ORNL SummitDev Training

Interconnect

MPI

GPU Direct

Outline

- Part 1 – IB Interconnect
  - Overview
  - Component technologies
  - RDMA
- Part 2 – GPU direct
  - GPUDirect capabilities
  - IB interface
  - Managed Memory vs Device memory
- Part 3 – Spectrum MPI use
  - Spectrum MPI overview/GPU support
  - Usage hints for MPI programs
  - GPU recommendations

Part 1 – IB interconnect

High Bandwidth low latency switched fabric interconnect

Technology	HW latency	1x effective bandwidth	4x effective bandwidth
FDR	~0.7 usec	1.7 GB/sec	6.8 GB/sec
EDR	~0.5 usec	3.0 GB/sec	12.1 GB/sec
HDR	~0.5 usec	6.1 GB/sec	24.2 GB/sec

Open hardware standard

Open source software stack, OpenFabrics Enterprise Distribution (OFED)
– Mellanox maintains its own version of OFED containing to be incorporated changes to the standard

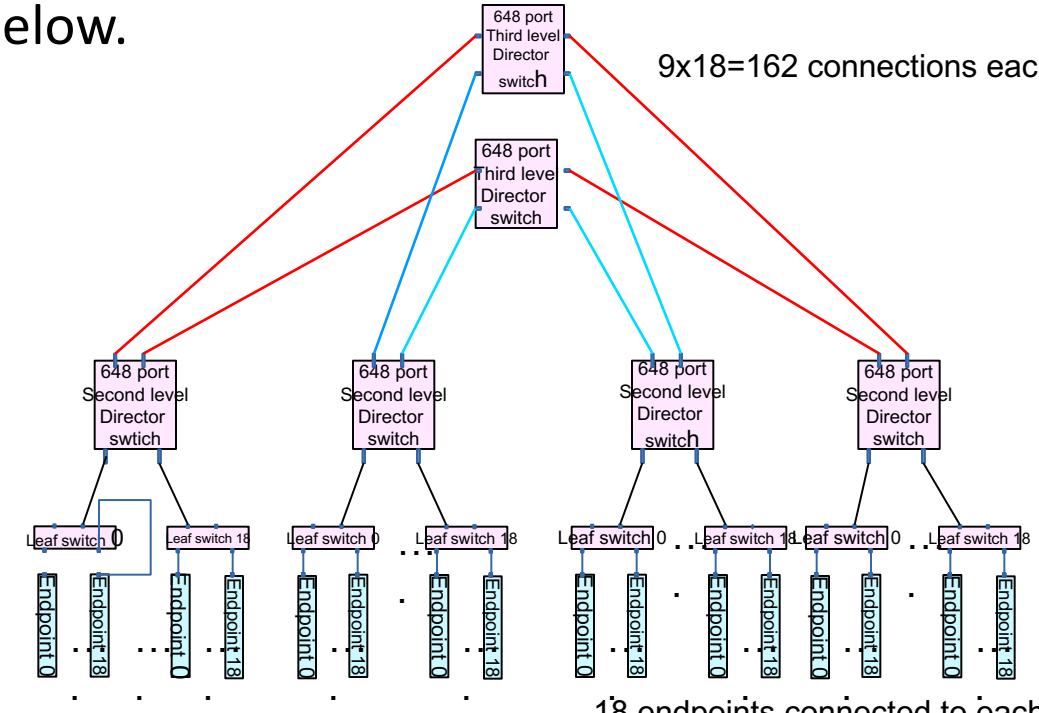
Lossless network with link level retry

IB components

- HCA – Host Channel Adapter
  - Connection from Host CPU to network
  - Has network unique Logical ID (LID)
  - Supports multiple independent network interfaces (Qpairs or QPs)
  - Supports multiple message types
    - RDMA (READ and WRITE)
    - SEND/RECEIVE
    - ATOMIC operations (Compare and Swap/Fetch and Add)
  - Supports multiple connection types
    - UD (Unreliable Datagram – SEND/RECEIVE only)
    - RC (Reliable Connected – All message types)
    - DCT (Dynamic Connected Transport – Eventually all message types)
  - CORAL system will have either EDR or HDR
  - Packets routed from source (LID,QP) to destination (LID,QP)

IB components – continued

- IB Switches
  - Routes packets from HCA to switch, HCA to HCA, switch to HCA, or switch to switch
  - Multiple topologies supported. CORAL will use FAT tree as below.



18 endpoints connected to each leaf switch
18(leaf switches)x18(connections/leaf)=324
connections from leaf switches to director switch

IB components – continued

- IB subnet manager
  - Assigns network unique LID to each HCA
  - Assigns to routes between LIDS
  - Monitors Network for network issues
  - Defines SL to VL translations
- For Mellanox the subnet manager is UFM
  - Can control UFM from a GUI
    - http://UFM_SERVER_NAME/ufmui
      - Network configuration
      - Event monitoring
  - Event log can be found in /opt/ufm/files/log/event.log

IB components – continued

- IB Verbs – standardized interface to HCA, software support with OFED
  - Components
    - Protection Domain
    - Send Queue
      - Work Queue Request Entry put on the send queue
        - Work request ID
        - Destination (UD and DCT)
        - Work Request type (ATOMIC, RDMA READ/WRITE, or SEND/Receive)
        - Tag matching WR under discussion for OFED
        - Memory Key (LKEY/RKEY) and virtual address for memory accesses
    - Completion Queue – notification of completed work requests
  - Network requests initiated by posting a request to send queue
  - IB complexity hidden in MPI – for OpenMPI in PAMI/OpenIB btl/MXM/UCX

Virtual memory registration

- User calls ibv_reg_mr with a pointer to memory, length, protection domain pointer, and access bits.
  - ibv_reg_mr calls kernel mr registration function
    - Pins the memory
    - Creates DMA bus mapping for region
    - Associates lkey and rkey with bus mapping
    - Provides HCA with DMA mapping lkey and rkey
    - Returns a ibv_mr struct containing
      - Virtual address
      - Length
      - Lkey and rkey
- The lkey/address can the be used for local access and the rkey/address can be used for remote access
- This flow is similar to the GPU direct case – will discuss later

On Demand Paging

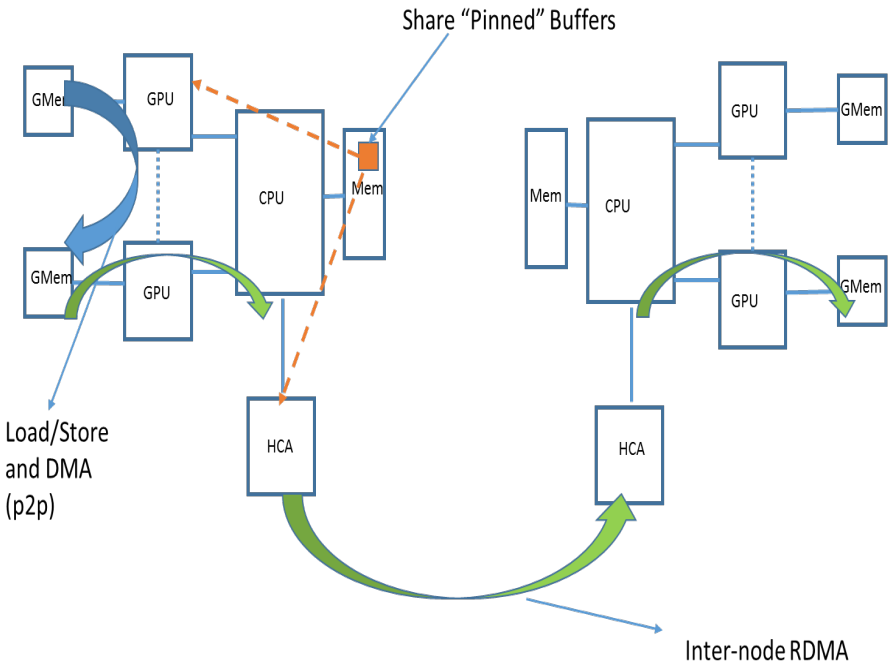
- A special ODP flag can be used when registering memory
  - Will not pin the memory
    - Translations are done on the fly and the HCA maintains a cache of translations
    - Final CORAL system will have enhancements
- Two types of ODP
  - Explicit – a standard memory region is created with unpinned pages
  - Implicit – The whole of the user address space can be mapped
    - Support for Atomic/RDMA operations for this will complete in 2017
- Allows for better use of system memory
- Will be important to support GPUDirect on final CORAL system with managed memory

Part 2 – GPUDirect

High Bandwidth low latency switched fabric interconnect

GPU Direct Family

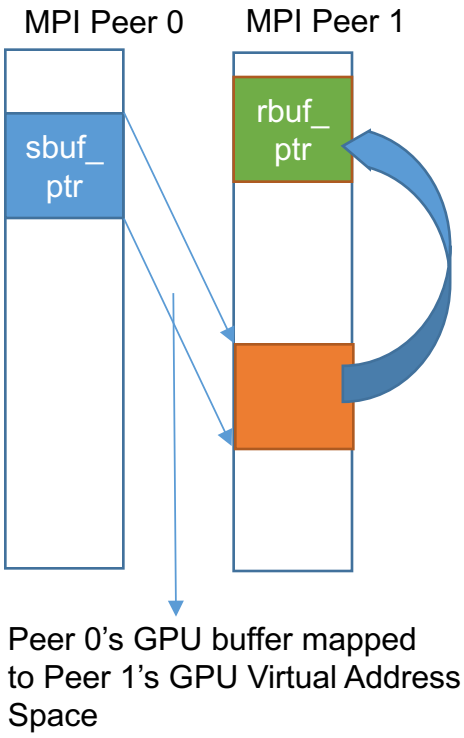
- Features to support efficient *inter-node* and *intra-node* GPU-GPU communication
- Accelerated communication via sharing "Pinned buffers" with IB's HCA (*inter-node*)
- Direct RDMA (*inter-node)*
  - Ability to perform send/recv, RDMA operations between GPUs via IB
  - RDMA allows HCA DMA engines to saturate IB link b/w
  - By-pass CPU memory for reducing latency
  - GPU Direct RDMA across NVLink2←→PHB←→PCIe←→IB.
- Async (*inter- or intra-node)*
  - GPU can initiate and wait for network operation from CUDA streams
  - Remove synchronization between GPU and CPU
- Peer-to-Peer Transfers between GPUs (*intra-node)*
  - Use GPU DMA engines to saturate the link b/w across GPUs
  - Within a socket via PCI-E and NVLINK1
  - Across sockets with NVLINK2

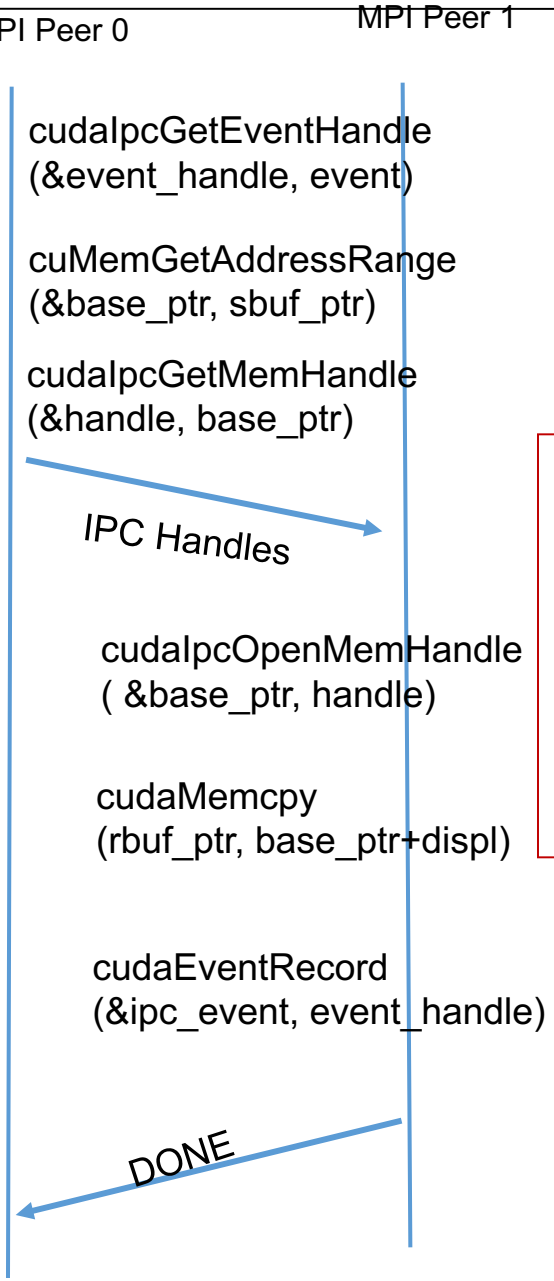


GPUDirect RDMA

- NVIDIA provides a kernel extension that provides calls to return a DMA bus mapping for GPU device memory
- When the user creates a IB memory region, it signals to the device driver that the address is for GPU device memory
- The IB device driver calls the NVIDIA provided routine to obtain the DMA mapping
  - After the DMA mapping is obtained, the MR setup continues normally
- The IB client program uses the returned ibv_mr structure as if it were pointing to host memory

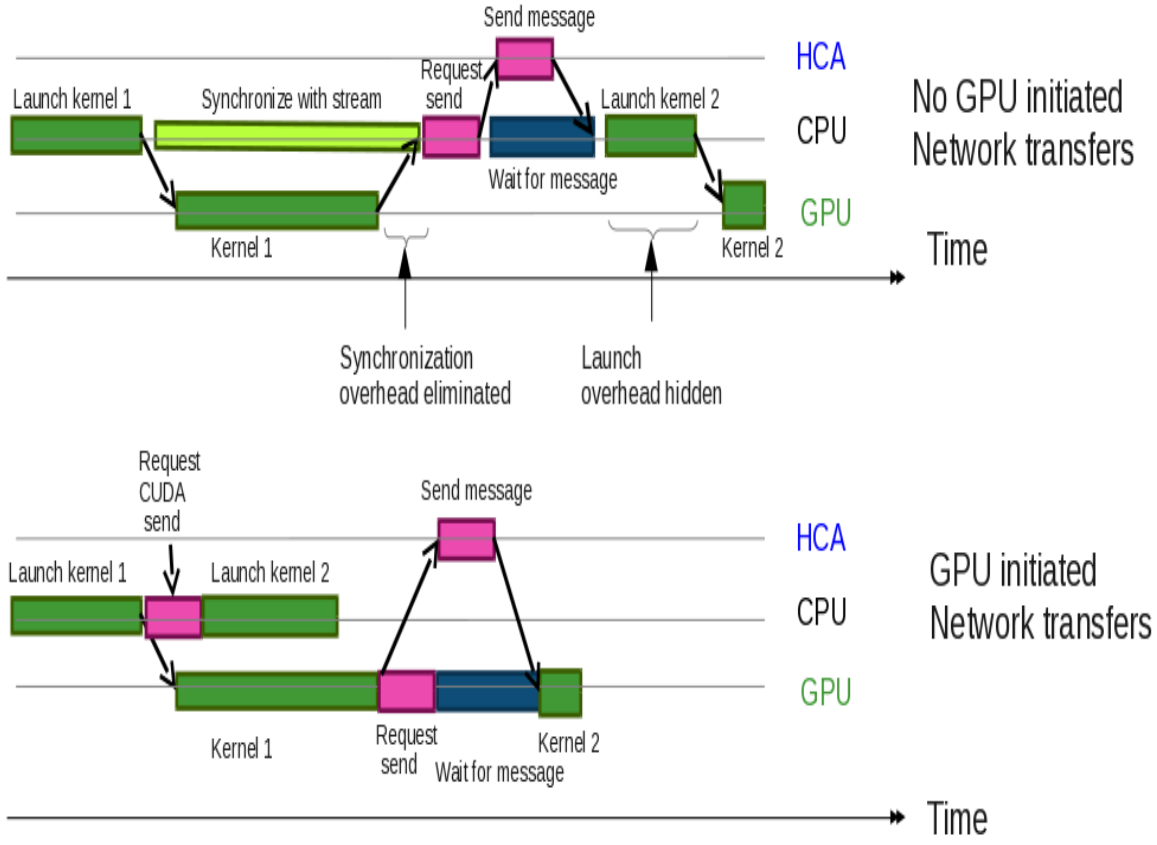
Cuda IPC – for GPUDirect P2P support





- Method of accessing the peer's buffer via Handles
- Useful for point-to-point as well as collective operations
- Getting and Opening handles costly
- Cache them

GPU Async

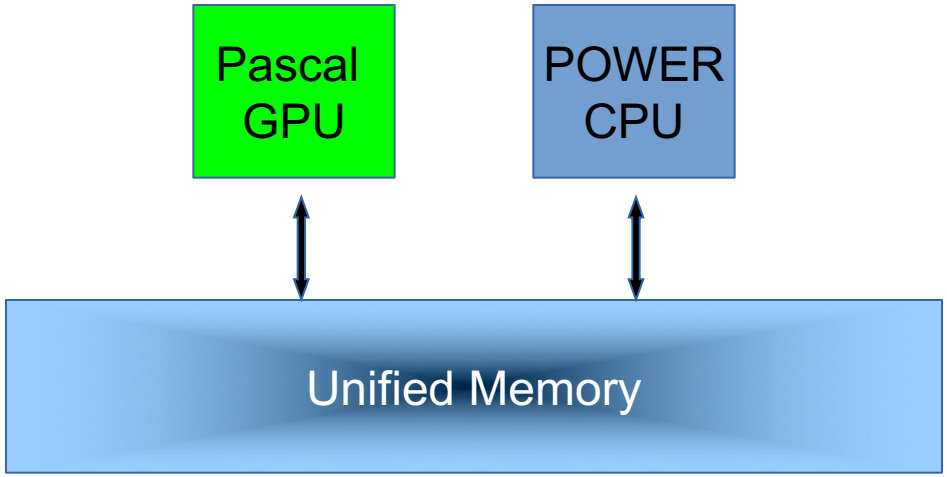

Send message
HCA
Request send
Launch kernel 1
Synchronize with stream
Launch kernel 2
CPU
Wait for message
GPU
Kernel 1
Kernel 2
Time
Synchronization overhead eliminated
Launch overhead hidden
No GPU initiated Network transfers
Request CUDA send
Send message
HCA
Launch kernel 1
Launch kernel 2
CPU
GPU
Kernel 1
Request send
Wait for message
Kernel 2
Time
GPU initiated Network transfers

Requirement for GPUDirect support

- Accessed GPU memory must be GPU Device memory, not a pointer to unified memory
- For GPUDirect RDMA
  - GPU and HCA both on PCIe buses on the same PCIe Domain
    - For POWER8, the data path for the Pascal GPU is over NVLINK1 (there is a control PCIe connection)
    - For POWER8, each PCIe slot is on a different PCIe Domain
      - GPUDirect RDMA will only work on POWER8 if both the HCA and GPU are on PCIe cards in an external PCIe expansion chassis.
      - We internally have done tests this way, but not intended for external use
- Async (*inter- or intra-node)*
  - Same requirements a GPUDirect RDMAU
- Peer-to-Peer Transfers between GPUs (*intra-node)*
  - GPU and HCA on same PCIe domain
  - Within a socket via PCIe and NVLINK1
- Some of these limitations will be relaxed in the future

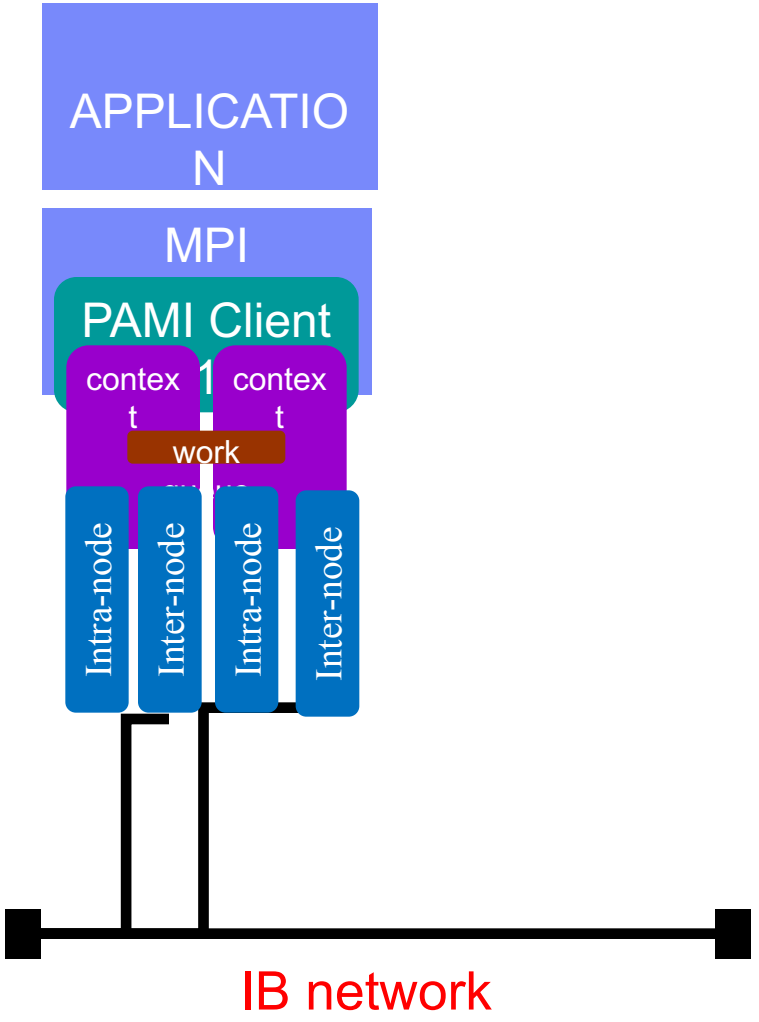
Unified Memory

- Available since CUDA 6 with enhanced support for GP100 and CUDA 8
- Allocate memory and return a pointer that can be dereferenced by either the GPU or CPU
  - Actual pages can be either on the GPU or in host memory
  - For GP100/POWER8 GPU only has load/store access to GPU memory and CPU only has load/store access to host memory
    - Drive moves data as needed to provide transparent access
  - In the future, we intend to enhance these capabilities



Part 3 – Spectrum MPI

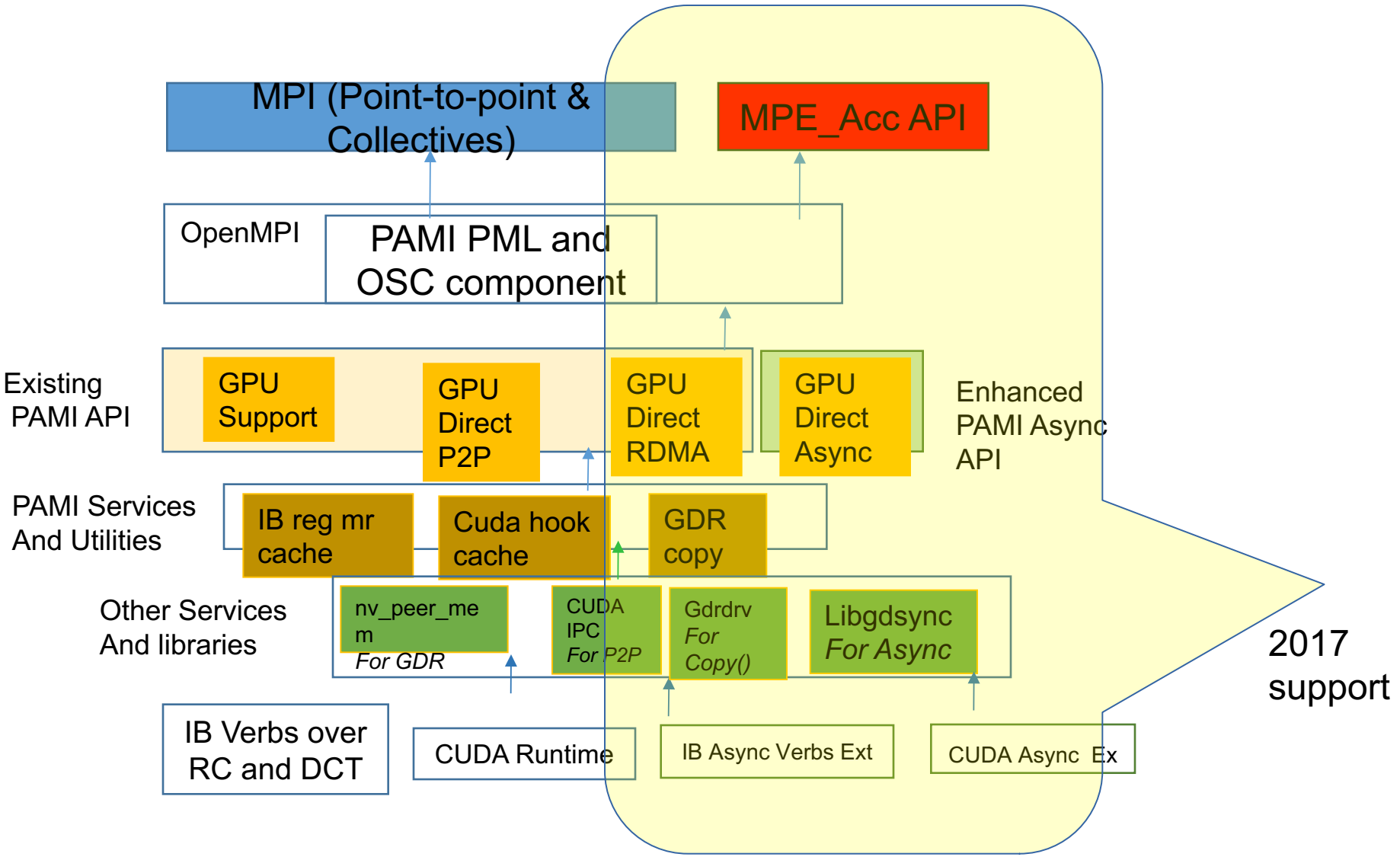
- PAMI
  - Contexts: Independent network end-points (addressable)
  - Protocols: variations of basic eager, rendezvous
  - In-line progress or spawn separate threads
- PAMI API
  - Non-blocking API, Active Message style semantics with completions via asynchronous callback handlers
  - Point-to-point: PAMI_Send_imm, PAMI_Send
  - Remote Memory Access: Various flavors of PAMI_Put/PAMI_Get operations and memory region management
  - Collectives: PAMI_Bcast, PAMI_Allreduce, PAMI_Barrier etc.
  - Context specific: independently progress a context / lock a context/ “hand-off ” messaging functions to a context etc.



Spectrum MPI Stack

Power Systems

IBM



Cuda hook caching for: a) GPU addresses/Contexts and b) GDR copy handles

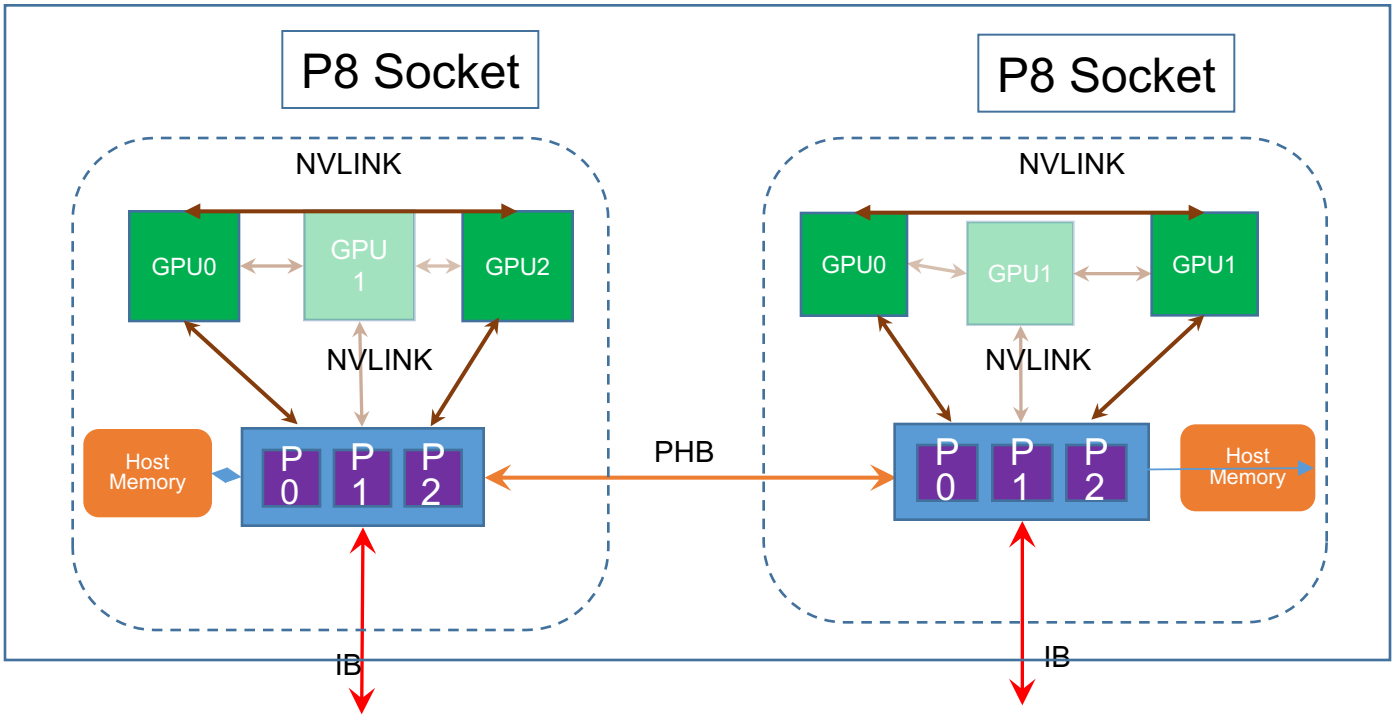
PAMI Caching services

- CudaHook
  - "Address Cache" to store all the GPU buffer pointers, ranges
    - Interception library with callbacks whenever cuda driver calls are invoked
    - Cache hit when the buffer is in the range stored in the cache
  - "GDR Cache" stores the GDR Copy handles when opened for first time
- "IPC Cache" stores the IPC handles
  - when they are first opened using get (sender side cache)
  - Also stores the mapped IPC handles from open (receiver side cache)
  - Cache entries are purged when the associated GPU buffer is deallocated
- Registration Cache
- Separate Cache to store GPU buffer registration entries with IB HCA
- Registration will be enhanced in the future

Protocol implementations for GPU_SUPPORT

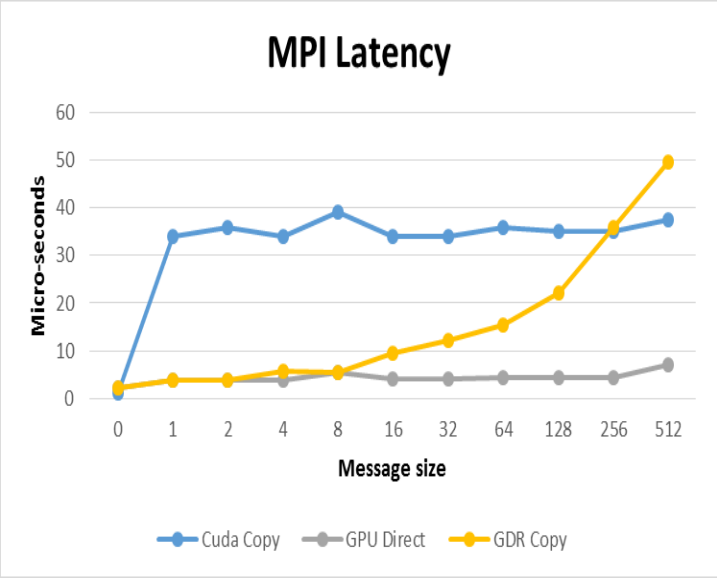
- PAMI Send Immediate & PAMI Send APIs
  - Use 2-Copy Eager protocols
  - cudaMemcpy into Cuda and IB registered pre-allocated buffers
  - Transports: Shared memory (intra-node) and IB (inter-node)
- PAMI Get
  - Pipelined Rndv
  - Protocol Pipelining overlaps cuda memcpys in the node with network transfers
  - Credit control by keeping a constant buffer pool (Memory scalability)
  - The number of elements posted on the recvQ is the total number of credits a process has
  - Current size is 32 with the buffer being 1MB
  - Can use both RC and DCT
- Optimizations
  - Latency: Use gdrcopy instead of cudaMemcpy
    - Effectively uses CPU to load/store from GPU memory to system memory
    - Map the GPU addresses by BAR and MMIO mappings
    - ~ 2 usec from 20-25 usec
  - Bandwidth:
    - All pipeline units are concurrently progressed
    - Near saturation of network b/w

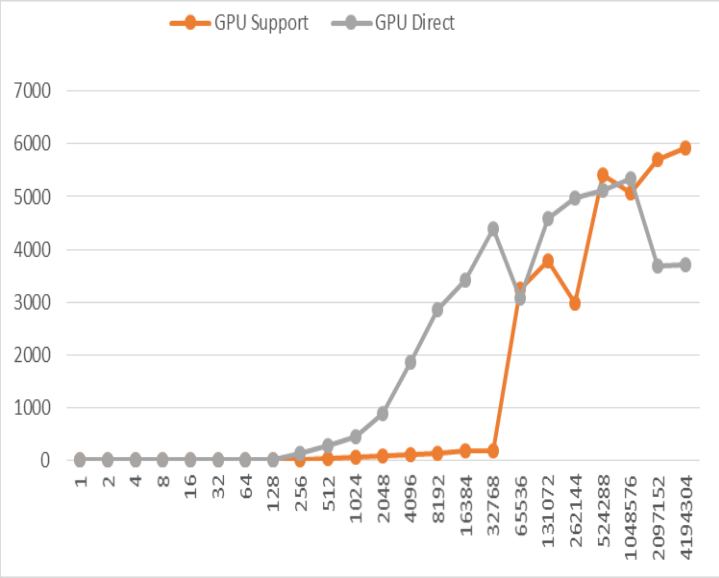
Coral Early Access (EA) Node



- GPUs per socket for ORNL(3), LLNL (2)
- Topology of NVLINK: Cliques (nice property)
- Each GPU is connected to others and the host
- Crucial for collectives (broadcast, reduce)
- Saturate IB network b/w

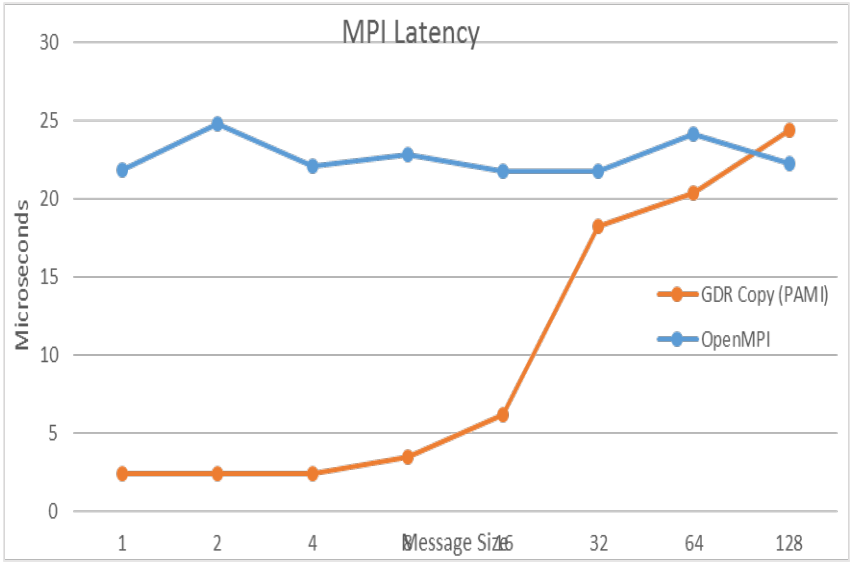
Firestones + K40 + Connect IB + External PCI Switch

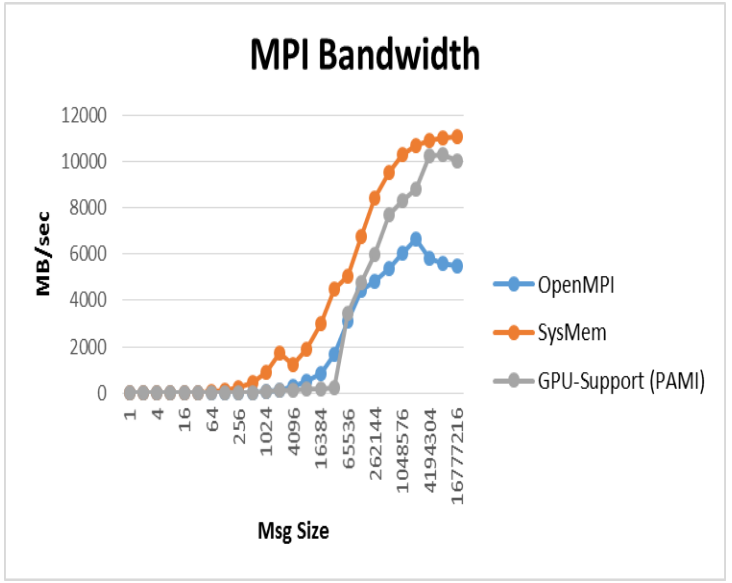




- Both GPU Direct and GDR Copy give latencies close to 3.79 usec
- GDR Copy: 2-Copy Eager, gdr copy used on both sides (read slower than v
- GPU Direct: 1-Copy Eager, gdr copy used on receiver (good for small to me
- GPU Support: Pipelined Rndv good for long messages, coming close to Ne
- Better than GPU Direct RDMA (True also for CORAL Whitherspoon?)
- GPU Direct: Z-Copy Rndv good for medium messages

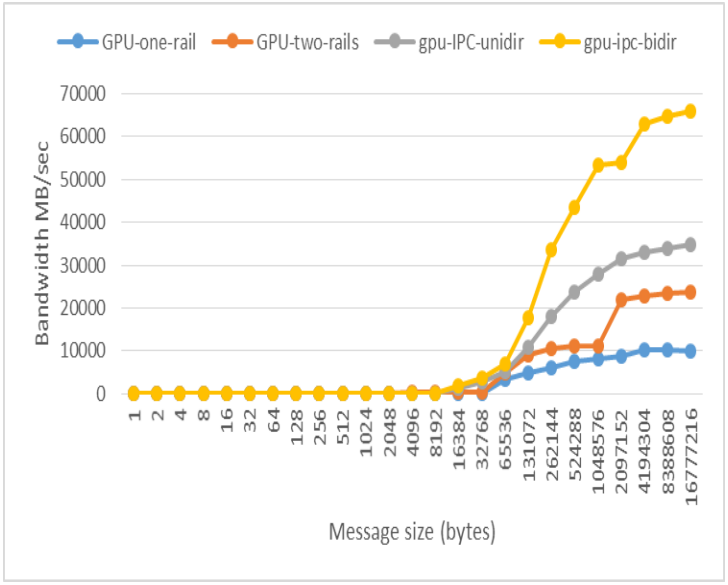
Firestone + ConnectX-4 + K80





- PAMI PML latency of 2.14 usec using 2-Copy Eager with gdrcopy optimization
- GPU Support good for long messages, coming close to SysMem bandwidth

Garrison/Pascal + ConnectX-4



- We get about 34.6 GB/sec B/W over NVLINK1 unidirectional (osu-bw)
- 65 GB/sec bidirectional (osu-bibw)
- Able to stripe across multiple network links (rail)
- 10.2 GB/sec with one lane and 23 GB/sec with two rails

Spectrum MPI usage

- Follows most of the OpenMPI usage
  - In the future we will support job steps with slightly different syntax
- For GPU support:
  - Add -gpu flag in the mpirun command
  - In the future we will be adding programmatic support for GPUDirect Async, but this is not available in the EA system
  - GPUDirect RDMA/P2P will be used when supported
  - Allows GPU device memory pointers to be passed into SMPI
- Other flags of interest
  - -hcoll – use HCOLL collectives
  - -aff – for predefined affinity options

Predefined affinity options

-aff=[option,option,…]

Enables affinity, with any of the following options:

v / vv

Displays output in verbose mode.

cycle:unit

Interleaves the binding over the specified element. The values that can be specified for unit are hwthread, core, socket (the default), numa, or board.

bandwidth | default

Interleaves sockets but reorders them.

latency

Pack.

width:unit

Binds each rank to an element of the size that is specified by unit. The values that can be specified for unit are hwthread, core, socket (the default), numa, or board.

Displaying affinity options

Options for showing the affinity mappings

- *-report-bindings option*
  - Output display similar to:

```
[hostA:ppid] MCW rank 0 bound to socket 0[core 0[hwt 0-1]]:
    [BB/../../../../../../..][../../../../../../../..]
```

- -display-devel-map option
  - Output display similar to:

```
Mapper requested: NULL Last mapper: round_robin Mapping policy: BYCORE
Ranking policy: CORE Binding policy: CORE:IF-SUPPORTED Cpu set: NULL
PPR: NULL Cpus-per-rank: 1
Num new daemons: 0
New daemon starting vpid INVALID
Num nodes: 2
Data for node: hostA
State: 3
Daemon: [[11988,0],1]
Daemon launched: True
Num slots: 4
Slots in use: 4 Oversubscribed: FALSE
Num slots allocated: 4 Max slots: 0
Num procs: 4
Next node_rank: 4
Data for proc: [[11988,1],0]
Pid: 0 Local rank: 0 Node rank: 0
App rank: 0
State: INITIALIZED
App_context: 0
Locale: [BB/../../../../../../..][../../../../../../../..]
Binding: [BB/../../../../../../..][../../../../../../../..]
```

Collective communication enhancements

IBM provides its own collective library

- Incorporates algorithms from Platform MPI and PEMPICH

This library usually, but not always beats other collective libraries

- We provide options to
  - » Select a collective library
  - » Selectively use other libraries for specific collectives
  - » Selectively choose the algorithm

Collective component default priorities

95 : ibm (libcoll)

90 : hcoll (Mellanox)

80 : fca (Mellanox - replaced by hcoll)

78 : cuda (disabled for our builds)

75 : self (loopback)

40 : inter (intercommunicator collectives)

30 : tuned

10 : libnbc (nonblocking collectives)

10 : basic (blocking, neighborhood collectives)

0 : sm (shared memory - seen as unstable)

Simple profiling support

Compile program with -g -Wl,--hash-style=sysv -emit-stub-syms and **xlc**

$ mpicc -O3 -g -Wl,--hash-style=sysv -emit-stub-syms -c scattermodule.c

$ mpicc -O3 -g -Wl,--hash-style=sysv -emit-stub-syms -c reducemodule.c

…

$ mpicc -Wl,--hash-style=sysv -o tstmod ${OBJS}

Instrument the code

$ hpctInst -dhpm -dmpi -dopm ./tstmod

Add instrumentation for hardware counters

Add instrumentation for MPI

Add instrumentation for OpenMP

Run:

$ HPM_EVENT_SET=49 [hpcrun] mpirun -np 4 ./tstmod.inst

Events found with the commands

$. /opt/ibmhpc/ppedev.hpct/env_sh

$ hpccount -l

Example profiling output

hpct_0_0.hpm.tstmod.txt:

```
########  Resource Usage Statistics  ########
Total amount of time in user mode          : 1.593604 seconds
Total amount of time in system mode        : 1.593604 seconds
Number of page faults without I/O activity   : 2562
Number of page faults with I/O activity      : 2
Number of voluntary context switches         : 1432
Number of involuntary context switches       : 22
....
 PM_DATA_FROM_L2 (The processor's data cache was reloaded from local core's L2 due to a demand load) :          2479
 PM_DATA_FROM_L2MISS (Demand LD - L2 Miss (not L2 hit))                                               :          1571
 PM_DATA_FROM_L3MISS (Demand LD - L3 Miss (not L2 hit and not L3 hit))                                :          81
 PM_DATA_FROM_L3 (The processor's data cache was reloaded from local core's L3 due to a demand load) :          1490
 PM_RUN_INST_CMPL (Run instructions)                                                                  :        437359
 PM_RUN_CYC (Run cycles)                                                                              :       1029751

 Utilization rate                    :        23.970 %
 Instructions per run cycle          :         0.425
 Total Loads from local L2           :         0.002 M
 Local L2 load traffic               :         0.303 MB
...
```

hpct_0_0.mpi.txt

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	3	0.0	0.000
MPI_Comm_rank	3	0.0	0.000
MPI_Send	36566	19872.6	0.059
MPI_Recv	36569	19871.0	0.088
MPI_Bcast	27	6.1	0.000
MPI_Barrier	171	0.0	0.000
MPI_Gather	48	8.0	0.000
MPI_Allreduce	73	6.6	0.000

CUDA/MPI programming recommendations

Feature	POWER8/GP100	Final CORAL system
Unified memory With CUDAMemAdvise	Currently testing SMPI	Good performance
CUDA aware MPI (pass GPU pointers)	Yes	Yes
GPUDirect RDMA	No	Yes
GPUDirect P2P	Yes (GPU device memory only)	Yes (GPU Device memory only)
GPUDirect Async	No (API prototype will be available 2017)	Yes