IBM OpenSHMEM Implementation over the Parallel Active Messaging Interface (PAMI)

Alan Benner, bennera@us.ibm.com , STG/RES, IBM

Tsai-Yang (Alan) Jea, HPC Protocols, STG, IBM

Planned Outline

- Intro to Power 775 (PERCS) hardware and system
- PAMI - Parallel Active Messaging Interface - Overview
- OpenSHMEM over PAMI - Basic Operations
- OpenSHMEM over PAMI on Power 775 - Details
- Performance: OpenSHMEM / PAMI / Power 775

Power775 Systems (aka "PERCS")

- **All data center power & cooling infrastructure included in compute/storage/network rack**
  - Water-cooling of all components (CPUs, DRAM, optics, disks, & power conversion) - 1.18 PUE
  - Integrated "one-window" management for all compute, storage, network, power, & thermal resources.
  - Scales to 512K P7 cores (192 racks) – just add optical fiber cables

Integrated Power Regulation, Control, & Distribution

Runs from any building voltage supply world-wide (200-480 VAC or 370-575VDC), converts to 360 VDC for in-rack distribution. **Full in-rack redundancy and automatic fail-over**, 4 power cords. Up to 252 kW/rack max / 163 kW Typ.

Integrated Storage – 384 2.5" HDD or SSD drives /drawer

230 TBytes\drawer (w/600 GB 10K SAS disks), 154 GB/s BW/drawer, software-controlled RAID, up to 6/rack (replacing server drawers) (up to **1.38 PBytes / rack**)

Servers – 256 Power7 cores / drawer, 1-12 drawers / rack

Compute: 8-core Power7 CPU chip, 3.7 GHz, 12s technology, 32 MB L3 eDRAM/chip, 4-way SMT, 4 FPUs/core, Quad-Chip Module; **>90 TF / rack**

- No accelerators: normal CPU instruction set, robust cache/memory hierarchy
- "Easy" programmability, predictable performance, mature compilers & libraries

Memory: 512 Mbytes/sec per QCM (0.5 Byte/FLOP), **12 Terabytes / rack**

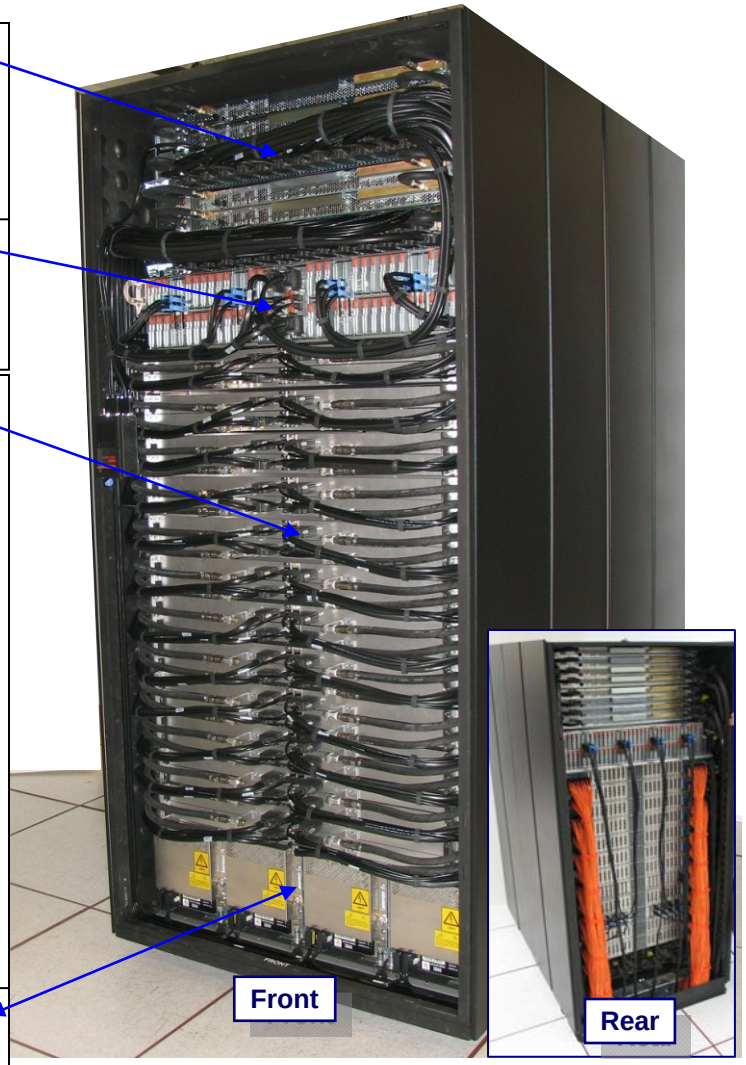
External IO: **16 PCIe Gen2x16 slots / drawer**; SAS storage, 1/10G Enet external

Network: Integrated Hub (HCA/NIC & Switch) per each QCM (8 / drawer), with 54-port switch, including total of 12 Tbits/s (1.1 TByte/s net BW) per Hub:

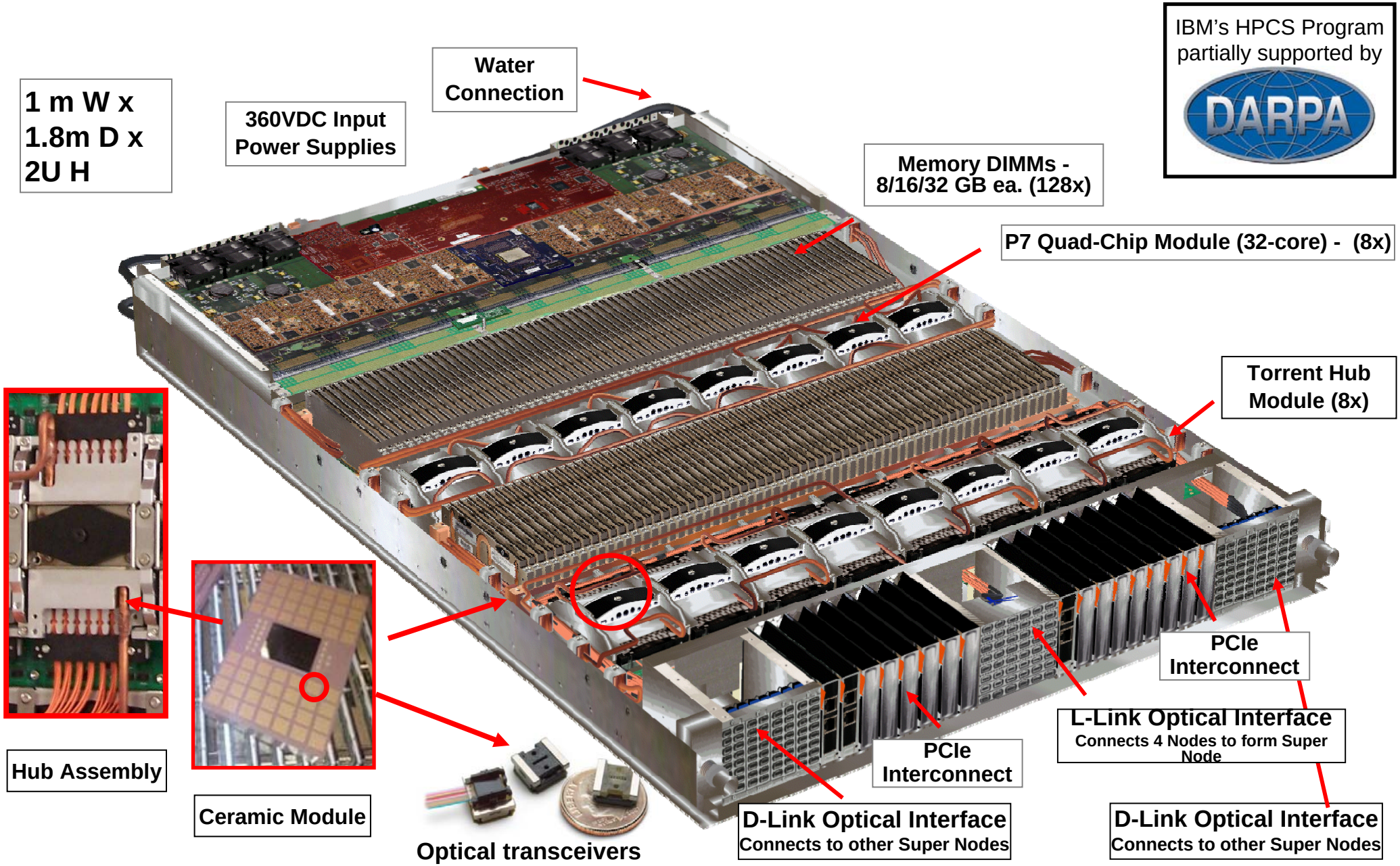
- Host connection: 4 links, (96+96) GB/s aggregate (0.2 Byte/FLOP)
- On-card electrical links: 7 links to other hubs, (168+168) GB/s aggregate
- Local-remote optical links: 24 links to near hubs, (120+120) GB/s aggregate
- Distant optical links: 16 links to far hubs (to 100M), (160+160) GB/s aggregate
- PCI-Express: 2-3 per hub, (16+16) to (20+20) GB/s aggregate

Integrated Cooling – Water pumps and heat exchangers

All heat transferred directly to building chilled water – **no thermal load on room**



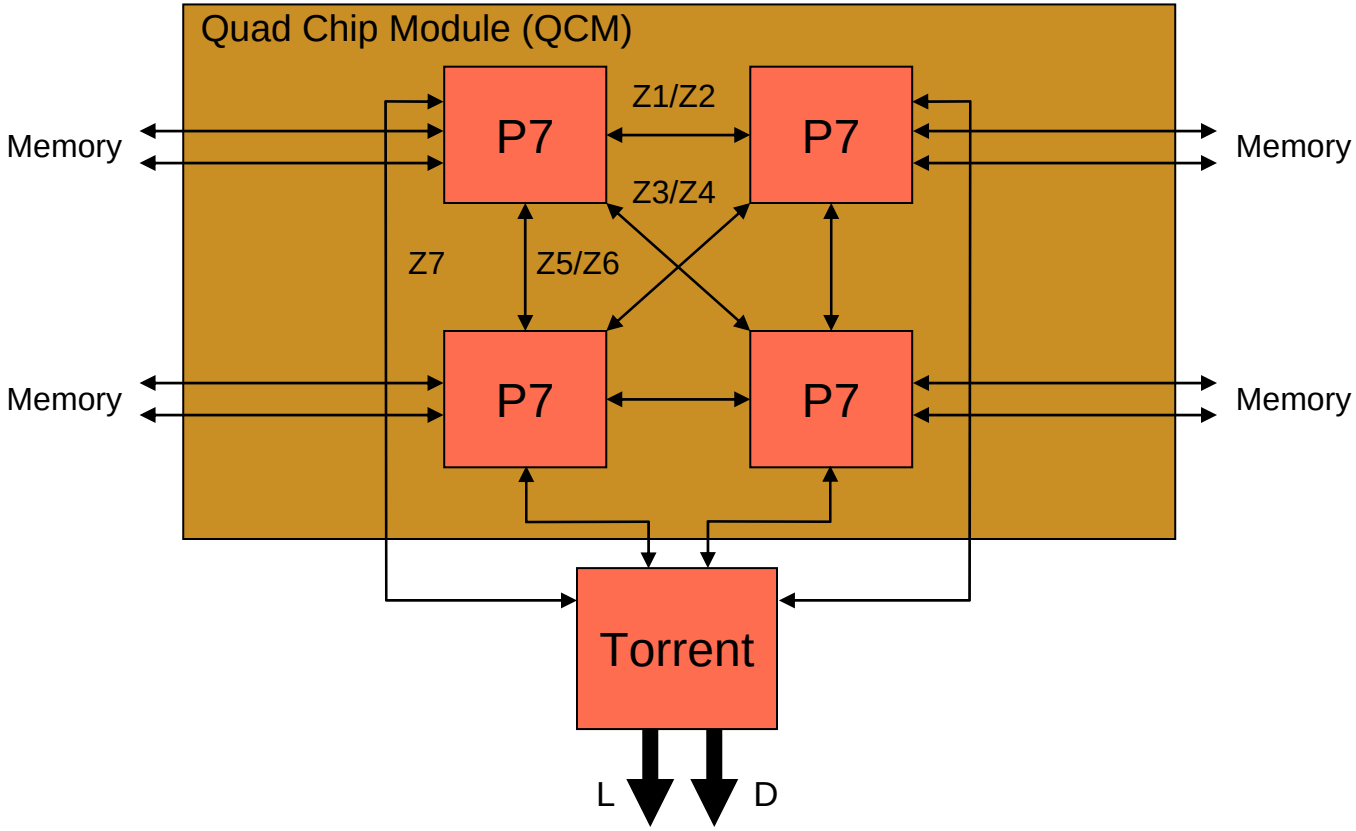
Power775 Node Drawer



IBM's HPCS Program partially supported by DARPA

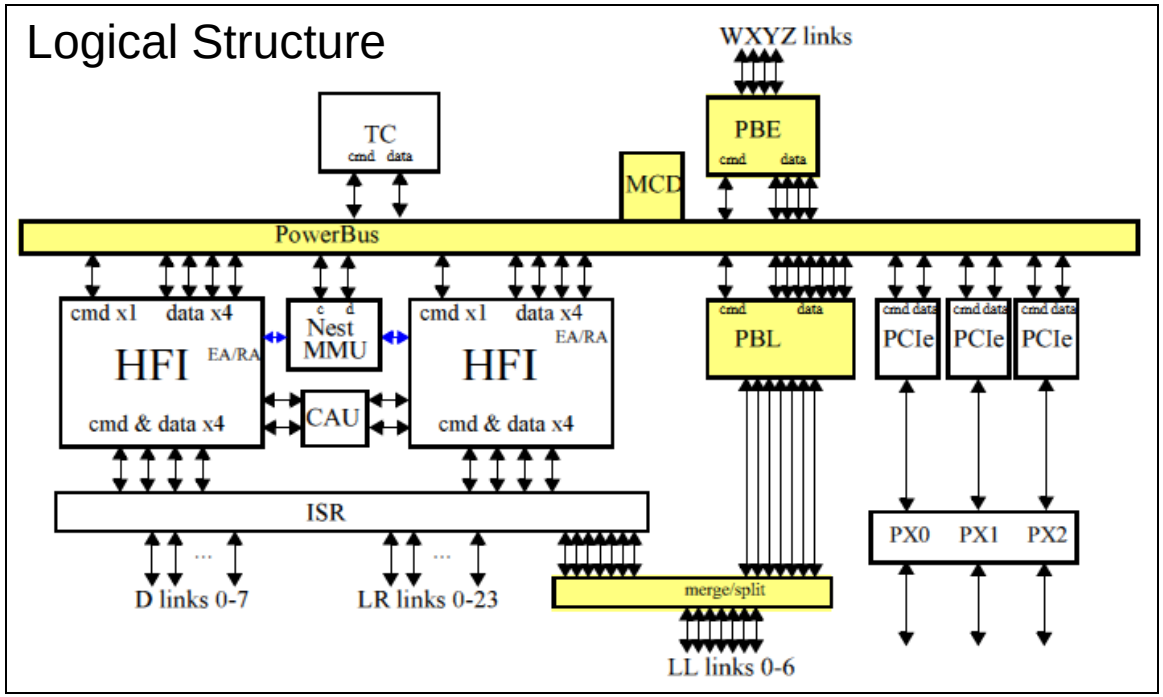
Power775 Node (aka “Octant”)

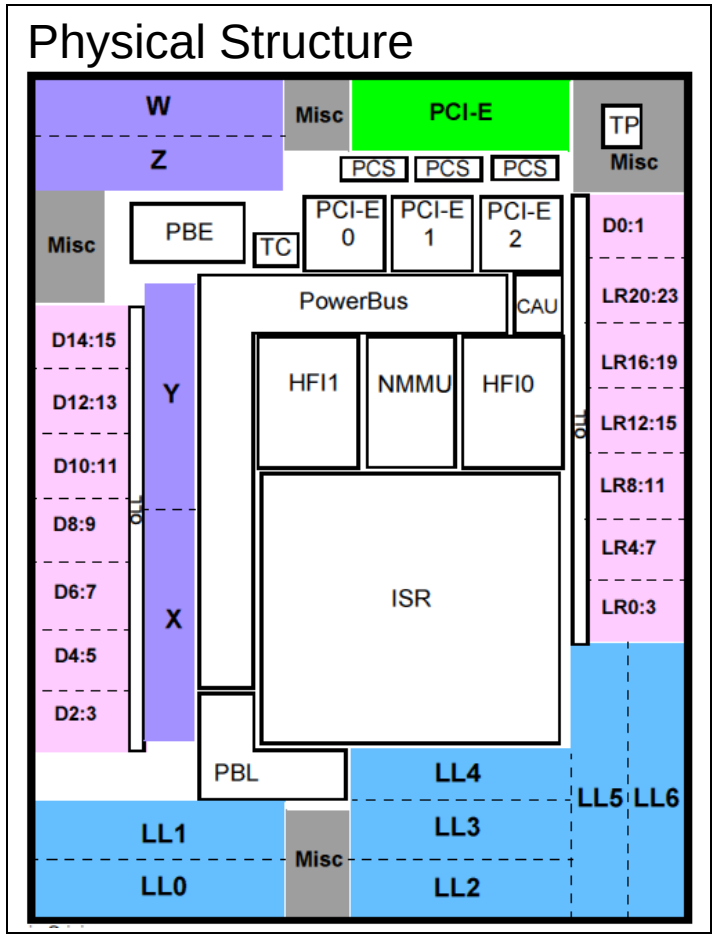
- **4 P7 chips - 32 cores - on a Quad-Chip Module**
- **Torrent Hub chip is “on the processor bus” - direct peer with Power7 chips in a 5-chip all-to-all mesh**



Power775 Torrent Hub Chip

- **5 basic functions in a single chip:**
  - Multi-Octant Coherency -- System Memory Bus + Translation Control + MMU
  - HFI (Host-Fabric Interfaces) with CAU (Collectives Acceleration Unit)
  - ISR (Integrated Switch Router)
  - LL + LR + D-Link Link-Level Interface Logic
  - PCIe bus bridge

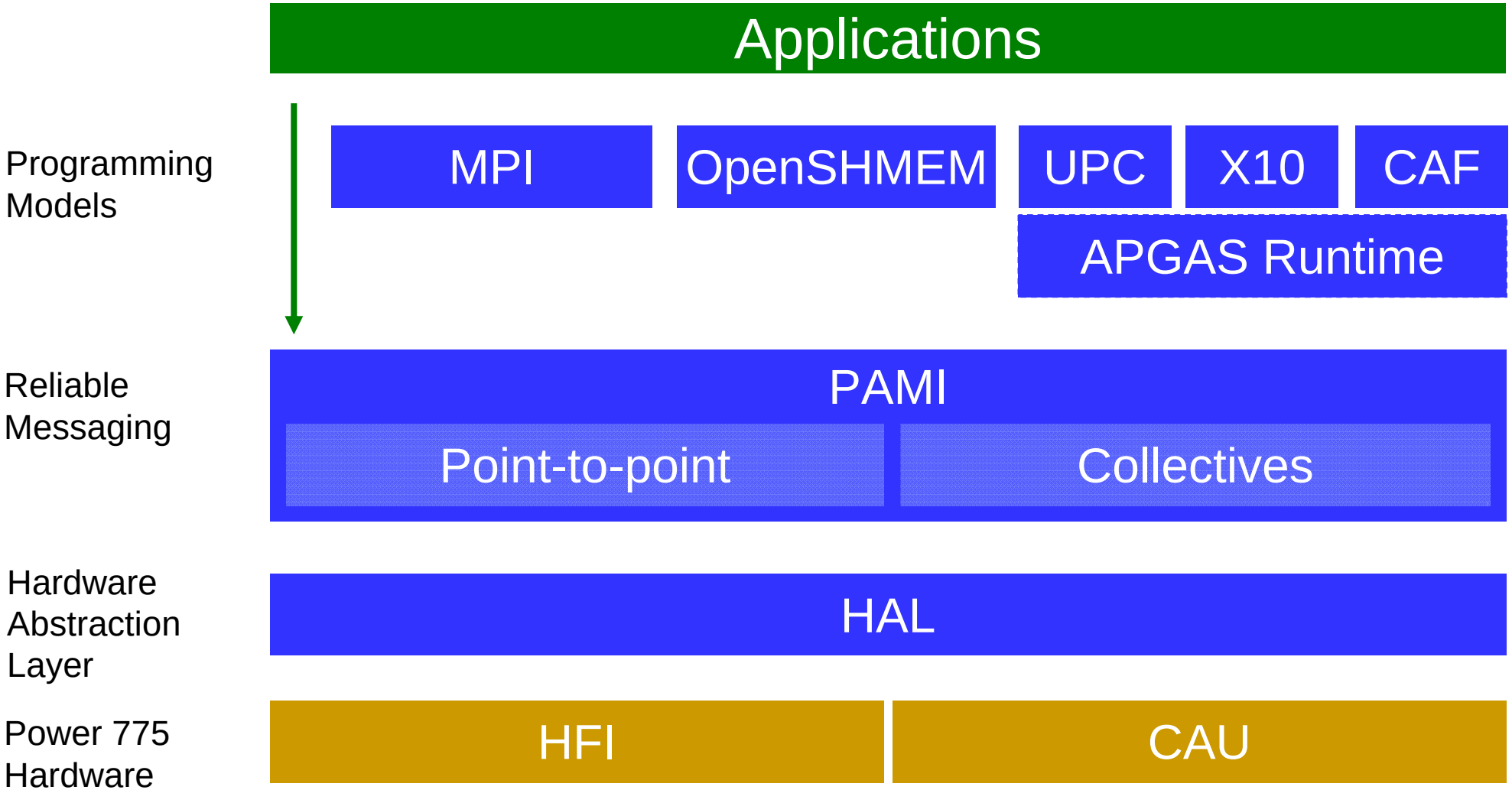




Outline

- Intro to Power 775 (PERCS) hardware and software
- PAMI - Parallel Active Messaging Interface - Overview
- OpenSHMEM over PAMI - Basic Operations
- OpenSHMEM over PAMI on Power 775 - Details
- Performance: OpenSHMEM / PAMI / Power 775

Protocol Stack on Power 775 System



Motivations of PAMI

- **Common Messaging Interface for all IBM HPC Platforms**
  - Improvements on P6/5/4,.. LAPI (Low-Level API) and BlueGene DCMF
  - Extensible to other platforms (InfiniBand,..)
- **Support broad range of programming models**
  - MPI, OpenSHMEM
  - APGAS (Async. Partitioned Global Address Space): UPC, X10, CAF
  - Mixed: (e.g., MPI + APGAS)
  - Direct: Applications can use PAMI directly
- **Reduce dev. cost by sharing components across platforms**
  - Support non-contiguous data types & optimized collectives, on all systems
- **Provide a research platform for messaging innovation**

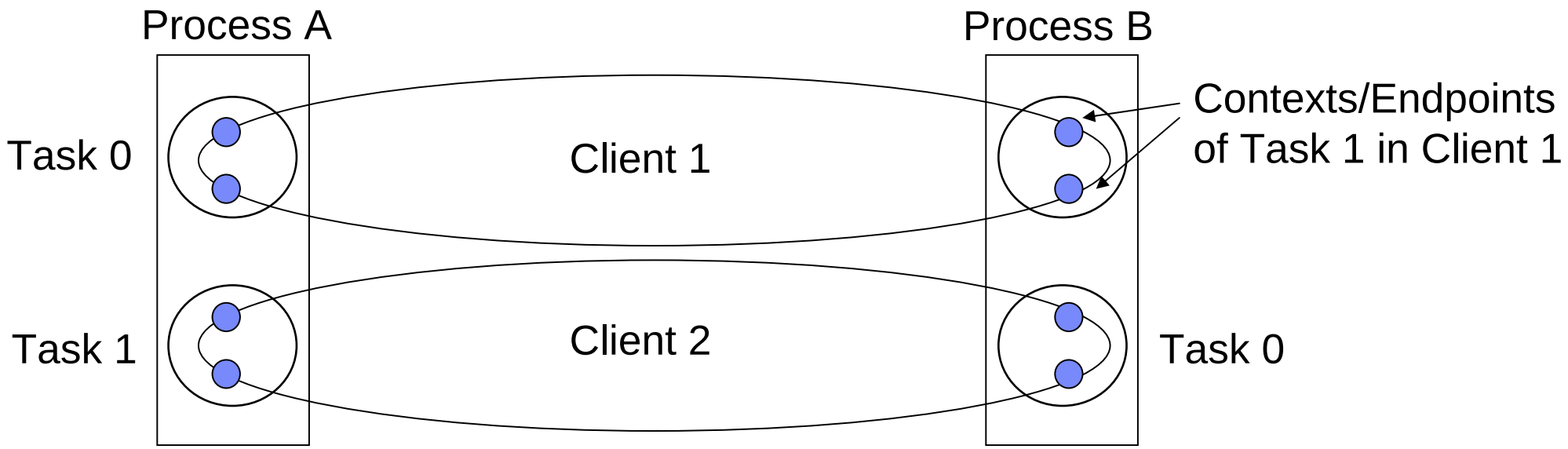
PAMI Design Philosophy

- **Focus on latency and throughput optimizations**
  - Benchmark: “Million messages per second” (or billion)
  - Avoid “features” that hurt performance (counters, out-of-line completion,..)
- **Optimize collectives to exploit hardware acceleration**
- **Application determines policy, for example:**
  - Applying threads to communication contexts
  - Progress model
- **Support current & future HPC libraries & models**
  - MPI 2.x, OpenSHMEM
  - ARMCI, GASNet
  - MPI 3.0
  - PGAS languages: UPC, X10 and CAF
  - Accelerator and Hybrid

PAMI Features

- **Multiple independent communications capability**
  - Allows network topology-aware applications
- **Communication endpoint addressing**
- **Reliable transport**
- **Non-blocking for all calls (higher level does blocking as needed)**
- **Non-contiguous type system and data conversion support**
- **Fence interface enables hardware optimization**
- **Extension framework**
  - Enables access to innovative messaging research
- **Integrated collectives**
  - Collectives optimized for platform and network topology
  - Collective selection capability provided for application
- **Failover and recovery, purge/resume**

PAMI Communication Components



- **Client: An independent network instance**
- **Task: Process identifier within a client**
- **Context: Communication resources within a client that make independent progress**
- **Endpoint: Communication address of a context**
- **Geometry: Group of tasks or endpoints used by collectives**

Point-to-point, Collectives, Synchronization

- **Point-to-point**
  - One-sided data transfer between a pair of local and remote PEs
- **Collectives**
  - Data movement and operation that over a set of involved PEs (active set)
- **Synchronization**
  - Ensure data ordering (fence, quiet)
  - Ensure execution ordering (barrier)

PAMI exploiting of POWER 775 hardware

- **Point-to-Point**
  - Immediate send, User level RDMA, Remote atomics , Send-side driven interrupt
- **Collectives -- BlueGene Framework + PERCS HW**
  - BSR: (Barrier Synchronization Register for on node synchronization)
  - CAU: Collective Acceleration Unit for off node collective traffic

	PAMI Functions	HFI Functions

Active message	`PAMI_Send_immediate`
	`PAMI_Send`
	`PAMI_Send_typed`
		`HFI Immediate Send`
		`HFI RDMA Read`
		`HFI FIFO`

Put	`PAMI_Put (PAMI_Rput)` `PAMI_Put_typed`	`HFI RDMA Write` `HFI FIFO`
Get	`PAMI_Get (PAMI_Rget)` `PAMI_Get_typed`	`HFI RDMA Read` `HFI FIFO`
Remote Atomics	`PAMI_Rmw`	`HFI Atomics`
Progress	`PAMI_Context_advance`	`HFI FIFO Receive`

Active message	`PAMI_Send_immediate`
	`PAMI_Send`
	`PAMI_Send_typed`
		`HFI Immediate Send`
		`HFI RDMA Read`
		`HFI FIFO`

Active Message Sends

- **PAMI_Send_immediate**
  - Fast path for sending small contiguous messages.
  - No event notifications at the sender's side.
  - Returning from function implies that send buffers (i.e. header plus data) are reusable.
- **PAMI_Send**
  - For sending contiguous messages of arbitrary sizes.
  - Event notifications at the sender's side
    - When send buffers are reusable, and/or
    - When the message has been received.
- **PAMI_Send_typed**
  - For sending non-contiguous messages of arbitrary sizes and layouts.
  - A user-defined data function or the default memory copy function to manipulate the data.
  - Event notifications at the sender's side
    - When send buffers are reusable, and/or
    - When the message has been received.
- **Each send must specify a dispatch ID to use at the receive side.**

Dispatch and Active Message Receive

- **PAMI_Dispatch_set defines a local dispatch which has three parts.**
  - An ID to be used in active message send functions
  - A dispatch function to be invoked when the first packet of an active message arrives
  - Hints to define how messages using the dispatch must be handled
- **PAMI queues or retransmits a message with a dispatch ID that's not yet defined.**
- **A dispatch function is invoked with**
  - Entire message header
  - Entire message data if the size is small
  - Message data size
  - Source of the message
- **A dispatch function can**
  - Consume a message immediately when the message data is entirely presented, or,
  - Specify the following receive parameters and let PAMI handle the incoming.
- **Receive parameters of an active message consist of**
  - A receive buffer, either contiguous or non-contiguous, for the incoming message,
  - A data function or the default memory copy function to manipulate the data,
  - An event function to notify that the message has been completely received

Remote Memory Access

- **PAMI_Memregion_create**
  - Prepare a buffer for RDMA operations by creating a memory region for it
- **PAMI_Memregion_destroy**
  - Destroy a previously created memory region
- **Two flavors of RMA operations**
  - Operations that don't require memory registration
    - Work on all interconnects
    - The user doesn't need to keep track of memory registrations
  - Operations that require memory registration (explicit RDMA/user level RDMA)
    - Work on RDMA-capable interconnects and fail otherwise
    - The user needs to keep track of memory registrations

Remote Memory Access – RDMA optional

- **PAMI_Put**
  - Stores data from a contiguous local buffer to a contiguous remote buffer
  - Generates initiator-side event notifications
    - When the local buffer is reusable
    - When the data has been stored into the remote buffer
- **PAMI_Get**
  - Loads data from a contiguous remote buffer to a contiguous local buffer
  - Generates initiator-side event notification
    - When the data has been loaded into the local buffer
- **PAMI_Put_typed and PAMI_Get_Typed**
  - Same as PAMI_Put and PAMI_Get except handling non-contiguous buffers
- **PAMI_Rmw**
  - Atomic operations like “fetch & add” and “compare & swap” with 4 and 8-byte operands
  - Generates initiator-side event notification
    - When the operation has completed and/or the fetched value has been received.
- **All above functions can issue hardware RDMA operations directly if**
  - Both local buffer and remote buffer have been registered for RDMA, and,
  - Hardware RDMA operations don’t require the user to carry RDMA registrations around.

Remote Memory Access – RDMA required

- **PAMI_Rput**
  - Stores data from a contiguous local buffer to a contiguous remote buffer with RDMA write
  - Generates initiator-side event notifications
    - When the local buffer is reusable
    - When the data has been stored into the remote buffer
  - Requires memory regions to be passed in
- **PAMI_Rget**
  - Loads data from a contiguous remote buffer to a contiguous local buffer with RDMA read
  - Generates initiator-side event notification
    - When the data has been loaded into the local buffer
  - Requires memory regions to be passed in
- **PAMI_Rput_typed and PAMI_Rget_typed**
  - Same as PAMI_Rput and PAMI_Rget_typed except handling non-contiguous buffers
- **RDMA-only version of Rmw**
  - Use buffer_registered hint also meet buffer alignment requirement of the hardware

Progress

- **PAMI_Context_advance**
  - Poll on a context until
    - A message-completion event has happened, or
    - A given polling count has been reached.
- **PAMI_Context_advancev**
  - Poll on multiple contexts
- **PAMI_Context_lock, PAMI_Context_trylock and PAMI_Context_unlock**
  - Acquiring a context exclusively
- **PAMI_Context_post**
  - Post a work function to a context without acquiring it first.
  - The work function is invoked when the context is being advanced.

Fence

- **PAMI_Fence_begin and PAMI_Fence_end**
  - Mark the code region where fence operations can be used
  - Potentially avoid fence-related overheads when fence is not use
- **PAMI_Fence_endpoint**
  - Fence on messages to a specified destination
- **PAMI_Fence_all**
  - Fence on messages to all destinations
- **Batch completion**
  - A fence generates event notification when all messages that the fence applies to have completed
- **User-controlled ordering**
  - The user must not issue new messages that have dependency on the fenced messages before the fenced messages complete.

Type system - Defined data types

- **PAMI_Type_create**
  - Create a new type
- **PAMI_Type_add_simple**
  - Add contiguous buffers into a type repeatedly with a stride
- **PAMI_Type_add_typed**
  - Add typed buffers into a type repeatedly with a stride
- **PAMI_Type_complete**
  - Specify the atom size of a type
  - Complete a type and no more modification allowed afterwards
- **PAMI_Type_destroy**
  - Destroy a type
- **A new type can be created by a combination of the following means.**
  - Enumerating contiguous buffers
  - Repeating a contiguous buffer with a stride
  - Enumerating typed buffers
  - Repeating a typed buffer with a stride

Type system (cont.) - Transferring a Type

- **PAMI_Type_serialize**
  - Retrieve the contiguous buffer holding a serialized type
  - A serialize type can be copied or transferred as regular data
- **PAMI_Type_deserialize**
  - Reconstruct a type from a contiguous buffer holding a serialized type

Type system - moving data between Types

- **PAMI_Type_transform_data**
  - Transform data from a typed buffer to another typed buffer in the same address space
  - Source type and destination type can differ
  - Data function can be specified to override the default data copy function
- **Packing or Gathering**
  - Source is non-contiguous and destination is contiguous
- **Unpacking or Scattering**
  - Source is contiguous and destination is non-contiguous

Fault Tolerance

- **PAMI_Purge**
  - Clean up all pending transfers to a destination and stop communicating to it.
- **PAMI_Resume**
  - Resume communication to a previously purged destination.
- **Also used by dynamic tasking**

PAMI Collectives - 1

- **Many reasons to include collectives in PAMI**
  - DCMF (Blue Gene) & LAPI (Power) had limited collectives, which proved useful
  - We have specialized hardware (Power775 CAU, BG Multicast Unit, Tree, etc)
  - We want to expose a variety of algorithms for each collective
  - We can do optimizations, even on non-specialized hardware
    - E.g., PAMI can use topology awareness where higher levels can't
- **PAMI collectives are influenced by many of the ideas from MPI**
  - Active message collectives
  - Non-blocking collective behavior
  - Sub-communicator support
  - Tighter integration with type layouts
  - Generalized math support
- **Middleware collectives can benefit all programming models**
  - MPI: Potentially better hardware primitive support
  - PGAS runtimes (UPC/CAF/,..) use collective operations as compiler target
  - GA/ARMCI have collective operations defined

PAMI Collectives - 2

- **Collectives Operation Support MPI-like operations. Differences include:**
  - Nonblocking
  - Integrated with PAMI Type system
  - Integrated with PAMI Contexts
  - Post collective work to a PAMI context, and allow progress model to advance
- **Active Message Collectives**
  - Non-blocking
  - Register a header handler with a special dispatch set routine
  - Issue the collective from a single node, use progress engine to complete operation
- **No 1-Sided collectives**

- Allreduce
- Allgather
- Allgatherv
- Allgatherv_int
- Alltoall
- Alltoallv
- Alltoallv_int
- Reduce
- Reduce_scatter
- Scan
- Gather
- Gatherv
- Gatherv_int
- Amscatter
- Amgather
- Amreduce
- Ambroadcast
- Broadcast
- Barrier

Outline

- Intro to Power 775 (PERCS) hardware and software
- PAMI - Parallel Active Messaging Interface - Overview
- OpenSHMEM over PAMI - Basic Operations
- OpenSHMEM over PAMI on Power 775 - Details
- Performance: OpenSHMEM / PAMI / Power 775

IBM OpenSHMEM General Information

- **Features of OpenSHMEM**
  - Standard API to improve portability between platforms
  - An API that allows the participating processes (PE) to view a Partitioned Global Address Space (PGAS)
    - Each PE has access to its own private local memory and also a shared memory space
  - One-Sided point-to-point communication - Reducing communication overhead
  - Collective communication
  - Atomic operations
  - Synchronization operations
- **Implement to OpenSHMEM v1.0 final spec (2/1/2012 version)**
- **OpenSHMEM on PERCS**
  - Natural mapping from OpenSHMEM to PAMI
  - Exploiting hardware
    - FIFO send/receive, RDMA read/write, RDMA atomics, Collective acceleration

IBM OpenSHMEM over PAMI

- **Take advantage of new PAMI communication middleware**
  - Most OpenSHMEM functions have natural counterpart in PAMI (minimum overhead)
  - Collective algorithm selection
- **Multi-protocol support**
  - MPI + OpenSHMEM
  - UPC + OpenSHMEM
- **Future Non-blocking OpenSHMEM support**
  - Discussed on OpenSHMEM forum
  - PAMI by design is non-blocking

OpenSHMEM	PAMI Function	Power775/HFI implementation
shmem_put*	PAMI_Put PAMI_Rput	GSM WRITE
shmem_get*	PAMI_Get PAMI_Rget	GSM READ
shmem_iput* (strided put)	PAMI_Put_typed	SEND/RECEIVE
shmem_iget* (strided get)	PAMI_Get_typed	SEND/RECEIVE
shmem_barrier	PAMI_Collective (barrier)	BSR, CAU
shmem_swap	PAMI_Rmw	GSM ATOMIC CAS
shmem_*_to_all	PAMI_Collective (allreduce)	CAU reduce/broadcast
shmem_broadcast*	PAMI_Collective (broadcast)	CAU broadcast
shmem_collect	PAMI_Collective (gather, gatherv)	SEND/RECEIVE

IBM OpenSHMEM over PAMI - 1

openshmem	PAMI Functions	Comments
Shmem_put*	PAMI_Put	• Create endpoint • Prepare parameters • Issue PAMI_Put • Wait for completion
Shmem_get*	PAMI_Get	• Create endpoint • Prepare parameters • Issue PAMI_Get • Wait for completion
Shmem_*_swap / Shmem_*_cswap Shmem_*_add / shmem_*_fadd shmem_*_inc / shmem_*_finc	PAMI_Rmw	• Create endpoint • Prepare parameters • Issue PAMI_Rmw • Wait for completion
Shmem_iput*	PAMI_Put_typed	• Create endpoint • Create simple types for local & remote • Prepare parameters • Issue PAMI_Put_typed • Wait for completion
Shmem_iget*	PAMI_Get_typed	• Create endpoint • Create simple types for local & remote • Prepare parameters • Issue PAMI_Get_typed • Wait for completion

IBM OpenSHMEM over PAMI - 2

openshmem	PAMI Functions	Comments
Shmem_barrier*	PAMI_Fence_endpoint PAMI_Collective (xfer_barrier)	• Issue PAMI_Fence_endpoint and wait for completion • Lookup collective algorithm • Prepare parameters • Issue PAMI_Collective • Wait for completion
Shmem_*_to_all Shmem_broadcast*	PAMI_Collective (xfer_allreduce/xfer_broadcast)	• Lookup collective algorithm • Prepare parameters • Issue PAMI_Collective • Wait for completion
Shmem_fcollect*	PAMI_Collective (xfer_allgather)	• Lookup collective algorithm • Prepare parameters • Issue PAMI_Collective(xfer_allgather) • Wait for completion
Shmem_collect*	PAMI_Collective (xfer_allgather)	• Do shmem_fcollect gather ***nlong*** from PEs • Lookup collective algorithm • Prepare parameters; construct ***disps*** array • Issue PAMI_Collective(xfer_allgatherv) • Wait for completion

IBM OpenSHMEM over PAMI - 3

openshmem	PAMI Functions	Comments
Shmem_wait	PAMI_Context_advance	• Loop on PAMI_Context_advance until condition being satisfied
Shmem_*_lock	PAMI_Send (send-side-driven interrupt) PAMI_Put	• Create endpoint • Prepare parameters (w/ remote_async_progress) • PAMI_Send (request msg) • PAMI_Put (reply)

Synchronization - SHMEM over PAMI

- **shmem_fence**
  - Ensures the order of a series of PUTs from local PE to a specific PE
  - PAMI_Fence_begin is called at initialization time
  - PMAI_Fence_end is called at the end of job
  - Implement on top of PAMI_Fence_endpoint
- **shmem_quiet**
  - Ensures the order of a series of PUTs from local PE to all PEs
  - PAMI_Fence_begin and PAMI_Fence_end same as above
  - Implement on top of PAMI_Fence_all
- **Barrier**
  - A collective routine that no PE can leave prior all PEs (of the active set) enter it
  - PAMI_Collective with PAMI_XFER_BARRIER

Other features

- **Error handling and reporting**
  - SHMEM_ERROR_REPORT
    - Used to change the terminating behavior of the application when API level error happens.
    - Valid values: no, print, assert, pause (default is no)
  - SHMEM_ASYNC_ERROR
    - Used to change the terminating behavior of the application when asynchronous hardware error happens.
    - Valid values: print, assert, pause (default is print)
- **Multiple protocol support**
  - PAMI is the common low level message interface for IBM MPICH2, UPC, and OpenSHMEM. As a result, user can run a program that using mixed programming model. For example MPI+OpenSHMEM

Outline

- Intro to Power 775 (PERCS) hardware and software
- PAMI - Parallel Active Messaging Interface - Overview
- OpenSHMEM over PAMI - Basic Operations
- OpenSHMEM over PAMI on Power 775 - Details
- Performance: OpenSHMEM / PAMI / Power 775

IBM’s OpenSHMEM on Power775

- **IBM's OpenSHMEM implementation is based on the final OpenSHMEM v1.0 specification announced on 2/1/2012 located on http://www.openshmem.org/**
- **Our header file is equivalent to the specification, except:**
  - No SMA_* environment variables are supported.
    - SMA_SYMMETRIC_SIZE has a corresponding implementation SHMEM_SYMMETRIC_HEAP_SIZE (default size = 256MB)
    - SMA_DEBUG has a corresponding implementation SHMEM_ERROR_REPORT (default = no, valid values: "yes", "no", "pause")
- **Only 64-bit C/C++ supported. No Fortran at this time.**
- **Single thread only, SPMD only**
- **In case of many distinct collective active sets, some active sets may use less efficient implementations**
    - Special hardware resources, such as CAU and BSR, have fixed scalability

Exploitations of POWER 775 features

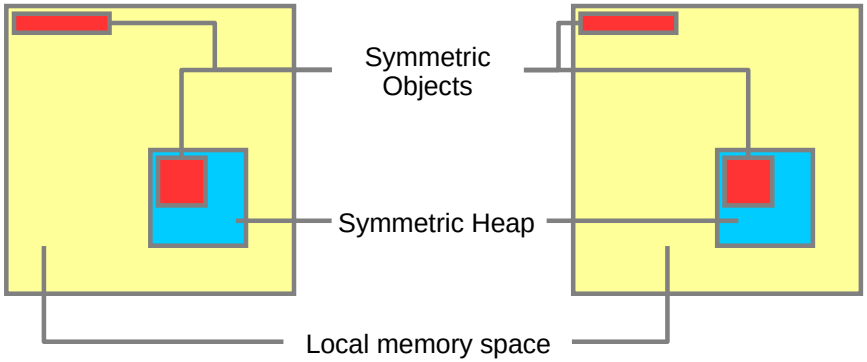
- **HFI and P7 exploitation**
  - Collective acceleration: CAU, BSR
  - CAU: Collective Acceleration Unit, a new hardware feature on Torrent to accelerate multicast and reduce of small data size.
    - Multi-cast: broadcast across one of 32 predefined trees
    - Reduce: Ops: (NOP, SUM, MIN, MAX, AND, OR, XOR). Operands: (32-bit or 64-bit, signed and unsigned, fixed-point and floating-point). BSR: Barrier Synchronization Register, for fast on-node synchronization.
- **Global Shared Memory RDMA w/o memory handle exchanges**
  - all RDMA operations take virtual addresses, so there is no need for a user to carry memory registration handles like with InfiniBand™.
- **Send Side Initiated Interrupt**
  - A special packet that enables receive side interrupt once the packet arrives
- **Hardware atomic operations**

No-op and Unsupported functions

Name	Description	Action
shmem_clear_cache_inv	Turn off the automatic cache coherency mode	no-op
shmem_set_cache_inv	Turn on the automatic cache coherency mode	no-op
shmem_set_cache_line_inv	Turn on the automatic cache coherency mode for the specified target only	no-op
shmem_udcflush	Make the entire user data cache coherent	no-op
shmem_udcflush_line	Make the specified target coherent	no-op
shmem_ptr	Return a pointer of a remote object	Return NULL
Note:	No-op functions are provided for compatibility only	

Initialization – Symmetric heap and objects

- **Symmetric heap**
  - A memory block each PE allocates upfront during the initialization routine and used for shmalloc, shmemalign, or shrealloc functions.



- **Symmetric Objects**
  - Symmetric data objects, also known as remotely accessible objects, are arrays or variables that exist with the same size, type, and effective (virtual) address on all PEs. In the interface definitions for IBM OpenSHMEM communication functions, one or more of the parameters are typically required to be symmetric (remotely accessible). The following data objects are remotely accessible:
    - The variables allocated by shmalloc, shmemalign, or shrealloc functions
    - Non-stack C and C++ variables, if SPMD (same executable file for all PEs)
    - Fortran data objects in common blocks or with SAVE attribute
    - Fortran arrays allocated with shpalloc function

Initialization – Symmetric Heap

- **shmalloc, shmemalign, shfree, shrealloc**
  - Operate on symmetric heap through our internal memory management framework
  - Barrier at the end of each function to ensure synchronization
- **Initialization**
  - Use mmap to reserve a chunk of memory of size SHMEM_SYMMETRIC_HEAP_SIZE
  - Attach the memory segment to the same address on all PEs
  - Register the whole symmetric heap with PAMI_Memregion_create
  - One time cost; no need to register/unregister during the run
- **start_pes(0) has to be called before other OpenSHMEM routines**
  - Symmetric heap is pre-allocated at job start. Be careful not to over commit memory usage on a node
  - SHMEM_SYMMETRIC_HEAP_SIZE used to describe needed space

Initialization (pseudo code)

- **Inside start_pes()**
  - Initialize symmetric heap
  - Initialize PAMI Client (with name “shmem”)
  - Query for task_size (total number of PEs)
  - Query for my_task (my PE number)
  - Initialize PAMI Context
  - Obtain World Geometry
  - Query world geometry for barrier algorithms
  - Pre-register symmetric heap
  - Register other symmetric objects
  - Issue Barrier operation on the context
  - Advance on the contexts, until message callback is complete

Point-to-point Communication

- **Point-to-point**
  - One-sided model: does not require remote PE to participate
    - Put: Copy data from local buffer to remote PE
    - Get: Copy data from remote PE to local buffer
  - Contiguous data transfer is built on top of PAMI_Put and PAMI_Get functions
    - Take advantage HFI GSM feature (user level RDMA)
      - Local and remote buffers are pre-registered
      - Pure one-sided operation
  - Non-contiguous data transfer is built on PAMI_Put_typed and PAMI_Get_typed
    - Future work could use RDMA for optimization

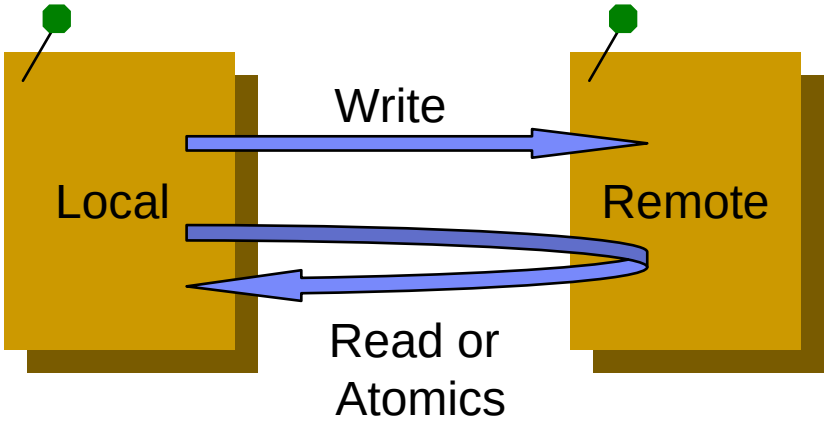
2-sided communication with acknowledgement

1-sided communication without acknowledgement

http://www.openshmem.org

RDMA Exploitation

- **Register shared memory segment to hardware**
- **Allocate memory from shared memory symmetrically**
- **Issue PAMI_Rput, PAMI_Rget and PAMI_Rmw with "buffer_registered" hint.**
- **Wait on completion for blocking semantics**



Collectives Communications

- **What is Active Set?**
  - Group of PEs that involved in the execution of a collective operation
  - An active set mapps to a PAMI Geometry object
    - It is created once on demand and cached for later use
- **Implemented as a wrapper around PAMI_Collective() call**
  - Many algorithms can be applied for a given operation
  - Collective selection helps to choose a good one
- **Supported operations**
  - Broadcast
    - Copy data from root PE to PEs in the active set
  - Collect
    - Concatenate data buffers from source array over PEs in the active set to target array
  - Reduction
    - Performance associative binary operations over PEs in the active set

Atomics

- **Perform atomic operations on symmetric objects**
  - Supported operations:
    - Swap,
    - compare_and_swap,
    - add,
    - fetch_and_add,
    - inc,
    - fetch_and_inc
  - "These routines guarantee that accesses by OpenSHMEM's atomic operations will be exclusive, but do not guarantee exclusivity in combination with other routines, either inside OpenSHMEM's or outside." -- *OpenSHMEM spec v1.0 final, OpenSHMEM.org*
  - Implemented on top of *PAMI_Rmw* (a wrapper)
  - Utilize HFI Remote atomic support (RDMA version of atomic operation)

CAU Exploitation - Example

- **Geometry cache**
  - (*PE_start, logPE_stride, PE_size*) → PAMI Geometry
- **Allocate CAU resource to the job**
- **shmem_double_sum_to_all**
  - Look up geometry cache; build a new geometry if missed.
  - Issue PAMI_XFER_ALLREDUCE, specifying hardware acceleration
  - Wait for completion

Outline

- Intro to Power 775 (PERCS) hardware and software
- PAMI - Parallel Active Messaging Interface - Overview
- OpenSHMEM over PAMI - Basic Operations
- OpenSHMEM over PAMI on Power 775 - Details
- Performance: OpenSHMEM / PAMI / Power 775

Performance (MPP Benchmarks)

- **Tests Leverage P775 features: GSM, FIFO, RDMA, BSR, CAU**
  - Compiler options for the benchmarks: **`-O -I. -DIBM_SHMEM`**
  - Environment Variables used for some tuning
- **Benchmark results are from DARPA Demonstration System ("Hurcules"), in Aug. 2012, for PERCS/HPCS Milestone**
- **Total size: 47,296 tasks (= 47,296 cores)**
  - 48 Supernodes (1478 octants); 32 tasks (PEs) per node/octant

Benchmark Results: barrier

Test case name	**barrier**
Test case parameters	100000
Test output	base seed is 1344518898 performed 100000 barriers in 4.899e+00 cpu secs (4.899e-05 /barrier) performed 100000 barriers in 4.870e+00 wall secs (4.870e-05 /barrier)

- **Time for a barrier across 47,296 tasks: <49 μsec**

Benchmark Results: bcast

Test case name	bcast
Test case parameters	0 100000 1000
Test output	base seed is 0 msize = 97.66 KiB randomizing root PEs (-1) performed 1000 8 B bcasts in 2.836e-01 wall secs (2.836e-04 /bcast) 8 B bcasts yield 27.55 KiB/sec (/PE), 1.24 GiB/sec (aggregate) performed 1000 16 B bcasts in 2.216e-01 wall secs (2.216e-04 /bcast) 16 B bcasts yield 70.50 KiB/sec (/PE), 3.18 GiB/sec (aggregate) performed 1000 32 B bcasts in 2.200e-01 wall secs (2.200e-04 /bcast) 32 B bcasts yield 142.05 KiB/sec (/PE), 6.41 GiB/sec (aggregate) performed 1000 64 B bcasts in 2.083e-01 wall secs (2.083e-04 /bcast) 64 B bcasts yield 300.02 KiB/sec (/PE), 13.53 GiB/sec (aggregate) performed 1000 128 B bcasts in 1.985e-01 wall secs (1.985e-04 /bcast) 128 B bcasts yield 629.81 KiB/sec (/PE), 28.41 GiB/sec (aggregate) performed 1000 256 B bcasts in 2.024e-01 wall secs (2.024e-04 /bcast) 256 B bcasts yield 1.21 MiB/sec (/PE), 55.70 GiB/sec (aggregate) performed 1000 512 B bcasts in 1.975e-01 wall secs (1.975e-04 /bcast) 512 B bcasts yield 2.47 MiB/sec (/PE), 114.20 GiB/sec (aggregate) performed 1000 1 KiB bcasts in 2.045e-01 wall secs (2.045e-04 /bcast) 1 KiB bcasts yield 4.77 MiB/sec (/PE), 220.52 GiB/sec (aggregate) performed 1000 2 KiB bcasts in 2.371e-01 wall secs (2.371e-04 /bcast) 2 KiB bcasts yield 8.24 MiB/sec (/PE), 380.44 GiB/sec (aggregate) performed 1000 4 KiB bcasts in 2.645e-01 wall secs (2.645e-04 /bcast) 4 KiB bcasts yield 14.77 MiB/sec (/PE), 682.08 GiB/sec (aggregate) performed 1000 8 KiB bcasts in 3.175e-01 wall secs (3.175e-04 /bcast) 8 KiB bcasts yield 24.60 MiB/sec (/PE), 1.11 TiB/sec (aggregate) performed 1000 16 KiB bcasts in 4.334e-01 wall secs (4.334e-04 /bcast) 16 KiB bcasts yield 36.05 MiB/sec (/PE), 1.63 TiB/sec (aggregate) performed 1000 32 KiB bcasts in 4.741e-01 wall secs (4.741e-04 /bcast) 32 KiB bcasts yield 65.91 MiB/sec (/PE), 2.97 TiB/sec (aggregate) performed 1000 64 KiB bcasts in 7.124e-01 wall secs (7.124e-04 /bcast) 64 KiB bcasts yield 87.73 MiB/sec (/PE), 3.96 TiB/sec (aggregate) performed 1000 98 KiB bcasts in 9.826e-01 wall secs (9.826e-04 /bcast) 98 KiB bcasts yield 97.06 MiB/sec (/PE), 4.38 TiB/sec (aggregate) cksum is 7b33a1ffdf2265c0

- **Max Aggregate bcast BW to 47296-1 tasks: 4.38 TiB/s, with 98 KiB bcast (97 MiB/s/PE)**

Benchmark Results: reduce

Test case name	**reduce**
Test case parameters	100000
Test output	base seed is 1344519108 performed 100000 reductions in 2.310e+01 cpu secs (2.310e-04 /reduce) performed 100000 reductions in 2.312e+01 wall secs (2.312e-04 /reduce) ans = 5847984864

- **Time for a reduction across 47,296 tasks: <232 μsec**

Benchmark Results: all2all

Test case name	all2all
Test case parameters	1 21504 2048 4
Test output	base seed is 1 tsize = 2048MB msize = 21504B cksum is f79a0a05c64cad6a wall reports 1.681 secs cpus report 1.699 secs 570.786 MB/sec with 21504 bytes transfers cksum is f200d900ae956399 wall reports 1.646 secs cpus report 1.700 secs 570.553 MB/sec with 21504 bytes transfers cksum is 88c6ec03bd7bdb40 wall reports 1.627 secs cpus report 1.600 secs 595.988 MB/sec with 21504 bytes transfers cksum is 95078c9af7cdc260 wall reports 1.619 secs cpus report 1.600 secs 599.195 MB/sec with 21504 bytes transfers

- **all2all: 570-599 MB/s with 21,504 bytes transfers**

Benchmark Results: pingpong

Test case name	pingpong
Test case parameters	0 8192 16
Test output	base seed is 0 pair 0:29024 ->32520: 1 iters in 1.609e-05 wall secs (1.609e-05 /iter) pair 0:29024 ->32520: 2 iters in 6.080e-06 wall secs (3.040e-06 /iter) pair 0:29024 ->32520: 4 iters in 1.144e-05 wall secs (2.861e-06 /iter) pair 0:29024 ->32520: 8 iters in 2.003e-05 wall secs (2.503e-06 /iter) pair 0:29024 ->32520: 16 iters in 3.910e-05 wall secs (2.444e-06 /iter) pair 1:9083 ->38337: 1 iters in 1.597e-05 wall secs (1.597e-05 /iter) pair 1:9083 ->38337: 2 iters in 5.960e-06 wall secs (2.980e-06 /iter) pair 1:9083 ->38337: 4 iters in 1.049e-05 wall secs (2.623e-06 /iter) pair 1:9083 ->38337: 8 iters in 1.955e-05 wall secs (2.444e-06 /iter) pair 1:9083 ->38337: 16 iters in 3.755e-05 wall secs (2.347e-06 /iter) …

- **Latency per 16 iterations: 2.373 μs / iter. Avg.**
  3.844 μs / iter. Max.
  2.064 μs / iter. Min.

Summary

- PAMI provides unified middleware layer for SHMEM, UPC, MPI, etc., on Power775 (..& BlueGene/Q, InfiniBand, …)
- SHMEM over PAMI works efficiently and well.
- Power775 special functions (CAU, BSR) and overall performance provide good OpenSHMEM performance to large scale.

Backups

Point-to-point function implementation example (e.g. shmem_double_get)

```
void shmem_double_get(double* target, const double* source, size_t len, int pe)
{
    volatile bool     finish = false;
    pami_get_simple_t get;
    pami_send_hint_t  hints = null_send_hint;

    hints.buffer_registered = PAMI_HINT_ENABLE; // whole symmetric heap is pre-registered

    get.rma.dest    = PAMI_Endpoint_create(pe);    // only support one endpoint per context
    get.rma.hints   = hints;
    get.rma.bytes   = len * sizeof(double);
    get.rma.cookie  = &finish;
    get.rma.done_fn = get_completion;        // to set finish flag to true
    get.addr.local  = target;
    get.addr.remote = source;

    PAMI_Get(context, &get);

    while (!finish)
        PAMI_Context_advance(context, POLL_CNT);
}
```

Make it block

Collective function implementation example (e.g. shmem_double_sum_to_all)

```
void shmem_double_sum_to_all(double *target, double *source, int nreduce,
     int PE_start, int logPE_stride, int PE_size, double *pWrk, long *pSync)
{
    volatile bool    finish   = false;
    pami_algorithm_t algo;
    pami_geometry_t  geometry = _geometry_cache.Get(PE_start, logPE_stride, PE_size);
    PAMI_Geometry_algorithms_query ( … ); // Get the always working or the best algorithm
    pami_xfer_t allreduce;
    allreduce.cb_done = cb_done;
    allreduce.cookie  = &finish;
    allreduce.algorithm                      = algo;
    allreduce.cmd.xfer_allreduce.sndbuf      = source;
    allreduce.cmd.xfer_allreduce.stype       = PAMI_TYPE_DOUBLE;
    allreduce.cmd.xfer_allreduce.stypecount  = nreduce;
    allreduce.cmd.xfer_allreduce.recvbuf     = target;
    allreduce.cmd.xfer_allreduce.rtype       = PAMI_TYPE_BYTE;
    allreduce.cmd.xfer_allreduce.rtypecount  = nreduce * sizeof(double);
    allreduce.cmd.xfer_allreduce.op          = PAMI_DATA_SUM;
    PAMI_Collective(context, &allreduce);
    while (!finish)
        PAMI_Context_advance(context, POLL_CNT);
}
```

Get cached geometry and algorithm

Make it block

Geometry cache (active set cache)

```
pami_geometry_t GeometryCache::Get (int pe_start, int pe_logstride, int pe_size) {
    volatile bool finish = false;
    pami_geometry_t geometry;
    hash_key_t key = GetKey(pe_start, pe_logstride, pe_size);
    geometry = _cache.find(key);
    if (geometry != PAMI_NULL_GEOMETRY) return geometry;
    // … create task list …
    ……
    PAMI_Geometry_create_tasklist ( client, NULL, 0, &geometry, PAMI_NULL_GEOMETRY,
                        _cache.size()+1, task_list, task_cnt, context, done, &finish);
    while(!finish)
        PAMI_Context_advance(context, POLL_CNT);
    _cache.add(key, geometry);// geometry on every participating PE is created
    return geometry;
}
```