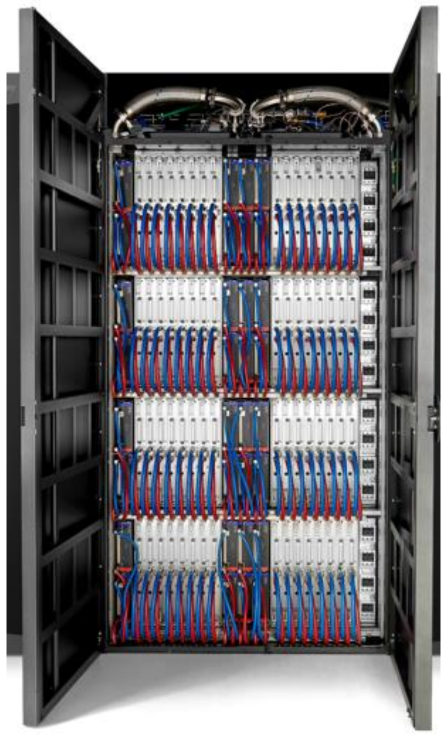
Hewlett Packard
Enterprise

HLRS Hunter – Architecture

Christian Simmendinger (HPE)

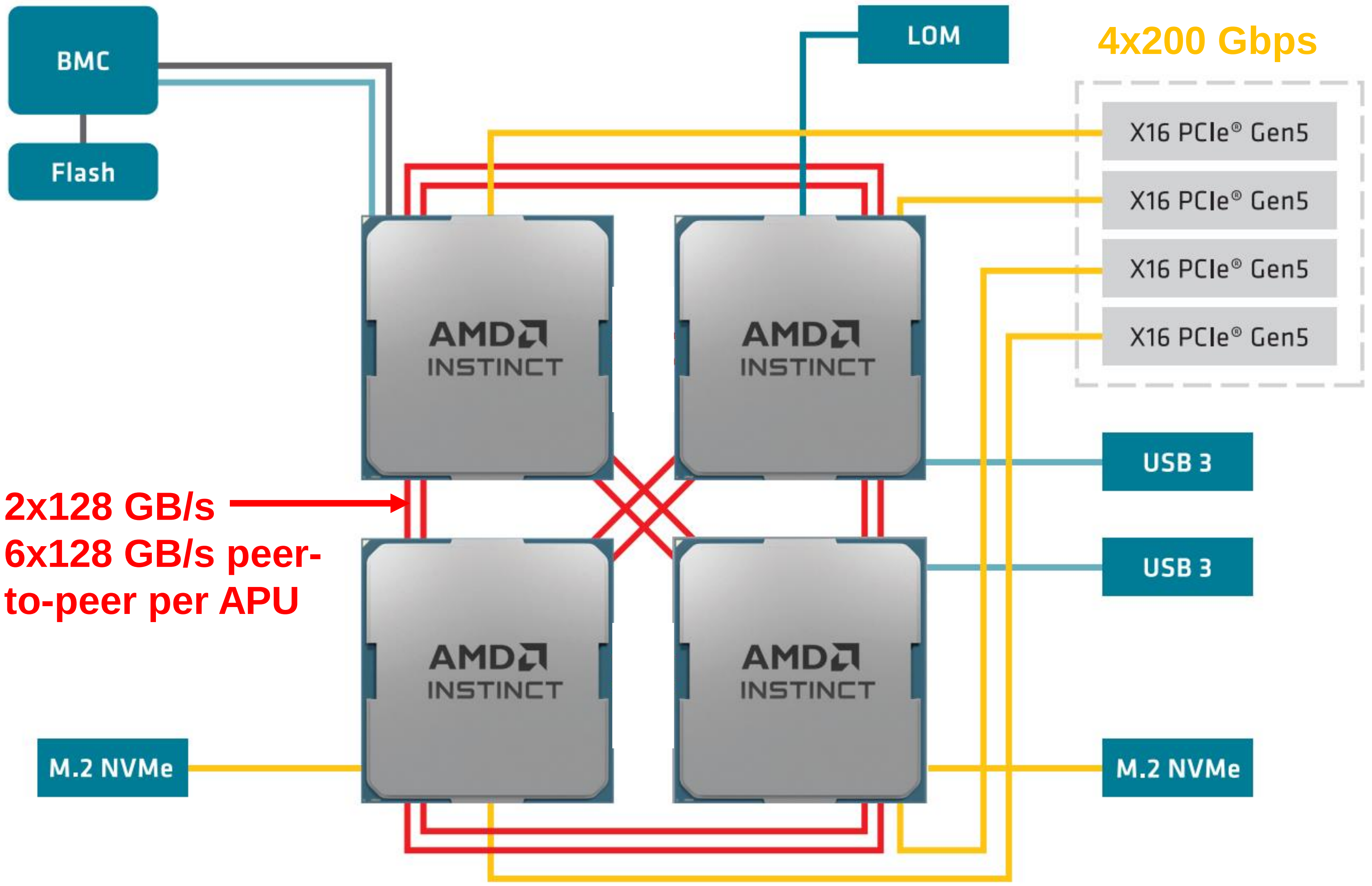
Januar 2025

Hunter – Stepping Stone System



- Hunter will be based on the HPE Cray EX4000 platform
  - HPE Cray EX255a (El Capitan blade architecture, MI-300A)
  - HPE Cray Slingshot Interconnect
- Work File Systems
  - HPE Cray ClusterStor E2000 Lustre Appliance
  - FS1: 13PB
  - FS2: 13PB
- Home File System: 540TB

HPE Cray Supercomputing EX255A Node Architecture



HPE Cray Supercomputing EX255a Specs

HPE Cray Supercomputing EX255a		Hawk Apollo 9000
Form Factor	1U blade for EX4000 and EX2500	1U blade Apollo 9000
Processors	AMD MI300A APU	EPYC 7742 CPU
Compute Blade	Two 4-socket MI300A APU nodes	Four 2-socket AMD Rome nodes
Core Count	24 CPU Cores and 228 Compute Cores per APU 96 CPU Cores and 912 Compute Cores per node	64 CPU Cores per CPU, 128 CPU Cores per node
Memory / blade	128GB HBM3 per MI300A APU; 512GB HBM3 per node	128GB DDR4 per socket, 256GB per Node
Memory Technology	HBM3 ~5,3 TB/s per MI300A APU	DDR4 ~205 GB/s per CPU socket
Intra Node	6x 128GB/s per APU, 2x 128GB/s Peer-to-Peer	96 GB/s Peer-to-Peer
Local Storage	0 or 1 local NVMe M.2 SSD per node	-
Fabric Option	HPE Slingshot 11 (4 injection ports per node, 4x 200 Gbps)	Infiniband HDR200 Socket-direct (1 injection port per node, 1x 200 Gbps)

HPE Cray Slingshot

HPE Slingshot

Dragonfly Network Architecture

- Packet-by-packet routing of unordered traffic (e.g. MPI/Lustre bulk data) optimally routed at each hop
- Adaptive routing of ordered traffic (e.g. Ethernet) Each new flow can take an optimal new path

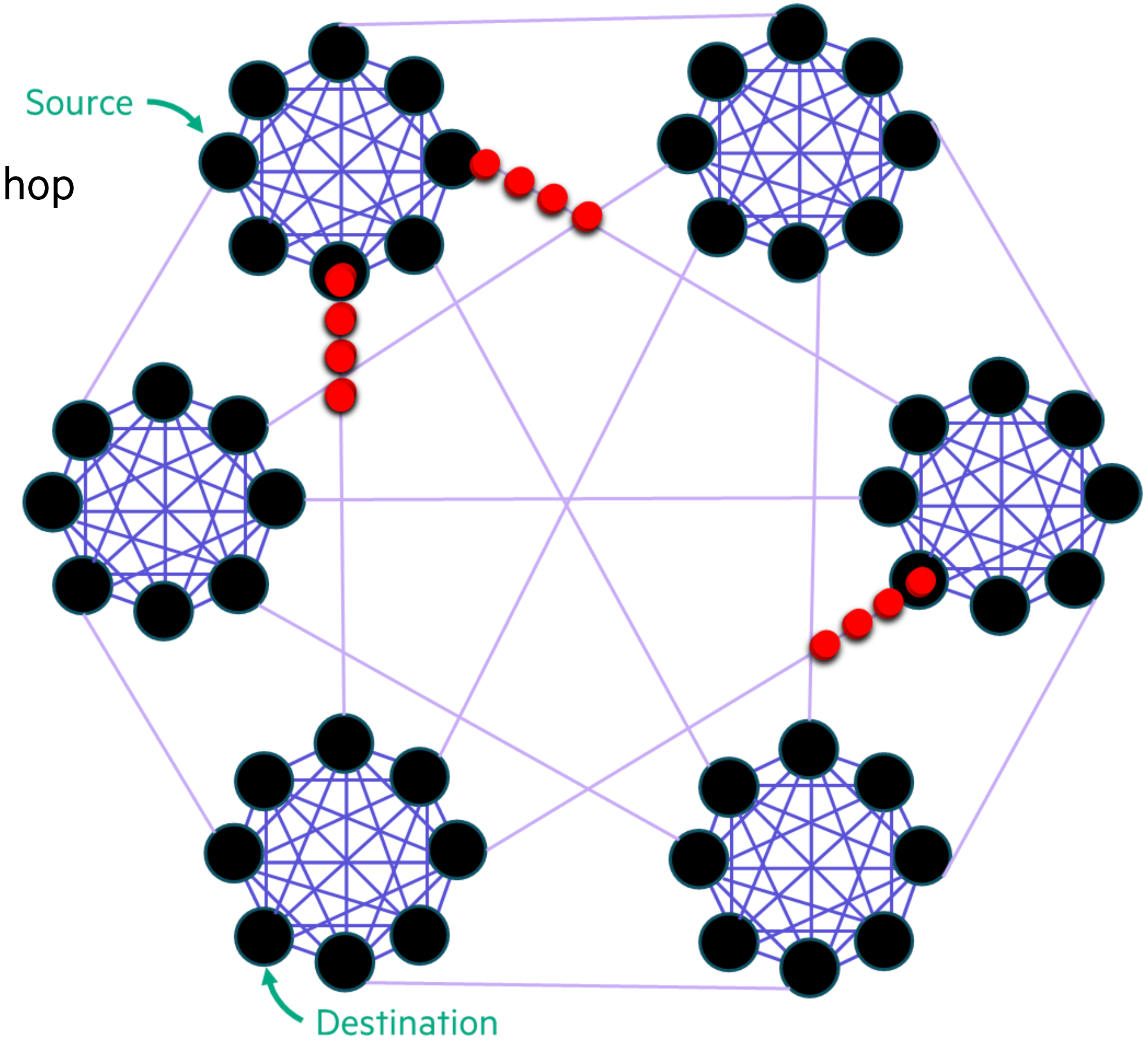
Rosetta Switch

64 port switch, 200 Gb/s

- Advanced adaptive routing
- Congestion control, QoS

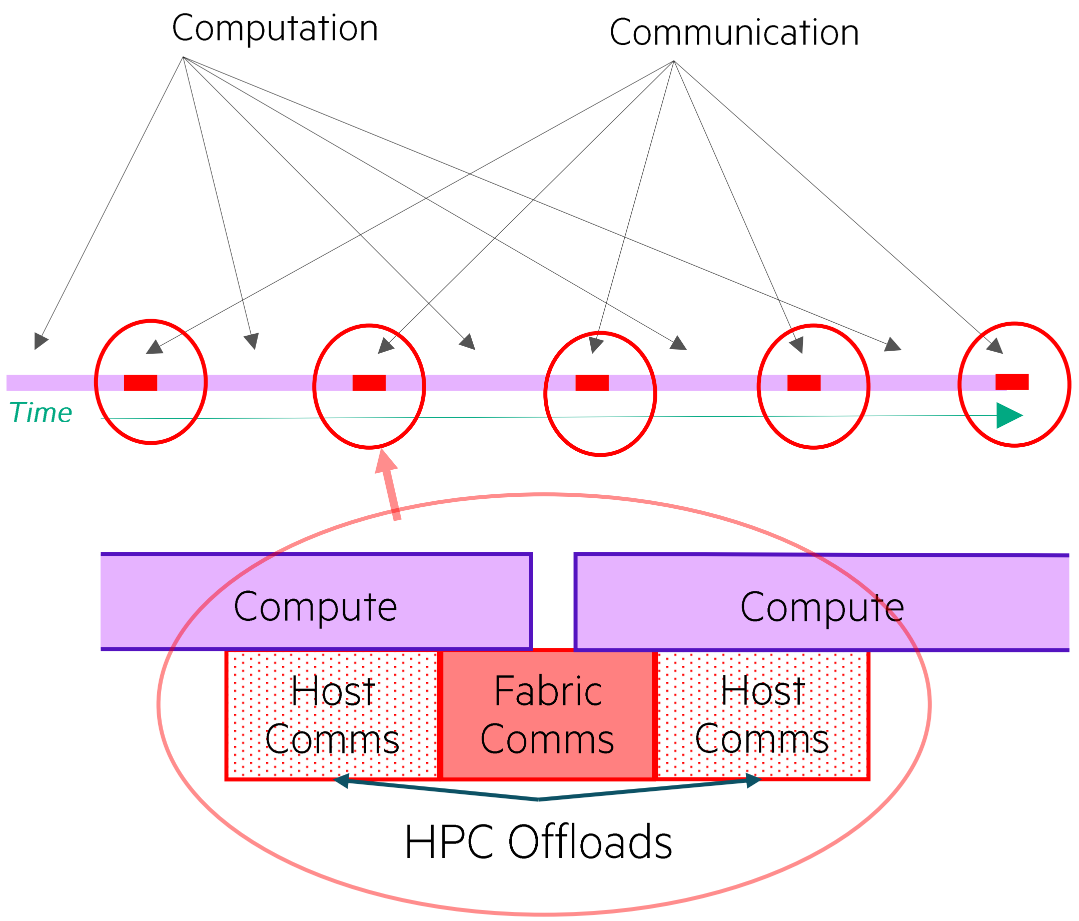
Cassini NIC

- MPI hardware tag matching
- MPI progress engine
- Hardware support for one-sided operations
- Hardware support for collective operations
- 200 Gb/s



Achieving great performance on tightly coupled codes

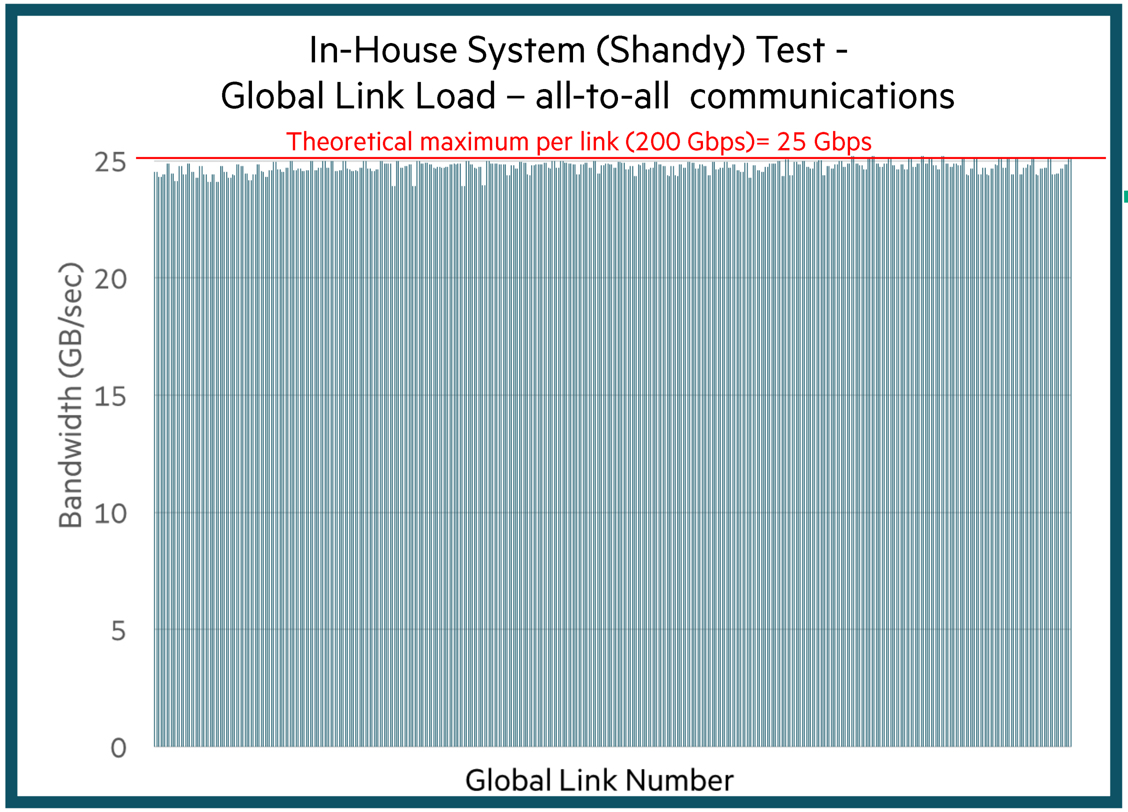
- Objective: overlap comms and compute

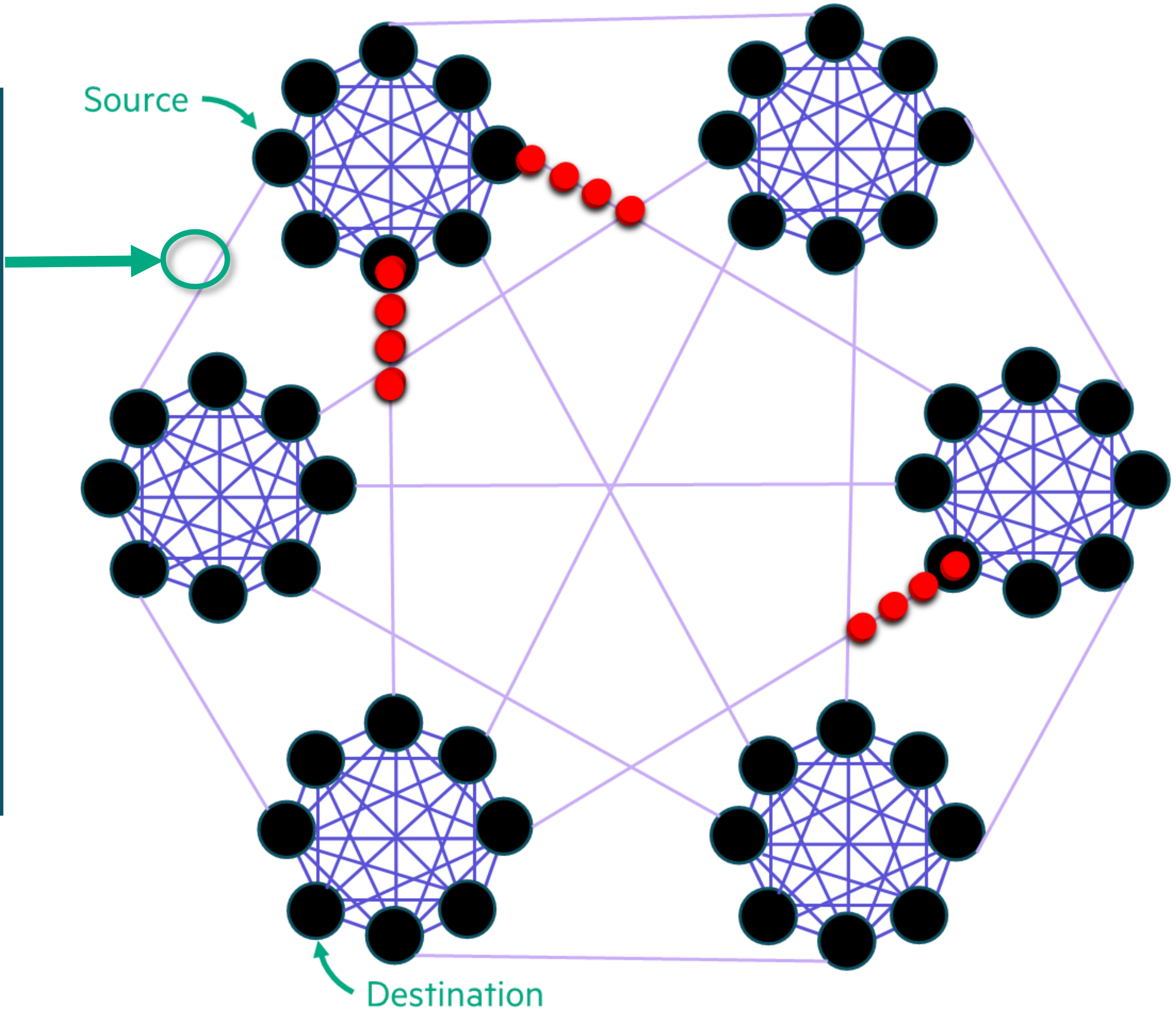


Acceleration Goals:

- Bypass host for processing communications
- Reduce overhead for message orchestration
- Reduce the number of messages needed
- Simplify writing of codes with “strong progression”

Achieving near maximal Bandwidth with fine grained adaptive routing





"Shandy" in-house system

- 8 groups, 1024 nodes
- Dual CX5 injection per node
- 25 TB/s aggregate injection BW
- 50% global bandwidth taper
- 12.5 TB/s aggregate global BW

Hewlett Packard
Enterprise

MPI - AMD mi300A

Tim Mattox (HPE)
Christian Simmendinger (HPE)

Januar 2025

GPU-Aware MPI Communication

GPU –Aware MPI and GPUDirect RDMA

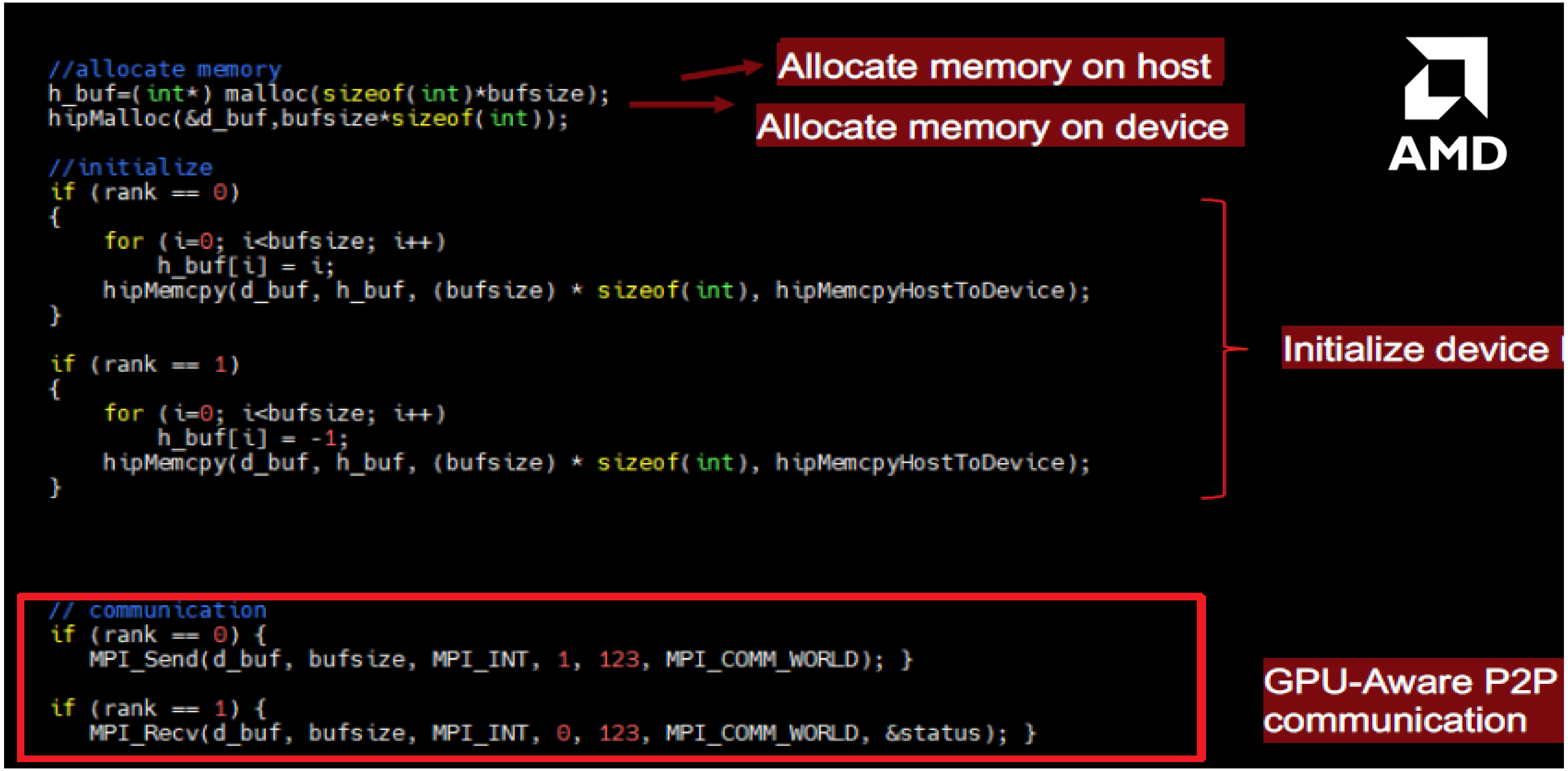
GPU-Aware MPI

- Traditionally, only pointers of the host buffers could be passed to MPI calls.
- GPU aware MPI provides the opportunity to pass GPU buffers to MPI calls.
- Without GPU-Aware MPI, GPU buffers have to be staged through host memory with hipMemcpy
- Many MPI implementations including CRAY-MPICH, MVAPICH and OpenMPI support GPU-Aware MPI

GPU Direct RDMA

- GPU Direct RDMA is a technology that provides the opportunity for network adapters to directly access GPU devices memory and bypass the host
- Note that GPU-Aware MPI refers to support passing GPU buffers to MPI calls in MPI implementations while GPUDirect RDMA is a technology that enables direct access to GPU memory.
- A GPU-Aware MPI may or may not use GPUDirect RDMA for communication between GPUs.

GPU –Aware MPI



```
//allocate memory
h_buf=(int*) malloc(sizeof(int)*bufsize);
hipMalloc(&d_buf,bufsize*sizeof(int));

//initialize
if (rank == 0)
{
    for (i=0; i<bufsize; i++)
        h_buf[i] = i;
    hipMemcpy(d_buf, h_buf, (bufsize) * sizeof(int), hipMemcpyHostToDevice);
}

if (rank == 1)
{
    for (i=0; i<bufsize; i++)
        h_buf[i] = -1;
    hipMemcpy(d_buf, h_buf, (bufsize) * sizeof(int), hipMemcpyHostToDevice);
}


// communication
if (rank == 0) {
   MPI_Send(d_buf, bufsize, MPI_INT, 1, 123, MPI_COMM_WORLD); }

if (rank == 1) {
   MPI_Recv(d_buf, bufsize, MPI_INT, 0, 123, MPI_COMM_WORLD, &status); }
```

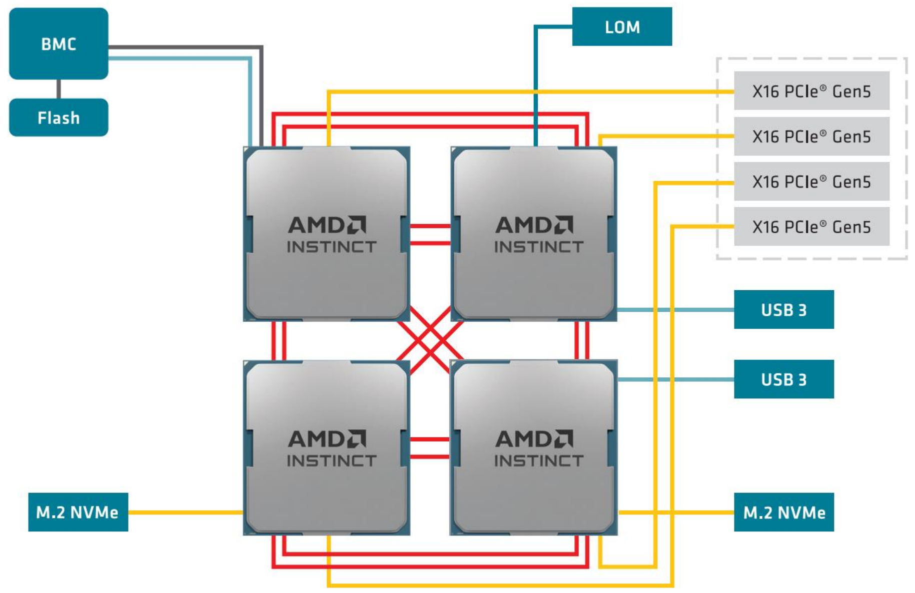
Build Environment

HPE Cray MPI: Building/Compiling

- Load AMD ROCm and Cray MPI modules
  - module load craype-accel-amd-gfx942 (if using OpenMP target offload)
  - module load rocm (you might want to try a newer version than the default)
  - module load cray-mpich (you might want to try a newer version than the default)
- Tell your build system how to link to the MPI and ROCm, in two primary ways
  - Use the HPE Cray Wrapper Compilers: cc/CC/ftn
    - With either module load PrgEnv-cray or PrgEnv-amd
    - Linker flags for the ROCm runtime: -L${ROCM_PATH}/lib -lamdhip64
  - Or specify these compiler and linker flags for your non-wrapped compiler of choice
    - Compiler flags: -I${MPICH_DIR}/include -I${ROCM_PATH}/include
    - Linker flags: -L${MPICH_DIR}/lib –lmpi
    - Linker flags for faster on-node CPU-side communication: -L/opt/xpmem/lib -lxpmem
    - Linker flags to handle GPU resident message buffers:

      ${PE_MPICH_GTL_DIR_amd_gfx942} ${PE_MPICH_GTL_LIBS_amd_gfx942}

Affinity - cray-mpich – mi300a (gpu)



cray-mpich

Affinity check - jobstep

```
https://code.ornl.gov/olcf/hello_jobstep
```

Build & Run

```
mpirun -ppn 4 -np 8 --cpu-bind list:0-23:24-47:48-71:72-95 --gpu-bind list:0:1:2:3
./hello_jobstep | sort -k2n -k5n
MPI 000 - OMP 000 - HWT 002 - Node x1000c0s2b0n0 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 02
MPI 000 - OMP 001 - HWT 003 - Node x1000c0s2b0n0 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 02
MPI 001 - OMP 000 - HWT 026 - Node x1000c0s2b0n0 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID 02
MPI 001 - OMP 001 - HWT 027 - Node x1000c0s2b0n0 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID 02
```

cray-mpich

SPMD pinning

```
export OMP_NUM_THREADS=22
export OMP_PROC_BIND=close
mpirun -ppn 4 -np 8 --cpu-bind list:2-23:26-47:50-71:74-95 --gpu-bind list:0:1:2:3 ./a.out
```

MPMD pinning

```
export OMP_NUM_THREADS=24
export OMP_PROC_BIND=close
mpirun -np 2 --cpu-bind list:0-23:24-47 --gpu-bind list:0:1 ./a.out : -np 1 ./a.out
```

Environment variables and Low Noise Mode (LNM)

NIC binding policies

map process to NIC nearest process’s NUMA domain

- MPICH_OFI_NIC_POLICY=NUMA.

map process to NIC nearest process’s attached GPU

- MPICH_OFI_NIC_POLICY=GPU

user defined mapping

- MPICH_OFI_NIC_POLICY=USER
- Plus: MPICH_OFI_NIC_MAPPING=<nic>:<local process_ids>;

Low Noise Mode

- The Linux OS is generally restricted to run its tasks on core 0
- Similarly, interrupts are mapped to the 1st core of each CCD, leaving 7 per CCD that should be less noisy
- GCD helper tasks can get scheduled on the 2nd core of a CCD
- Avoid all these potentially noisy cores

Environment variables

Rank specific environment - who am I

```
s32708 x1002c0s0b0n0 2045$ mpirun -ppn 1 -np 1 printenv | grep -e PMI.*RANK
PMI_RANK=0
PMI_LOCAL_RANK=0
```

Multiple ranks per GPU

```
mpirun -ppn 8 -np 16 --cpu-bind list:2:3:26:27:50:51:74:75 --gpu-bind list:0:0:1:1:2:2:3:3
MPI 000 - OMP 000 - HWT 002 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 02
MPI 001 - OMP 000 - HWT 003 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID 02
MPI 002 - OMP 000 - HWT 026 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID 02
MPI 003 - OMP 000 - HWT 027 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID 02
MPI 004 - OMP 000 - HWT 050 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID 02
MPI 005 - OMP 000 - HWT 051 - Node x1000c0s1b0n0 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID 02
```

MPI Best Practices

MPI Best Practices

- Set (and verify) your process, NIC, and GPU affinities
- Check the MPI error/return codes
- Post non-blocking receives before their matching sends
- Don’t abuse the MPI layer… It isn’t magic
  - It can only handle a finite number of “things” at once: Communicators, Tags, and Pending Messages
  - Give MPI a chance to make progress
- Avoid *unnecessary* use of **MPI_ANY_SOURCE** or **MPI_ANY_TAG**
- Don’t roll-your-own MPI Collectives
  - File a bug if performance is not what you expect (unless you are an HPL developer :-)
  - Why not? Roll-your-own won’t take advantage of hardware collective acceleration
  - Also, have a look at the “new” non-blocking collectives in MPI-3

Communication Patterns

How to post non-blocking receives before their matching sends

I put the i-receives before the sends, right?

```
for (i = 0; i < dimensions; ++i) {
  MPI_Irecv;
  MPI_Isend;
}
MPI_Waitall;
```

Well, not exactly…

You need to do it across ranks, like this:

```
for (i = 0; i < dimensions; ++i) {
  MPI_Irecv;
}
// Do some useful compute here
// Maybe do MPI_Barrier here
for (i = 0; i < dimensions; ++i) {
  // Okay place for some compute†
  MPI_Isend;
  // Better place for some compute†
}
// Best place for compute overlap†
MPI_Waitall;
```

MPI Best Practices, Explanations
Why post non-blocking receives before their matching sends?

- If the **MPI_Irecv** is *already* posted when a message arrives:
  - The payload can go directly into the destination buffer without needing to make a temporary copy
  - Even for larger messages using the rendezvous protocol, the RDMA-read of the payload can start immediately

  Note: Counted in the LPE_NET_MATCH_PRIORITY Cassini Performance counter
- Otherwise, the message is put into an "Overflow/Unexpected Message" queue
  - For small messages using the eager protocol:
    - The payload is copied into a temporary buffer
    - The payload is copied again when the matching receive is posted
  - For larger messages using the rendezvous protocol:
    - The (bulk of the) payload waits at the sender until the matching receive is posted
    - This waiting delays getting those bytes onto the wire (effectively averaging in "zero" bandwidth during this delay)
  - A non-empty "Unexpected Queue" must be searched for a match any time an MPI receive call is made.

  Note: Counted in the LPE_NET_MATCH_OVERFLOW Cassini Performance counter

How to give MPI a chance to make progress?

Cassini NICs do **both** tag-matching and progress the rendezvous protocol in hardware!

However, if profiling shows a non-trivial amount of time in MPI_Wait, etc., try one of these two methods:

Make calls into MPI every so often

- During "computation overlap" code, *especially* if there are pending non-blocking collectives
- MPI_Testsome is preferred over MPI_Testany
- Many MPI calls will guarantee MPI progress[†]
  - MPI_Test (and all its variants)
  - MPI_Wait (and all its variants)
  - MPI_Request_get_status might be a better alternative with an MPI 4.1 standard implementation

[†]Note: The MPI 4.1 standard has clarified "progress guarantees" for several MPI calls.

Set MPICH_ASYNC_PROGRESS=1

- This will spawn an MPI progress thread
- That thread will need a CPU core[‡]
- Forces thread-safety to **MPI_THREAD_MULTIPLE**
- Might cause some MPI performance overhead

Avoid this if you can. It is rarely actually needed. With Cassini NICs, the only common case that would need this is when using non-blocking collectives and your code doesn't have MPI calls for a long time.

[‡]Note: It is worth a try if you have spare cores

Documentation

HPE Cray MPI: Documentation

- man mpiexec
  - All about job startup, binding
- man intro_mpi
  - Most useful MPI environment variables are documented here
  - Pointers to other useful man pages
- man fi_cxi (The CXI Fabric Provider for libfabrics)
  - Cassini (CXI) is the name of the NIC in the Slingshot-11 network
  - This man page documents environment variables for libfabrics (FI) that are specific to Cassini NICs
- module show cray-mpich
  - Change log and bugs fixed by the currently loaded version

Performance and Tuning

GPU-to-GPU Communication Options

SDMA engine

- Provides the opportunity to overlap communication with computation
- Each SDMA can provide max communication bandwidth of 49GB/s between GCD

Blit kernels

- Lauch kernel to handle communication
- Higher bandwidth
- Cannot overlap communication with computation
- export HSA_ENABLE_SDMA=0

Thank you

christian.simmendinger@hpe.com

© 2025 Hewlett Packard Enterprise Development LP