is water cooled. Thermal waste dissipated

[1] While the Power 775 currently holds the number two spot in Global FFT, its efficiency when computing the FFT exceeds that of the number one system's by over $3.5\times$.

by purified water circulating over the system components is exchanged with the facility chilled water through a set of four water conditioning units at the bottom of each rack.

2.1 The POWER7

The POWER7 chip is the main compute engine of the P775. The POWER7 processor has been thoroughly described elsewhere [5, 14, 15], so we only discuss some of its most pertinent features here. The POWER7 chip has eight cores and each core has an independent set of inclusive L1, L2, and L3 caches [2]. The cores are superscalar, highly speculative, and support up to 4-way simultaneous multi-threading (SMT). The microarchitecture also implements a set of integer and floating point SIMD instructions, referred to as VSX (Vector Scalar eXtension). Using VSX, each core can process up to eight double precision floating point operations per cycle. There are two memory controllers in every POWER7 chip, which connect through eight memory channels to multi-ported DDR3 1333MHz DIMMs. The POWER7 hardware prefetchers maintain enough state to prefetch up to twelve concurrent memory streams.

2.2 The Compute Node (Octant)

With the computational building block of the P775 in place, we can discuss the *compute node*, which for historical reasons, is also called an *octant*. A compute node contains a *Quad Chip Module* (QCM) and a *hub chip*. The QCM, shown in Figure 1, contains four tightly connected POWER7 chips providing 32 cores within a Linux or AIX SMP image. The tight connection is provided by the XYZABC links. This name makes more sense in the context of Figure 1, where we can see the direct communication channel between all pairs of POWER7 chips in the QCM. The bandwidth of each of these links is 48 Gbyte/s in each direction.

A compute node supports up to 256 Gbytes of memory, and its 32 cores run at 3.84GHz, attaining a peak performance of 0.98 Tflop/s.

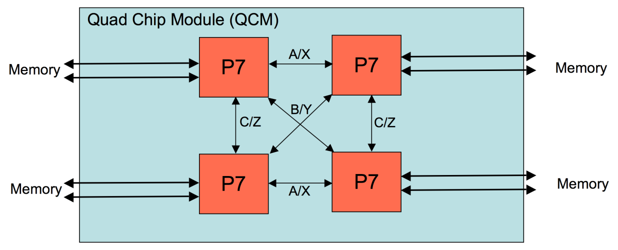

Figure 1. The XYZABC links in a QCM.

From a management perspective, it is important to note that a QCM runs a single operating system image [21], and at the level of a QCM, programmers can use posix threads and shared memory to orchestrate communication and computation.

[2]The L3 can also be shared among the cores, as noted in [6].

2.2.1 The Hub Chip

The compute node contains a fifth chip, the *hub chip*, that is the heart of the P775 architecture. The hub chip is a high-radix router that enables system sizes up to 16,384 compute nodes. The hub chip uses L_{local}, L_{remote}, and D links to connect to other compute nodes. We discuss these later.

The hub chip performs several important I/O functions and optimizations. It uses a high-performance coherent bus, the *PowerBus*, to fully participate in coherence operations that take place among the four POWER7 chips. Software can register portions of a process' effective address space with the hub chip's memory management unit. Consistency can then be maintained with the translation look-aside buffers (TLBs) on the QCM's cores. This important feature is the building block for using a global address space across the entire system. Specific packet types exist that can instruct a hub chip to atomically perform certain operations on registered memory without involving the local cores.

The hub chip also has PCI connections (two $16\times$ and a single $8\times$) that are used to connect to storage adapters. Storage disk enclosures are integrated into the P775 rack. Storage server functions are executed on QCMs set aside for that purpose. A description of the storage performance of the P775 is outside the scope of this paper.

Hub chips are the gateways for global communications. All *collective* operations (*e.g.*, barrier, reduce, broadcast, prefix sums, etc.) necessarily funnel through hub chips. To improve the performance of collective operations, the hub chip contains a *Collective Acceleration Unit* (CAU), which reduces the involvement of the POWER7 cores for many collective operations [24]. In the work we present here, we found the synchronization acceleration provided by the CAU to be especially useful.

Figure 2 shows a broader view of the QCM links (QCMs are on the blue planes). The links that connect the hub chip to the POWER7 chips are called *W* links, and they provide a bandwidth of 24 Gbyte/s in each direction.

The hub chip is 582 mm^2 in size, is implemented using 45 nm Cu SOI lithography, and contains 440 million transistors and 13 levels of metal [4].

2.3 Drawers

Figure 2 shows the P775 architecture at scales larger than a single compute node. A *drawer* is a collection of eight compute nodes (the green boxes in the figure). Physically, a drawer is 3.6" high (2U), 34.4" wide, and 54" deep, and weighs close to 350 lbs. The 256 cores in a drawer attain a peak performance of 7.86 Tflop/s.

Within a drawer, a set of copper $\mathrm{L_{local}}$ links provides direct connectivity between each pair of the eight compute nodes. These links operate at 24 Gbyte/s in each direction.

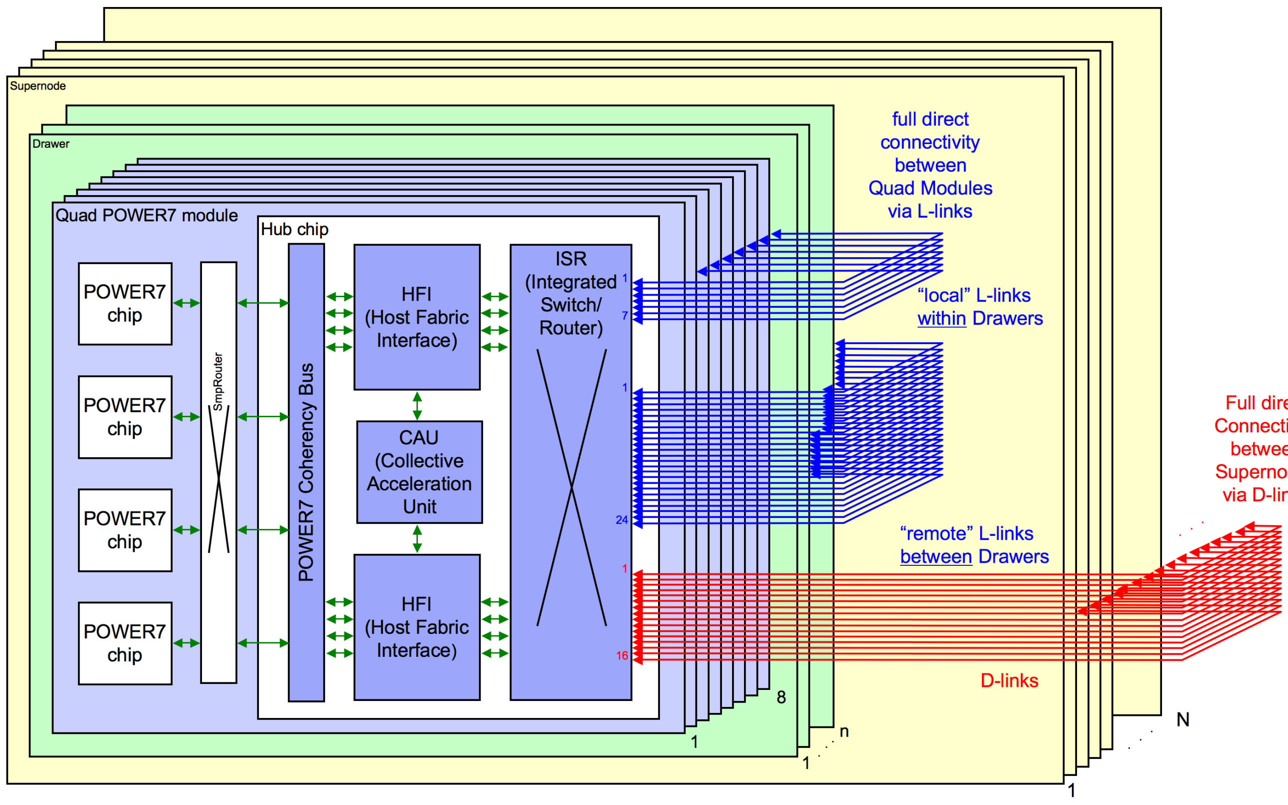
Supernode
Drawer
Quad POWER7 module
Hub chip
POWER7 chip
POWER7 chip
POWER7 chip
POWER7 chip
SmpRouter
POWER7 Coherency Bus
HFI (Host Fabric Interface)
CAU (Collective Acceleration Unit)
HFI (Host Fabric Interface)
ISR (Integrated Switch/ Router)
full direct connectivity between Quad Modules via L-links
"local" L-links within Drawers
"remote" L-links between Drawers
Full direct Connectivity between Supernodes via D-links
D-links
1
7
1
24
1
16
8
1
n
1
N
1

Figure 2. The P775 topology. A *QCM* consists of four POWER7 chips and a hub chip. A *drawer* contains up to eight compute nodes, and a *supernode* contains up to four drawers. There is full, direct connectivity between the compute nodes within a drawer (via local L-links), and between drawers (via remote L-links). Supernodes also have full, direct connectivity (via optical D-links).

2.4 Supernodes

In canonical form, four drawers comprise a supernode (the yellow boxes in the figure). A supernode therefore contains 32 nodes for a total of 1024 cores, and a peak performance of 31.4 Tflop/s. A rack can be configured to house two or three supernodes, with a two-supernode-per-rack configuration permitting more storage than the three-supernode-per-rack configuration.

Within a supernode, $\mathrm{L}_{\mathrm{remote}}$ links connect each pair of compute nodes in different drawers with a bandwidth of 5 Gbyte/s in each direction. The combination of $\mathrm{L}_{\mathrm{local}}$ and $\mathrm{L}_{\mathrm{remote}}$ links fully connects a compute node within a supernode to every other compute node within that supernode.

2.5 At Scale

The maximal configuration supported by the architecture is 512 supernodes, resulting in 16,384 nodes, 512K cores, and a peak performance of 16 Pflop/s.

Optical D-links provide full connectivity between each pair of supernodes. The bandwidth of each link is 10 Gbyte/s in each direction and they can be ganged up. Because it not obvious from the figure, we specifically point out that the topology was designed as a two-level interconnect that fully connects any two compute nodes in at most three hops [21].

2.6 Routing

The P775 moves packets using both *direct* and *indirect* routes [3]. A direct route employs one of potentially many shortest paths between any two compute nodes in the system. With only two levels in the topology, the longest direct route can have at most three hops.

Indirect routes provide increased point-to-point bandwidth and guard against potential hot spots in the interconnect. An indirect route has an intermediate compute node in the route that resides on a different supernode from that of the source and destination compute nodes. Thus, it is a composition of a shortest path from the source compute node to the chosen intermediate node, and a shortest path from the intermediate node to the destination node.

3 Communication-Heavy HPC Workloads

This section describes three communication-heavy workloads from the HPC Challenge benchmark suite [9, 19] that we consider in this paper. The motivation for using these in our evaluation is:

- These workloads are communication heavy and memory intensive. It is a common observation in the computer architecture literature that memory and interconnection bandwidths are not scaling nearly as fast as compute capabilities. These workloads nicely highlight the widening capability gap.
- These workloads can be heavily modified to take advantage of special features of the target platform. We were able to leverage the P775's architecture to elegantly tune these applications from the algorithm level down to the assembly level.
- Because these are standardized benchmarks, vendors and researchers can conveniently compare our results to those for other systems.

3.1 RandomAccess (GUPS)

Whereas some benchmarks, such as LINPACK, are primarily designed to measure the compute capabilities of a system, RandomAccess is designed to stress memory and interconnect performance.

Conceptually, a "global address space" is created that is made up of the memories of all the compute elements participating in the execution. Each compute element then launches a series of operations that update the values of locations within this global address space. The locations are chosen at random and could point to a location local or remote to the current compute element. An update consists of performing an "exclusive-or" operation between the existing contents of the memory location and a random value. The contents of the memory location do not have to be retrieved. Performance is measured in terms of Giga Updates Per Second (GUPS).

3.2 PTRANS

PTRANS (parallel matrix transpose) is a conceptually simple benchmark that performs the matrix operation

$$A = A^T + B \tag{1}$$

where A and B are random, double precision n by n matrices (*i.e.*, $A, B \in \mathbb{R}^{nxn}$). The stated purpose of PTRANS is to measure "the total communications capacity of the system interconnect" [19]. The data transfer rate is simply the size (in bytes) of matrix A divided by the time it takes to perform the operation in Equation 1.

It is interesting to note that the benchmark imposes strict requirements on the way data is distributed between the system's processes. The rules ensure that communication cannot be circumvented through an artful mapping between processes and compute elements. Figure 3 shows the salient points of the PTRANS data distribution including the notions of P and Q that determine how the data is tiled [7]. For the cases considered in this paper, we chose P and Q to be relatively prime, necessitating that every process communicate data to every other process in the system.

3.3 Global Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) is one of the most widely used computations in the world. It is used in a diverse set of fields, for computations as varied as convolution to solving partial differential equations. The FFT computes the Discrete Fourier Transform in time $\Theta(n \log n)$ [10].

Algorithm 1 The six-step recursive FFT algorithm.

```
function $F$ = FFT($f$)
1   $n$ = length($f$)
2   $[n_1, n_2]$ = factor($n$)
3   $M$ = reshape($f, n_1, n_2$)
4   $M_1$ = transpose($M$)              % Step 1: global all-to-all
5   for $k$ = 1:$n_2$                   % Step 2: local computation
6       $M_2(k,:)$ = FFT($M_1(k,:)$)    % Compute the FFTs along
7   end                                 % the rows of $M_1$
8   $M_3$ = transpose($M_2$)            % Step 3: global all-to-all
9   $M_4 = M_3 .* V_{n_1 \times n_2}$   % Step 4: twiddle factors [10]
10  for $k$ = 1:$n_1$                   % Step 5: local computation
11      $M_5(k,:)$ = FFT($M_4(k,:)$)    % Compute the FFTs along
12  end                                 % the rows of $M_4$
13  $F$ = transpose($M_5$)              % Step 6: global all-to-all
```

In this work we consider the widely used Cooley-Tukey algorithm, which is stated recursively in Matlab in Algorithm 1. Of particular importance for large-scale multiprocessor systems, note that there are three *global* transposes (requiring all-to-all communication), and two *local* compute-intensive phases.

The algorithm treats the 1D input f as a 2D $n1 \times n2$ matrix M, where $n1 \cdot n2 = length(f)$.[3] Thereafter, all of the local computation, including the recursive calls on lines 6 and 11, operates along the rows of the intermediate matrices. The explicit transpositions on lines 4, 8, and 13 optimally localize the data for subsequent computation [12]. With this view in mind, it becomes more evident how to map this algorithm to processes: each process "owns" a contiguous slab of the intermediate matrices so that all computation is process-local.

Notice that we do not include the bottom-out condition of the recursion. In the multiprocessor setting it is typical to relegate the recursive steps (lines 6 and 11) to a highly-tuned FFT library. Again, this is feasible because the memory required for these steps is local to the process.

[3]Without loss of generality, we assume row-major order. We also assume $length(f)$ is not prime.

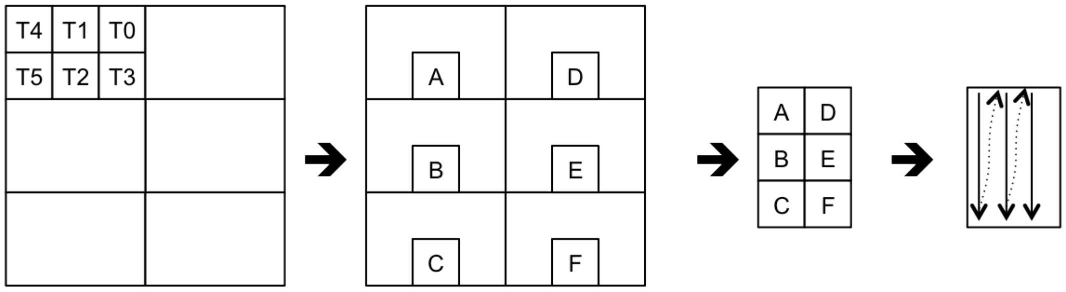

Figure 3. The A and B matrices are block-cyclic distributed with a randomized assignment of matrix chunks to processes. The processes are arranged as a $P \times Q$ grid (2×3 here). The three sub-figures on the right-hand side show the data "owned" by process T2 and how it is stored in column major order within its address space.

4 Results

This section discusses results obtained on a 64-supernode P775 system named Hurcules [4] with a peak performance of 2 Pflop/s. For each of the workloads, we highlight important aspects of the system's design, including software considerations, that were necessary for attaining high performance. Performance data on additional workloads is available on the HPC Challenge website [9].

We obtained a first set of results at the full Hurcules size involving 1,980 compute nodes running Linux.[5] In addition, we measured the scalability of RandomAccess and PTRANS on up to 1,472 compute nodes equating to 48 supernodes. We then infer the scalability of Global FFT using the measured PTRANS scalability and the performance of serial FFT on a single compute node.

4.1 RandomAccess

RandomAccess rules forbid sorting that transforms the random updates into regular (non-random) operations. Furthermore, no more than 1,024 updates can be buffered at a compute element at any given time. On large systems, the buffered updates are almost all bound for distinct destinations. The traditional high-performance solution to RandomAccess introduces software-based dimension-ordered routing that aggregates individual updates into larger packets bound to the same region of the larger system [13]. Such a scheme is inefficient because it uses more interconnect bandwidth than required and because of the software overheads in breaking up and reassembling packets at intermediate destinations.

On the P775, we implement RandomAccess as a multi-threaded program running concurrently on all compute nodes. At startup, the software "registers" the local portion of the RandomAccess table with the local hub chip. The threads craft packets (each specifying a distinct update operation) placing them in send FIFOs, which are data structures from where the hub chip picks up packets. Each packet is 32-bytes in size and specifies the effective address of the remote location, the compute node housing the location, and the operand to be used in the update. The local hub chip sends the packets to the remote compute node's hub chip, where the update is performed. The hub chip hardware is optimized to move 32-byte atomic read-modify-write packets that are aggregated (dis-aggregated) into (from) 128-byte network *flits* at intermediate hub chips.

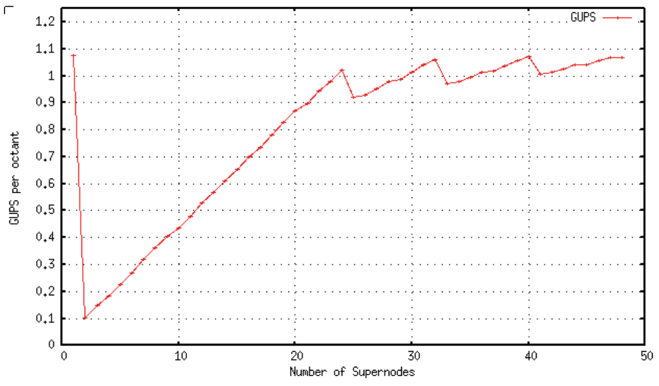

Figure 4. RandomAccess scaling on Hurcules. The Y-axis is in terms of GUPS per compute node (System GUPS performance *divided* by the number of nodes used) The 1.08 GUPS/node asymptote indicates linear performance scaling.

Figure 4 shows the performance of RandomAccess as the number of supernodes is varied. The Y-axis shows the system performance in Giga Updates Per Second (GUPS) divided by the number of compute nodes. A flat line on this graph indicates that an incremental increase of compute power yields a linear increase in performance. Indeed, the graph asymptotes to a performance of 1.08 GUPS per compute node. No known previous system has exhibited this level of performance.

The P775 has an abundance of interconnect links and interconnect bandwidth. Every pair of supernodes on Hurcules is connected by eight D optics links, providing a bandwidth

[4]The machine name is indeed (mis)spelled Hurcules [*sic*].

[5]Even though a 64-supernode system contains $64 \times 32 = 2,048$ compute nodes, a small number of these are set aside for storage, login, and fail-in-place.

of 80 Gbyte/s per supernode pair in each direction. However, the achievable bisection bandwidth of a two-supernode subset of Hurcules is far higher, since indirectly routed traffic can bounce off any of the remaining 62 intermediate supernodes in the system (see Section 2.6). An express goal of our scaling studies was to determine where the interconnect stops becoming a bottleneck. We achieve this by deliberately considering only direct routing.

Thus, a dramatic performance drop can be seen when going from one supernode (where the performance is 1.08 GUPS per compute node) to two supernodes (where the performance is 0.1 GUPS per compute node). RandomAccess performance steadily increases *linearly* as more supernodes are added to the system, continuing until the interconnect is no longer the bottleneck at 24 supernodes.

Beyond 24 supernodes, the performance starts oscillating in a "sawtooth" pattern that gradually trends upwards. The reason for the drop in GUPS per compute node from 24 to 25 supernodes, from 32 to 33 supernodes, and from 40 to 41 supernodes as well as the rising trend of the overall GUPS per compute node has to do with how non-local RandomAccess traffic is routed.

The first hop taken by a non-local RandomAccess flit (i.e., a RandomAccess flit that is destined for a different supernode) is always a $\mathrm{L}_{\mathrm{local}}$ hop. This link also has a limit on how much traffic it can carry. When switching from 24 to 25 supernodes, there is one $\mathrm{L}_{\mathrm{local}}$ link that has to carry twice the traffic as the other links within that drawer. As the number of supernodes increases, the proportionally more traffic to be carried by the one $\mathrm{L}_{\mathrm{local}}$ link as compared to the other links in the drawer decreases. In other words, the imbalance in $\mathrm{L}_{\mathrm{local}}$ link traffic reduces as the number of supernodes increases. We expect the asymptote to eventually become the 1.08 GUPS obtained on a single supernode.

4.2 PTRANS

The P775 has very high raw interconnect bandwidth. The main challenge with obtaining high-performance on the parallel matrix transpose is in delivering that bandwidth through the communications stack to the workload. We achieved this by implementing an elegant solution for dealing with the boundary condition of packet losses.

In the absence of fully reliable hardware packet transport, protocol stacks typically make a local copy of all transmitted data, and use a timer-driven, acknowledgment-based mechanism to perform re-transmissions and garbage collect copies of the sent data. On the P775, there are two reasons why a data packet sent to/from the hub chip FIFOs [3] can get lost. First, the hardware optical laser transceiver can itself fail. This is rare (with a predicted MTTF of around a week) for which an out-of-band checkpoint-restart technique is apt. The more prevalent reason for packet losses is due to the receive FIFO into which the hub chip deposits packets [3] becoming full. This happens when the receiving software is not emptying a receive FIFO as fast as the hub chip is filling it. Packets are dropped in this situation to prevent traffic from backing up and unfairly causing other parts of the system (perhaps executing other workloads) to get congested.

Rather than add a re-transmission protocol to handle such dropped packets, we devise a solution to eliminate the possibility of the receive FIFO ever getting full. We break up the data transfer into "rounds" that only transfer as much data as can fit into pre-allocated receive FIFO buffers. We set aside 4 Gbytes of buffer space on every compute node, equal to slightly over 3% of the node's physical memory. Because the size of the data being transferred is known a priori, each process is able to precompute the exact receive FIFO at the destination for each packet it will send. The receivers can also calculate exactly how much data will arrive on each FIFO.

After all data is sent for a round and after it has all been received, each process enters a global barrier that precludes buffer overruns. Upon exit from the barrier, all processes begin the procedure again until all rounds are completed. This algorithm makes use of the fact that on the P775, the time to carry out a round (i.e., pump out 4 Gbytes) far exceeds the time to perform a global barrier using the CAU.

To eliminate hot spotting in the interconnect, we would like to randomize the destinations to which packets are sent. Interestingly, the data distribution adopted by PTRANS already randomizes the mapping of data blocks to tasks; although this was adopted to discourage solutions that reduce data transfer through clever mappings of tasks to physical system nodes. Traversing the A matrix in order is therefore sufficient to ensure that the packets being sent are bound for different destinations. We revisit this aspect when discussing the transpose phase of the Global FFT.

We further simplify our implementation by eliminating the need to maintain state beyond that which can be encapsulated in a single packet. The largest packet that can be sent on the P775 interconnect is 2048 bytes (of course, software can always compose longer messages using multiple packets). We limit ourselves to sending no more that 2048 bytes of data and metadata, which a receiver uses to deposit received data into the correct places within its address space. More specifically, we use a PTRANS block size of 15×15 to ensure that each such block can be both transported within an indivisible P775 packet and dealt with independently by the receiver. Each block and its associated metadata fits within a single 1920-byte packet.

Figure 5 shows the scalability of PTRANS between one and forty-eight supernodes. As in the case of RandomAccess, the Y-axis on the graph shows the system performance divided by the number of compute nodes. A flat line on this graph indicates that an incremental increase of compute power yields a linear increase in performance. The graph asymptotes to a performance of over 30 Gbyte/s per compute node. This level of performance is far superior to that exhibited by other high performance computing systems [9].

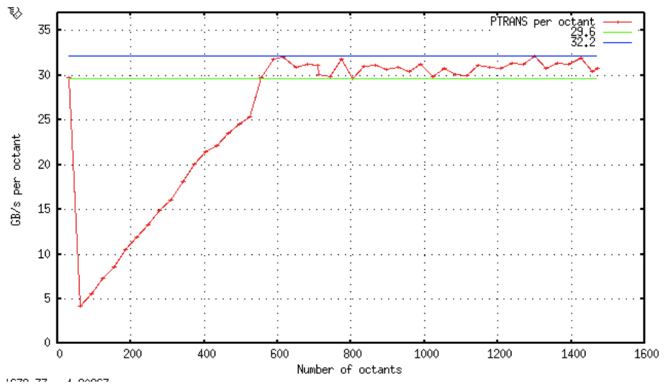

Figure 5. PTRANS scaling on Hurcules. The Y-axis is in terms of PTRANS per compute node (system PTRANS performance *divided* by the number of compute nodes used). Linear scaling is clearly evident from the performance asymptote.

As with RandomAccess, performance falls dramatically when going from one to two supernodes *simply because in Hurcules, directly routed traffic can only use the eight physical wires connecting the two supernodes.*

At one supernode, the system performs at 919 Gbyte/s corresponding to 29.7 Gbyte/s/octant. At two supernodes, the performance falls to 260 Gbyte/s, corresponding to 4.2 Gbyte/s/octant. As the number of supernodes increases beyond two, PTRANS performance steadily increases on a *per-supernode basis*. This happens until we reach 19 supernodes at which point the system has sufficient bisection bandwidth such that the D-link capacity is no longer a bottleneck.

Beyond 19 supernodes, there is performance variation at different supernode levels, but the per-octant PTRANS performance stabilizes to be $\approx$ 30 Gbyte/s/octant. The sawtooth shaped performance behavior is not seen in G-PTRANS because the local L-links offer the full link capacity for carrying data, as opposed to being gated by the packet recombination rate for 32-byte RandomAccess packets.

4.3 Global FFT

The Global FFT in HPC Challenge requires a single-dimensional double-precision complex–to–complex FFT to be performed by all compute elements. The input vector is distributed over all nodes and the result vector is required to be deposited back in–place. As mentioned in Section 3.3, we treat the large single-dimensional input vector as a two-dimensional matrix, and we follow Algorithm 1. We choose HPC Challenge parameters such that the matrix is approximately square.

We use FFTW 3.2 [11] to compute the row FFTs (lines 6 and 11 in the algorithm), which we modified to support POWER7's VSX SIMD instruction set [5]. Load balancing while computing the row FFTs is achieved by having the cores compete for work.

The three global transpositions (lines 4, 8, and 13 in the algorithm) are similar to that used in PTRANS. We use an internal block size of 8×8 during the transpose operation (note that $8 \times 8 \times 16 = 1024$, which is again smaller than the 2048-byte packet limit even after accounting for necessary software headers. Interestingly, the P775 interconnect performs as well for packets of this size compared to larger 2 Kbyte packets.

During the transpose phase of the Global FFT, each process must communicate with every other process in the system. To avoid interconnect hot spots, we use a software architected pseudo-randomized routing scheme. Each process randomly traverses the set of all processes with which it must communicate– care being taken to ensure that every node uses a different pseudo-random traversal through the node set. Our implementation is particularly simple. Let N be the total number of tasks with which a given task must communicate. We provide each task with a number P (taken from a large statically constructed list of prime numbers) that is both relatively prime to N as well as different from the value of P used on any other task. The task set is ordered exactly the same way on each node, but in iteration i each node picks task $P \cdot i \bmod N$ in the ordered set with which to communicate. Of course, many other randomization schemes are possible. We also experimented with a regular routing scheme where every node traverses the task set starting with the node to its "right". Such a scheme is oft-advocated but performed 12% worse than our randomized scheme on the single supernode upon which we were able to test.

Table 1. Performance of Global FFT at different scales

System size (compute nodes)	Raw performance (Gflop/s)	Percentage of peak
8	512.1	6.52%
1,458	94855.8	6.63%
1,944	132658.1	6.95%

Table 1 shows the performance of Global FFT at different system sizes (HPCC constrains node counts to be expressible as $2^i.3^j.5^k$ for integer i, j, k.). Unlike RandomAccess and PTRANS, we were unable to obtain a full scaling curve due to the prohibitive time it would have taken to plan the FFTW component at each of the different data set sizes.

The asymptotic Global FFT performance on large P775 systems can be calculated as a composition of the performance for each stage of Algorithm 1. The performance for the two compute phases (lines 5–7 and 10–12) depends on how long FFTW takes to compute the associated "slabs" of rows on the POWER7 compute node. The performance for the three transpose phases is determined by the asymptotic PTRANS performance of 30 Gbyte/s/compute node. Note that the extra floating point operations in line 9 for performing the so-called "twiddle" multiplications get transparently folded in to the transpose operation before data blocks are written out. These operations can be trivially overlapped with the memory loads that bring data into the caches.

Let n be the size of the Global FFT being solved. FFT efficiency calculations assume that the FFT itself requires $5 \cdot n \cdot \lg n$ floating point operations. Let P be the number of nodes, F be the floating point execution rate (flops/s) per node when executing the serial compute phases, η_s be the efficiency when computing the serial portion of the FFT, and B represent the bandwidth that each node can sustain during the transpose phase (we know from figure 5 that B asymptotes to 30 Gbyte/s). Note that each FFT data point is double-precision complex and 16 bytes in size. Then:

$$
\begin{aligned}
&\text{Time for each compute phase } = 5 \cdot n \cdot \lg n/(F \cdot P) \\
&\text{Time taken by each transpose } = 16 \cdot n/(B \cdot P) \\
&\text{Total time } = 3 \times \text{TransposeTime } + 2 \times \text{ComputeTime} \\
&\qquad\qquad = 48 \cdot n/(B \cdot P) + 10 \cdot n \cdot \lg n/(F \cdot P) \\
&\text{FLOP rate } = \text{Floating point operations /Time taken} \\
&\qquad\qquad = 5 \cdot \lg n \cdot B \cdot F \cdot P/(48 \cdot F + 10 \cdot B \cdot \lg n) \\
&\text{Efficiency } = 5 \cdot \lg n \cdot B \cdot \eta_s/(48 \cdot F + 10 \cdot B \cdot \lg n)
\end{aligned}
$$

For the weakly scaled case, the size of the FFT is proportional to the number of nodes used in the computation. Then, $\lg n = k + \lg P$, where k is some constant. Assume that each node can contribute 2^{30} data points, corresponding to 16 Gbytes of data. We can later observe that our results are not sensitive to this assumption. Each node or compute node can also compute at a peak rate of 982 Gflop/s. We can then compute the efficiency for the global FFT:

$$
\begin{aligned}
\eta_g &= (5.30.(30 + \lg P.\eta_s))/(48.982.\eta_s + 300.(30 + \lg P)) \\
&\approx 6.8\% \text{ when } P = 1000
\end{aligned}
$$

A few interesting observations can be made from the above equation. First, since we compute the serial FFT on only either the row or the column, and since the 1D vector is structured as a roughly square matrix, an increase in the 1D vector length by a factor of f only increases the size of the serial FFT by $\sqrt{f}$. On the POWER7, η_s is around 30% for n corresponding to what will fit in the memory of up to a few thousand nodes. Solving the above equation, we see that the efficiency of the global FFT changes very slowly as the number of nodes is changed; the insensitivity to the constant k can also be immediately grasped.

4.4 Analysis

Hurcules the maximal P775 system we were able to access, was made up of 64 supernodes with a peak performance of 2 Pflop/s. At this scale, the RandomAccess, PTRANS, and Global FFT performances were 2,021 GUPS, 60.5 Tbyte/s, and 132.7 Tflop/s respectively. By itself, the raw performance on RandomAccess and PTRANS places the P775 at the number one spot in the world. However, comparing the 2 Pflop/s Hurcules system we experimented upon against larger production systems such as the K computer [6] or the Cray XT5 system at ORNL[6] is inherently unfair. A better technical comparison would be to normalize the performance on these workloads against peak system performance, especially since we have empirically shown linear performance scaling on these three workloads. Figure 6 does exactly that. The dramatically improved efficiency of the P775 is very clear in this figure, which shows normalized performance against peak flops that various systems have been able to achieve over time, on the HPC Challenge benchmarks.[7]

5 Heroic Programmer's Retrospective

This section describes some interesting observations and lessons learned while designing, developing, and evaluating the system. We mention up front that our goal was to tease as much performance out of the system as possible. From a software point of view, this involved continuous tweaking, and redesign. A common theme among the lessons we learned is that architectural and software features designed to aid programmer productivity can severely inhibit performance for heroic programmers. In a few cases we decided to bypass architectural features that are in place to relax the burden on software. Likewise, we also avoided some programmer productivity libraries such as MPI and PAMI. Given the context of this paper, this should not be viewed as a condemnation of these features. However, we hope this discussion leads to a more automatic middle ground that strikes a better balance between programmer productivity and optimal performance.

5.1 Core Architecture Observations

Somewhat surprisingly, simultaneous multi-threading was not useful for any of the workloads we ran, and in many cases hurt performance. Traditionally, SMT helps performance by permitting the core to overlap instruction execution with high latency instructions. We designed our codes with appropriate software–inserted prefetches to make demand load latencies to generally not be a performance bottleneck. Consequently, the resource division that occurs with simultaneous multithreading [23, Page 1:3 "POWER7 Core"] far outweighed any latency hiding advantages. For less carefully designed code, SMT is likely to be beneficial.

Reinforcing much related work (*e.g.*, [20]), we found the VSX SIMD extensions to be useful for the FFT (as well

[6]http://www.aics.riken.jp/en/ and http://www.olcf.ornl.gov/

[7]BG/Q's recently published results of 858 GUPS on a 3.36 Pflop/s system still place it at only 0.00026 GUPS/Peak GFLOP.

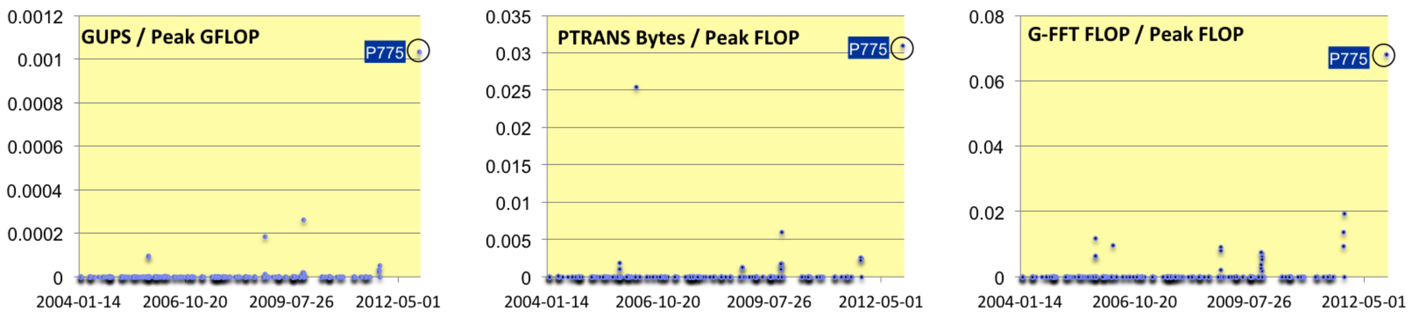

Figure 6. Historical normalized performance per peak flop for a variety of different systems.

as for HPL and DGEMM, which we have not addressed in this paper). For the sizes of FFTs we consider in this paper, our VSX-enabled FFTW-3.2 library is $\sim$18% faster than its scalar counterpart on POWER7.

For the benchmarks on which we have evaluated the P775, hardware prefetching was not strictly required and was often a hindrance. Specifically, for the benchmarks surveyed in this paper, RandomAccess is random by design, and therefore hardware prefetchers hurt performance by squandering memory bandwidth on data that will never be referenced. PTRANS and FFT do have predictable strides for which it seems reasonable to presume that hardware prefetching will prove useful. In fact, the POWER7 architecture somewhat impressively maintains enough hardware state for prefetching up to 12 different streams.

However, we discovered that in the case of PTRANS, array subtiles larger than 12×12 were necessary in order to get sufficient communications performance; such subtiles rendered the 12–stream hardware prefetcher useless. An execution of PTRANS on a 1.7 Pflop/s subset of Hurcules with hardware prefetcher enabled *realized a 15% performance loss*, showing how system–level issues (in our case communication) outweigh micro-level design decisions. A more beneficial approach was to explicitly prefetch required data through hand-inserted software prefetch instructions. The interaction of software and system-level issues is often very complicated, making a compelling argument that hardware focus more on providing strong support for explicit software prefetching.

On the positive side, the POWER7 architecture provides a mechanism to change the mode of the prefetcher on-the-fly. We used this feature to great effect in the case of Global FFT, where we placed the cores in full hardware prefetch mode during the sequential compute phases and then turned them off when moving on to the transpose phases. We found the combination of hardware feature and Linux kernel support that achieved this be particularly elegant and eminently useful.

5.2 Large-scale Observations

While communications libraries such as MPI simplify programming, using them incurs a significant amount of overhead. In particular, the copy and garbage collection overheads involved in the use of such stacks (see Sections 4.2 and 4.3) eats away precious memory bandwidth that is no longer available to the application. We used a custom designed software stack that more tightly integrates communication with the application logic; such stacks need to be further explored. [8]

When evaluating all-to-all communications on the machine, we noticed that naively communicating between all pairs of processes creates hotspots. While randomized routing is an effective approach for reducing hotspots [17], it requires additional hardware support. We instead advocate moving the complexity to software. Our all-to-all communications randomize the $\langle source, dest \rangle$ pairs involved in communication at the algorithm level to avoid network contention. The approach performs well and is quite practical, as it still relies on deterministic routing (see Sections 4.2 and 4.3). To the best of our knowledge, this approach to alleviating hotspots is novel.

6 Related Work

Various aspects of the P775 system have previously been discussed. The POWER7 chip has been described in [15, 14]; the network characteristics of the system are discussed in [3, 24]. Finally, the performance of an early, drawer-level version of the P775 has been previously evaluated [6, 21]. At 64 supernodes and 2 Pflop/s, this system is significantly larger than those in earlier studies [6, 21]. Furthermore, the benchmarks we consider in this paper significantly stress the interconnect.

The P775 hub chips are high-radix routers: Even for the largest supported configurations, the shortest path between any two nodes is at most three hops. Other notable implementations of high-radix routers include Cray's BlackWidow system [1, 22], and the Cray Dragonfly network [2, 17]. As Kim et. al note in [17] where they introduce the Dragonfly topology, high-radix routers simultaneously reduce the diameter and the latency of interconnection networks.

To further improve the interconnect performance, the P775 hub chip has hardware support for collective communication operations, and supports several high-performance

[8]We have also used such an integrated stack in the design of our Graph500 implementation [8], a non-trivial workload.

computing protocols [3, 24]. The Blue Gene/Q interconnect also has hardware acceleration for collective operations [18].

Unlike the Dragonfly topology, the P775 does not employ dynamic routing. Instead, the hub chip implements deterministic and hardware-directed randomized indirect routing [3]. We use deterministic routing for the experiments presented in this paper; however, we developed a randomized algorithm for the all-to-alls in PTRANS and FFT that substantially reduces network congestion.

7 Conclusion

We rigorously evaluated and analyzed large-scale configurations of the IBM P775 on three HPC Challenge benchmarks. The benchmarks particularly stress a system's interconnect, and require optimization from the algorithm level down. Several interesting features of the P775 (the hub chip's high-radix routing, the Collective Acceleration Unit, software prefetching, etc.) allowed us to extract a high level of performance from these benchmarks. In fact, the P775 currently holds the top spot for PTRANS and RandomAccess. As we have seen for these applications, the P775 especially shines when we consider system efficiency.

Enabling peak performance of these benchmarks on the P775 taught us much about the system. We opted to write our own application-specific communication layers to avoid the overhead of libraries like MPI. Though the effort involved was significant, the protocols we devised were extremely thin and effective; further improvements were gained by crafting randomized versions of our protocols. In a couple of cases we were surprised to discover that for our carefully written code, ordinarily beneficial hardware features consistently hindered performance. Though we couch our lessons learned in the context of heroic programmers, they are hopefully of interest to the broader architecture community.

References

[1] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, T. Johnson, M. Bye, and G. Schwoerer. The Cray BlackWidow: a highly scalable vector multiprocessor. In *Proceedings of ACM/IEEE Supercomputing (SC)*, 2007.

[2] B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard. Cray Cascade: a Scalable HPC System based on a Dragonfly Network. In *Proceedings of ACM/IEEE Supercomputing (SC)*, 2012.

[3] 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. In *2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI)*, pages 75 –82, Aug 2010.

[4] B. Arimilli, S. Baumgartner, S. Clark, D. Dreps, D. Siljenberg, and A. Maki. The IBM POWER7 HUB Module: A Terabyte Interconnect Switch for High-Performance Computer Systems. *Hot Chips 22*, 2011.

[5] R. X. Arroyo, R. J. Harrington, S. P. Hartman, and T. Nguyen. IBM POWER7 systems. *IBM Journal of Research and Development*, 55(3):2:1 –2:13, May-June 2011.

[6] K. J. Barker, A. Hoisie, and D. J. Kerbyson. An early performance analysis of POWER7-IH HPC systems. In *Proceedings of ACM/IEEE Supercomputing (SC)*, 2011.

[7] J. Choi, J. Dongarra, and D. Walker. Parallel matrix transpose algorithms on distributed memory concurrent computers. *ORNL Tech Report ORNL/TM-12309*, October 1993.

[8] G. Committee. *Observe P775 having number two rank*. http://www.graph500.org/results_june_2012.

[9] H. Committee. HPC Challenge Benchmark Results. http://icl.cs.utk.edu/hpcc/.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[11] M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 2005.

[12] M. Frigo, C. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual FOCS Symposium* . IEEE, 1999.

[13] R. Garg and Y. Sabharwal. Optimizing the HPCC randomaccess benchmark on blue Gene/L Supercomputer. *Proceedings of SIGMETRICS/Performance*, 2006.

[14] R. Kalla and B. Sinharoy. POWER7: IBM's Next Generation Server Processor. *Hot Chips 21*, 2010.

[15] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd. POWER7: IBM's next-generation server processor. *IEEE Micro*, 30(2):7–15, 2010.

[16] S. Keckler, W. Dally, B. Khailany, M. Garland, and D. Glasco. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7 –17, Sep-Oct 2011.

[17] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, 2008.

[18] S. Kumar, A. Mamidala, D. Faraj, B. Smith, M. Blocksome, B. Cernohous, D. Miller, J. Parker, J. Ratterman, P. Heidelberger, D. Chen, and B. Steinmacher-Burrow. PAMI: A Parallel Active Message Interface for the Blue Gene/Q Supercomputer. In *26th International Parallel and Distributed Processing Symposium (IPDPS),*, May 2012.

[19] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi. The HPC Challenge (HPCC) benchmark suite. In *Proceedings of ACM/IEEE Supercomputing (SC)*, 2006.

[20] H. Nguyen and L. K. John. Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology. In *Proceedings of the 13th International Conference on Supercomputing (ICS)*, pages 11–20. ACM, 1999.

[21] R. Rajamony, L. B. Arimilli, and K. Gildea. PERCS: The IBM POWER7-IH High-Performance Computing System. *IBM J. Res. Dev.*, 55(3):233–244, May 2011.

[22] S. Scott, D. Abts, J. Kim, and W. J. Dally. The BlackWidow High-Radix Clos Network. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006.

[23] B. Sinharoy and et. al. IBM POWER7 multicore server processor. *IBM Journal of Research and Development*, 55(3):1:1 –1:29, May-June 2011.

[24] G. I. Tanase, G. Almási, H. Xue, and C. Archer. Composable, non-blocking collective operations on POWER7 IH. In *Proceedings of the 26th ACM International Conference on Supercomputing (ICS)*, ICS '12, pages 215–224. ACM, 2012.