Hewlett Packard
Enterprise

HPE CRAY MPI – SPOCK WORKSHOP

Noah Reddell (on behalf of Krishna Kandalla and MPT Team)

May 20, 2021

AGENDA

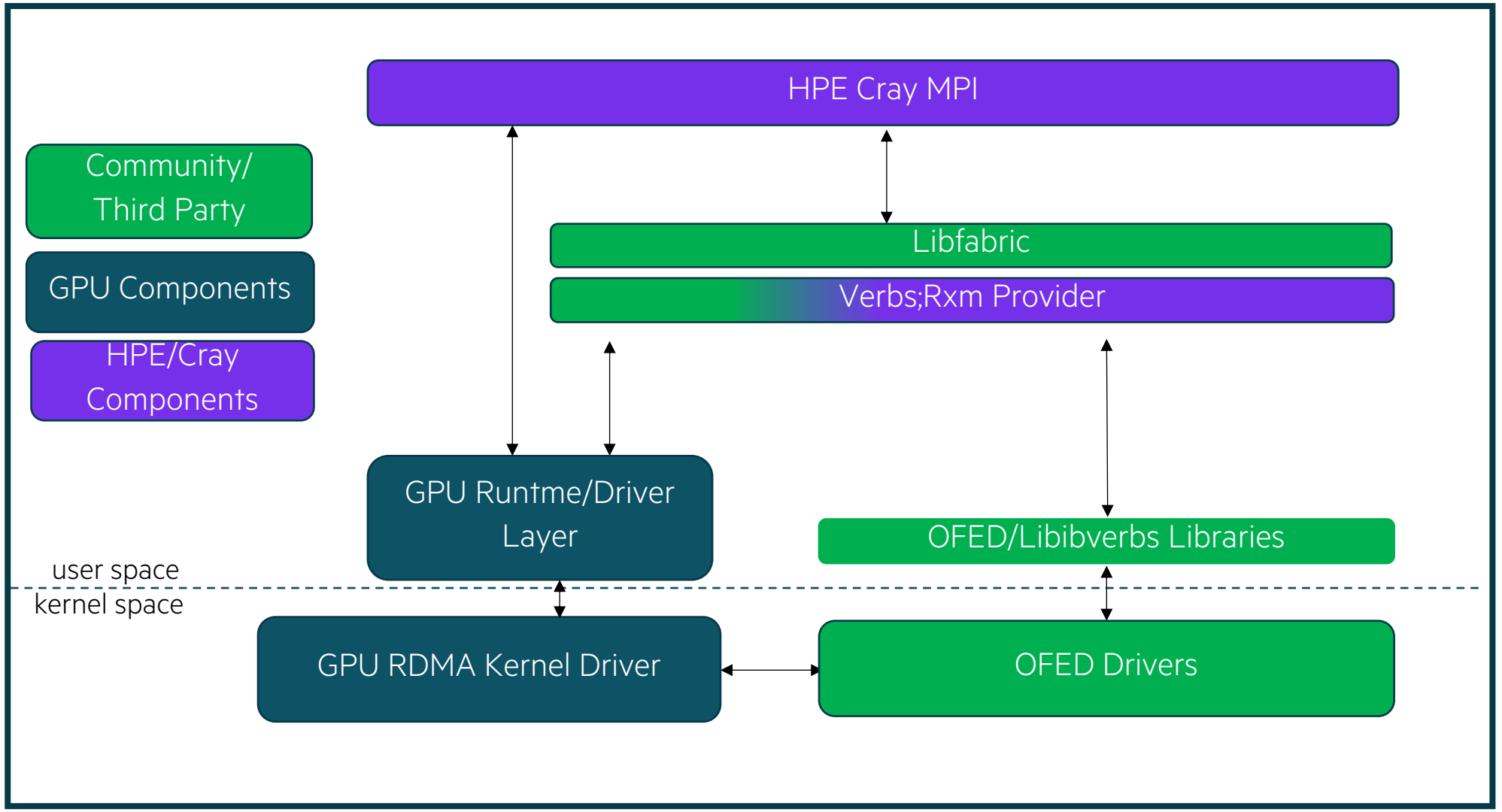
- HPE Cray MPI overview
- HPE Cray MPI tuning & placement
- GPU support in HPE Cray MPI
  - Overview
  - GPU-NIC Async features

HPE CRAY MPI – IMPLEMENTATION OVERVIEW

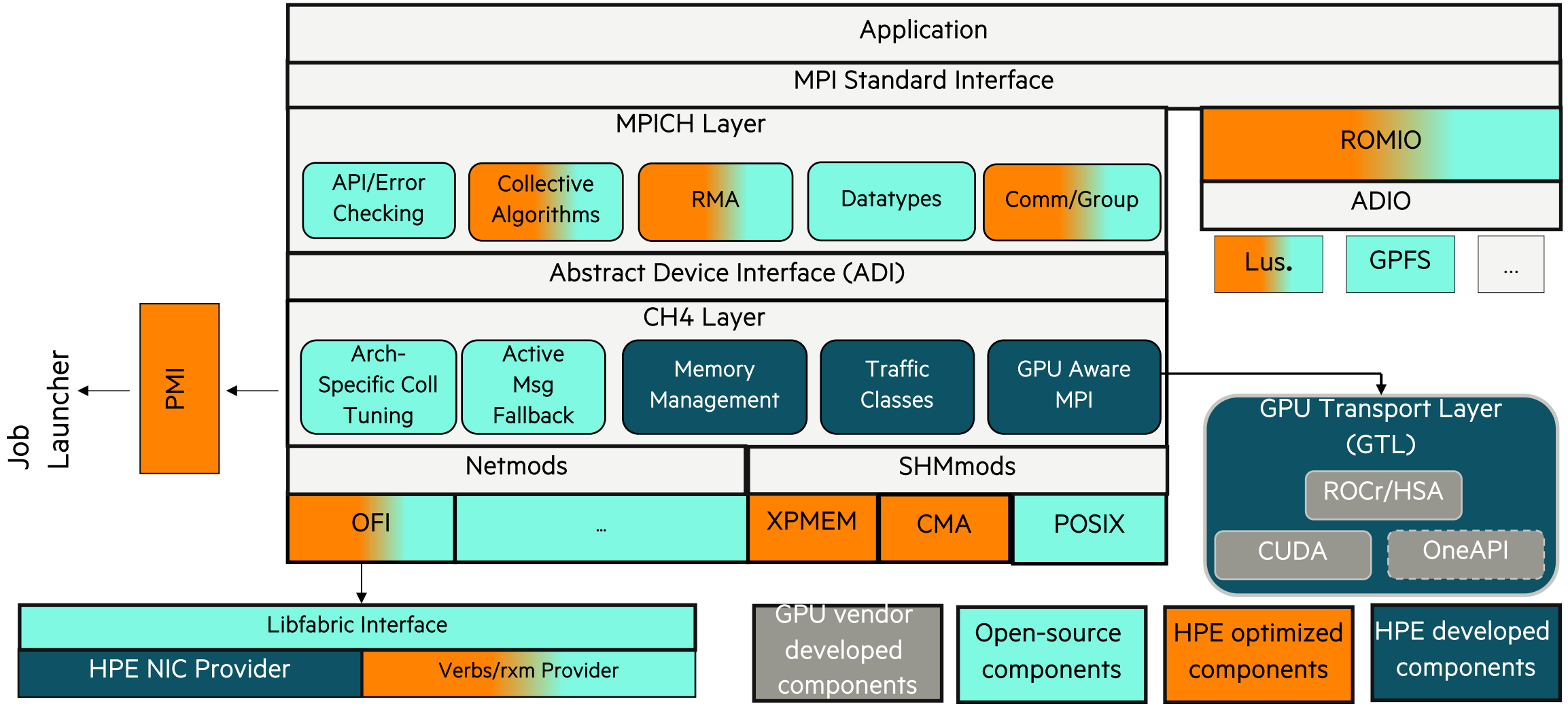
KEY FEATURES AND OPTIMIZATIONS – HPE CRAY MPI

- Robust support for multiple GPU architectures (AMD and NVIDIA)
- Performance and scaling optimizations for many collectives
- Small-msg on-node collective optimizations
- Support for multiple NICs per node
- Notable MPI I/O performance enhancements and stats
- Support for huge page allocations and memory management enhancements
- Scalable Cray PMI implementation for fast launch/job startup
- Flexible, intuitive rank re-ordering feature

MPI COMMUNICATION SOFTWARE STACK FOR HPE SLINGSHOT-10



CRAY MPI SOFTWARE ARCHITECTURE



HPE CRAY MPI – TUNING AND PLACEMENT

MPI MESSAGE PROTOCOLS

- Message consists of envelope and data
  - Envelope contains tag, communicator, length, source information, plus implementation private data
- Eager (short)
  - Message is sent, based on the expectation that the destination can store; if no matching receive exists, the receiver must buffer or drop
- Segmentation And Reassembly (SAR)
  - Longer message is broken into chunks that are each like Eager messages
- Rendezvous (long)
  - Only the envelope is sent (and buffered) immediately
  - Message is not sent until the destination posts a receive
  - A long message is any message longer than a short message
- For more information see the `intro_mpi` man page

NEW MPI ENVIRONMENT VARIABLES FOR EX

- The default netmod that Cray-MPICH uses is libfabric (OFI)
  - Libfabric is an open-source project as a subgroup of the OpenFabrics Alliance

- Cray-MPICH explicitly sets default values for a subset of new OFI environment variables
  - These variables generally start with "`FI_`" or "`MPICH_OFI_`"
  - See the `intro_mpi` man page for additional information

- Cray MPICH is the primary source of GPU support in the Cray Programming Environment
  - Both NVIDIA and AMD GPUs are supported by Cray MPICH as of the current release
  - Several environment variables are used to manage GPUs in MPI operations
    - These variables generally start with "`MPICH_GPU_`"

NEW LIBFABRIC OFI ENVIRONMENT VARIABLES

Environment Variable	Purpose
`FI_OFI_RXM_BUFFER_SIZE`	Specifies the transmit buffer size/inject size in bytes. Messages of size less than this will be transmitted via an eager protocol and those above will be transmitted via a rendezvous or SAR (Segmentation And Reassembly) protocol. Default is 16,364
`FI_OFI_RXM_SAR_LIMIT`	Messages of size greater than this (in bytes) are transmitted via rendezvous protocol. Setting this to 0 disables SAR protocol entirely. Default is 262,144
`MPICH_OFI_USE_PROVIDER`	Specifies the libfabric provider to use. By default, the "verbs;ofi_rxm" provider is selected for Slingshot-10 systems.
`MPICH_OFI_VERBOSE`	If set, displays verbose output during MPI_Init to verify which libfabric provider was selected, along with the name and address of the NIC(s) being used. Not set by default
`FI_VERBS_MIN_RNR_TIMER`	This sets the minimum backoff time used when the Mellanox NICs experience congestion. Allowable values are 0-31, with higher values corresponding to longer backoffs. recommended value for Slingshot10 is 3 to 6 - Default is 6

NEW LIBFABRIC OFI ENVIRONMENT VARIABLES FOR MULTI-INJECTION

Environment Variable	Default Value	Purpose
MPICH_OFI_NIC_POLICY	Block	Selects the rank-to-NIC assignment policy used by Cray MPI. Options: BLOCK, ROUND-ROBIN, NUMA, GPU, and USER
MPICH_OFI_NIC_MAPPING	Unset	Specifies the precise rank-to-NIC mapping to use on each node.
MPICH_OFI_NIC_VERBOSE	0	If set to 1, verbose information pertaining to NIC selection is printed at the start of the job.
MPICH_OFI_NUM_NICs	Unset	Specifies the number of NICs the job can use on a per-node basis. By default, when multiple NICs per node are available, MPI attempts to use them all.
MPICH_OFI_RMA_STARTUP_CONNECT	0	If set to 1, Cray MPI will create connections between all ranks on each node in the job during MPI_Init. May be beneficial for RMA jobs requiring an all-to-all on-node communication pattern.
MPICH_OFI_SKIP_NIC_SYMMETRY_TEST	0	If set to 1, the check for NIC symmetry (i.e. make sure all nodes in the job have the same number of Nics available) performed during MPI_Init will be bypassed.

NEW MPI ENVIRONMENT VARIABLES FOR GPU SUPPORT

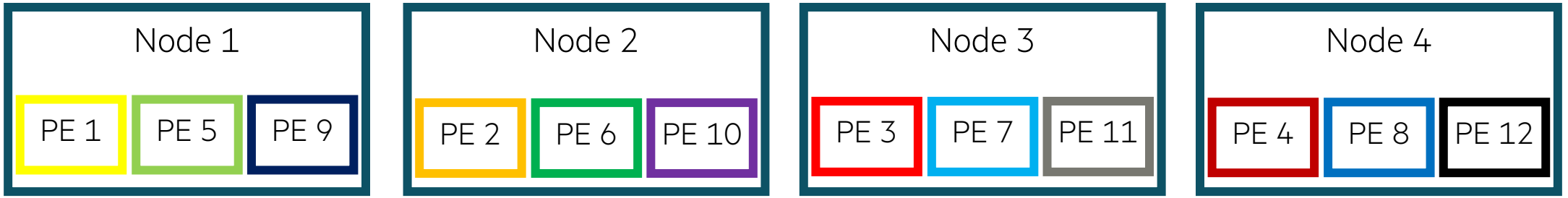
Environment Variable	Default Value	Purpose
MPICH_GPU_SUPPORT_ENABLED	0	Enables a parallel application to perform MPI operations with communication buffers that are on GPU-attached memory regions.
MPICH_GPU_IPC_ENABLED	1	Enables GPU IPC support for intra-node GPU-GPU communication operations.
MPICH_GPU_EAGER_REGISTER_HOST_MEM	1	Registers the CPU-attached shared memory regions with the GPU runtime layers.
MPICH_GPU_IPC_THRESHOLD	8192	Intra-node GPU-GPU transfers with payloads of size greater than or equal to this value will use the IPC capability. Transfers with smaller payloads will use CPU-attached shared memory regions.
MPICH_GPU_NO_ASYNC_MEMCPY	1	Enables optimization for intra-node MPI transfers involving CPU and GPU buffers. If set to 0, it reverts to using blocking memcpy operations for intra-node MPI transfers involving CPU and GPU buffers.
MPICH_GPU_NO_ASYNC_MEMCPY	0	Enables experimental optimization for collective operations (e.g. MPI_ALLreduce) involving GPU-GPU transfers with large payloads

RANK PLACEMENT

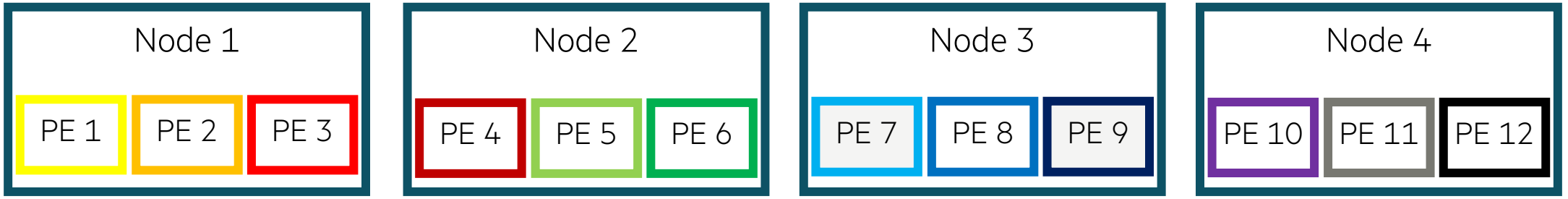
- Ordering of ranks is controlled by the environment variable `MPICH_RANK_REORDER_METHOD`
- Four possible values:
  - 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: (DEFAULT) SMP-style placement – Sequential ranks fill up each node before moving to the next.
  - 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.
- When is rank placement useful?
  - When point-to-point communication consumes a significant fraction of program time and a load imbalance is detected
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out IO across nodes
  - Frankly, little real-world experience has been gained on Slingshot-10 systems like Spock …yet.

RANK PLACEMENT OPTIONS

MPICH_RANK_REORDER_METHOD =0 (Round-robin placement)



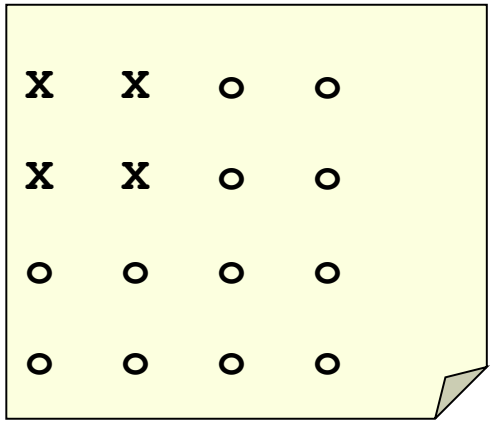
MPICH_RANK_REORDER_METHOD =1 (SMP-style placement)



MPICH_RANK_REORDER_METHOD =2 (Folded rank placement)

RANK ORDER CHOICES

- Many options, depends on pattern
  - Check out pat_report, grid_order, and mgrid_order for generating custom rank orders based on:
    - Measured data
    - Communication patterns
    - Data decomposition
  - Nodes marked X use a shared resource heavily
    - If the shared resource is:
      - Memory bandwidth: scatter the X's
      - Network bandwidth to others, again scatter
      - Network bandwidth among themselves, concentrate



GPU SUPPORT IN HPE CRAY MPI

GPU SUPPORT STATUS IN HPE CRAY MPI

- HPE Cray MPI is optimized and tuned for AMD and NVIDIA GPUs on HPE Slingshot-10 systems

- Current support status on HPE Slingshot-10 systems with AMD and NVIDIA GPUs:
  - Intra-node GPU-GPU Peer-to-Peer IPC
    - Optimized intra-node transfers between GPU-attached memory regions
    - Efficient data movement mechanisms for transfers between CPU- and GPU-attached memory regions

  - Inter-node GPU-NIC RDMA
    - Enables direct transfers between NIC and GPU without requiring CPU-attached memory staging areas

  - Efficient multi-NIC and multi-GPU support
    - HPE Cray MPI strives to select the best NIC for each process based on process-to-CPU and process-to-GPU mappings

  - GPU Managed memory is functionally supported

GPU-NIC ASYNCHRONOUS: PREVIEW

GPU-NIC ASYNC OVERVIEW

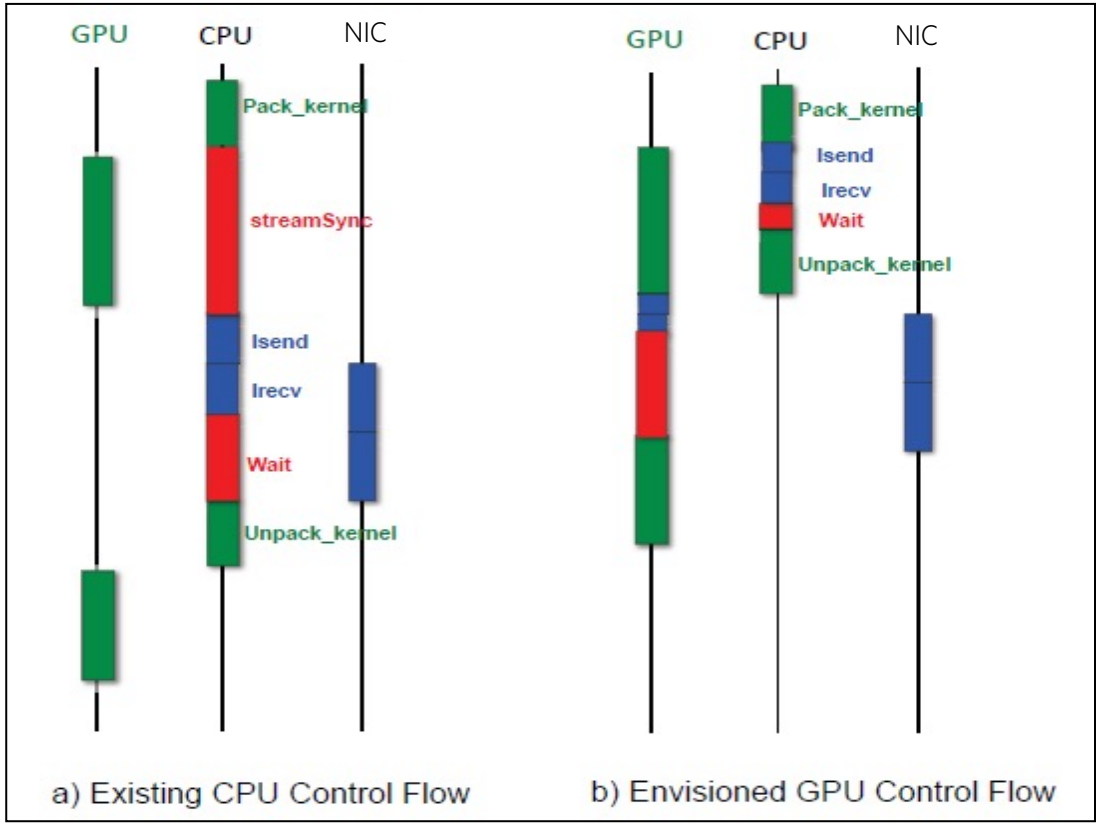
- Current MPI solutions require CPU cycles to orchestrate control and data flows for GPU-enabled parallel applications
  - Most GPU applications currently involve distinct phases of compute and communication
    - Achieving efficient communication/computation overlap is challenging
  - Efficiently utilizing compute and networking resources requires non-trivial amount of application redesign
    - Such solutions may not be portable across different system architectures

- GPU-NIC Async proposals
  - Decouple CPU / GPU control and data paths
  - Reduce frequency and overheads of CPU / GPU synchronization points
  - Potentially improve utilization of all three critical resources on the compute nodes: CPU, GPU, and NIC
  - New MPI APIs and API extensions are being developed
    - Requires application-level changes

GPU-NIC ASYNC: STREAM TRIGGERED (ST)

GPU-NIC ASYNC STREAM TRIGGERED (ST): COMMUNITY INTEREST (*)

```
for (timestep = 0; …) {
1   compute_interior_kernel <<<…,interior_stream>>> (…)
2   pack_kernel <<<…,boundary_stream>>> (…)
3   cudaStreamSynchronize(boundary_stream)
    MPI_Irecv(…)
4   MPI_Isend(…)
    MPI_Waitall(…)
5   unpack_kernel <<<…, boundary_stream>>> (…)
6   compute_xboundary_kernel <<<…, boundary_stream>>> (…)
7   compute_yboundary_kernel <<<…, boundary_stream>>> (…)
8   cudaDeviceSynchronize(…)
}
```



Minimizes CPU – GPU synchronization overheads, communication still occurs at kernel boundary

(* A. Venkatesh , C. Chu , K. Hamidouche , S. Potluri , Davide Rossetti , and DK Panda, "**MPI-GDS: High Performance MPI Designs with GPUDirect-aSync for CPU-GPU Control Flow Decoupling**", ICPP 2017 : International Conference on Parallel Processing, Aug 2017)

HPE'S GPU-NIC ASYNC (ST) PROPOSAL (1/2)

- MPIX_Isend_stream(const void *send_buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *send_request, void *stream);
  - MPI implementation creates a send-request object, operation has not been initiated

- MPIX_Enqueue_stream(MPIX_STREAM_TRIGGER_ISEND_OPS, void *stream);
  - MPI implementation creates Libfabric Deferred Work Queue elements for MPIX_Isend_stream operations
  - MPI, OFI, and GPU runtime layers will launch the execution of the Deferred Work Queue elements at a later point in time
    - Deferred Work Queue elements are processed in GPU stream order

- MPIX_Enqueue_stream(MPIX_STREAM_WAIT_ISEND_OPS, void *stream);
  - MPI, OFI, and GPU runtime layers will track completion of pending "MPIX_Isend_stream" operations
  - Subsequent kernels appended to the GPU stream can access "send_buf" contents safely

HPE'S GPU-NIC ASYNC (ST) PROPOSAL (2/2)

- MPIX_Irecv_stream(void *recv_buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *recv_request, void *stream);
  - MPI implementation creates a recv-request object, operation has not been initiated

- MPIX_Enqueue_stream(MPIX_STREAM_TRIGGER_IRECV_OPS, void *stream);
  - MPI implementation creates Libfabric Deferred Work Queue elements for MPI_Irecv_stream operations
  - MPI, OFI, and GPU runtime layers will launch the execution of the Deferred Work Queue elements at a later point in time
    - Deferred Work Queue elements are processed in GPU stream order

- MPIX_Enqueue_stream(MPIX_STREAM_WAIT_IRECV_OPS, void *stream);
  - MPI, OFI, and GPU runtime layers will track completion of pending "MPIX_Irecv_stream" operations
  - Subsequent kernels appended to the GPU stream can access "recv_buf" contents safely

GPU-NIC ASYNC: KERNEL TRIGGERED (KT)

GPU-NIC ASYNC KERNEL TRIGGERED (KT): OVERVIEW

- Potential issues with the GPU-NIC Async ST approach:
  - Communication operations still occur at kernel boundaries
  - Kernel launch and teardown overheads are not fully eliminated

- Design and implementation criteria for the GPU-NIC Async KT approach:
  - CPU cycles are required to define communication requests ahead of time
    - For KT: Communication pattern needs to be known ahead of time
  - Communication operations are started from **within** GPU kernels
  - Long running kernels minimize kernel launch and teardown overheads
  - Potentially improves communication/computation overlap
  - Communication runtime implementations are more complex:
    - GPU threads must perform select MPI operations
    - CPU, GPU, and NIC resources need to be efficiently managed

PERSISTENT API PROPOSAL FOR GPU-NIC ASYNC KT

- Using existing persistent MPI APIs

```
int MPI_Send_init (const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)
int MPI_Recv_int (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)
```

- Extending persistent start and wait operations to support GPU-NIC Async KT:
  - Supports only MPI_STATUS_IGNORE in the initial version

```
__device__ int MPI_Start( MPI_Request * request )
__device__ int MPI_Startall( int count, MPI_Request array_of_requests[] )

__device__ int MPI_Wait( MPI_Request * request, MPI_Status * status )
__device__ int MPI_Waitall( int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[] )

__device__ int MPI_Test( MPI_Request *request, int *flag, MPI_Status * status )
__device__ int MPI_Testall( int count, MPI_Request array_of_requests[], int *flag, MPI_Status array_of_statuses[] )
```

SUGGESTIONS FOR SPOCK

IMPROVEMENTS IN HPE CRAY PE 21.05 VERSUS 21.04 RELEASES

- Upgrade early and often!

- Fixes to module **`cray-mpich`**
  - No longer need to set **`PE_MPICH_GTL_DIR_amd_gfx908`**
  - No longer need to set **`PE_MPICH_GTL_LIBS_amd_gfx908`**

- Fix to optimized MPI collectives
  - No longer need to set **`MPIR_CVAR_GPU_EAGER_DEVICE_MEM=0`**

- Improvements to compute node stability

RESOURCES

- Very nice Spock Quick-Start Guide: https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html#
- $ man **intro_mpi**

THANK YOU

QUESTIONS?