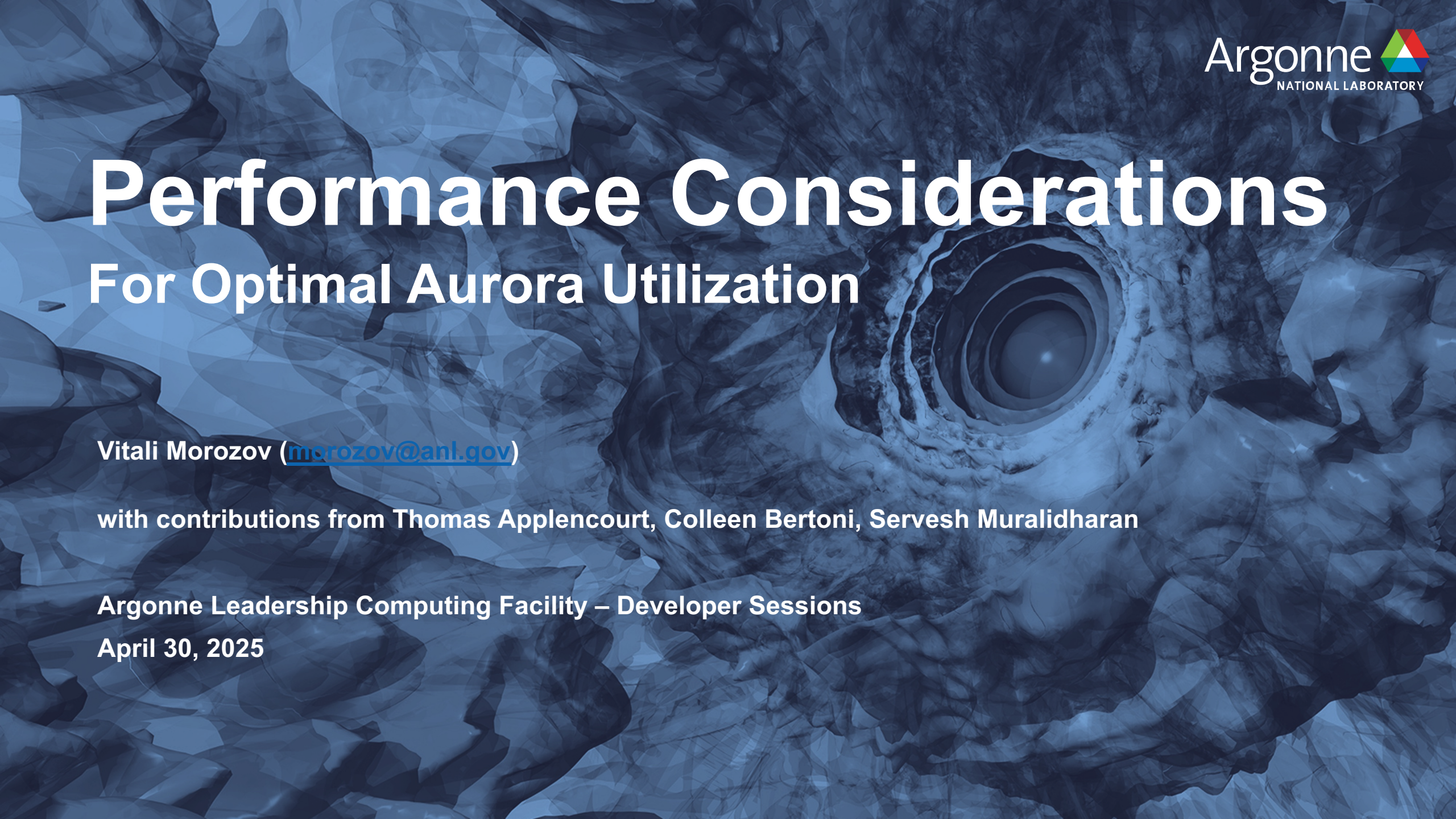


Performance Considerations

For Optimal Aurora Utilization

Vitali Morozov (morozov@anl.gov)

with contributions from Thomas Applencourt, Colleen Bertoni, Servesh Muralidharan

Argonne Leadership Computing Facility – Developer Sessions

April 30, 2025

Overview and Expectations

- ❑ How to get started with compilers, modules, scripts, and queues
- ❑ Node schematics – Locality of the components
- ❑ Runtime identification and settings
- ❑ Mapping
- ❑ Benchmarks

- ❑ Aurora Documentation: https://docs.alcf.anl.gov/aurora/getting-started-on-aurora/
  - ✓ Not Intended to Repeat; Intended to Complement
  - ✓ Not intended to “work-as-is”; Intended to Illustrate

- ❑ SYCL Documentation: https://docs.alcf.anl.gov/polaris/programming-models/sycl-polaris/
  - ✓ Intended to Illustrate in C, OpenMP, and MPI only

How to Get Started

```
morozov@aurora-uan-0010:~> module restore
Running "module reset". Resetting modules to system default. The following $MODULEPATH directories have been removed: None
morozov@aurora-uan-0010:~>
morozov@aurora-uan-0010:~>
morozov@aurora-uan-0010:~> module list

Currently Loaded Modules:
  1) gcc-runtime/12.2.0-267awrk   4) mpc/1.3.1-ygprpb4                   7) hwloc/master-git.1793e43-level-zero  10) libfabric/1.20.1
  2) gmp/6.2.1-yctcuid            5) gcc/12.2.0                          8) yaksa/0.3-aw2kkvy                    11) cray-pals/1.4.0
  3) mpfr/4.2.1-fhgnwe7           6) oneapi/eng-compiler/2024.07.30.002  9) mpich/opt/4.3.0rc3                   12) cray-libpals/1.4.0



morozov@aurora-uan-0010:~>
```

```
morozov@aurora-uan-0012:~> mpi
mpic++        mpicc         mpichversion  mpicxx        mpiexec       mpif77        mpif90        mpifort       mpirun        mpivars
morozov@aurora-uan-0012:~>
morozov@aurora-uan-0012:~> mpicc --version
Intel(R) oneAPI DPC++/C++ Compiler 2025.0.0 (2025.x.0.20240629)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /opt/aurora/24.180.3/updates/oneapi/compiler/eng-20240629/bin/compiler
Configuration file: /opt/aurora/24.180.3/updates/oneapi/compiler/eng-20240629/bin/compiler/../icx.cfg
morozov@aurora-uan-0012:~>
morozov@aurora-uan-0012:~> mpicc -show
icx -I/opt/aurora/24.180.3/spack/unified/0.8.0/install/linux-sles15-x86_64/oneapi-2024.07.30.002/mpich-4.3.0rc3-fzmrfta/include -L/opt/aurora/24.180.3/sp
ack/unified/0.8.0/install/linux-sles15-x86_64/oneapi-2024.07.30.002/mpich-4.3.0rc3-fzmrfta/lib -Wl,-rpath -Wl,/opt/aurora/24.180.3/spack/unified/0.8.0/in
stall/linux-sles15-x86_64/oneapi-2024.07.30.002/mpich-4.3.0rc3-fzmrfta/lib -lmpi
morozov@aurora-uan-0012:~>
morozov@aurora-uan-0012:~> mpif90 --version
ifx (IFX) dev.x.0 Mainline 20240629
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

morozov@aurora-uan-0012:~>
```

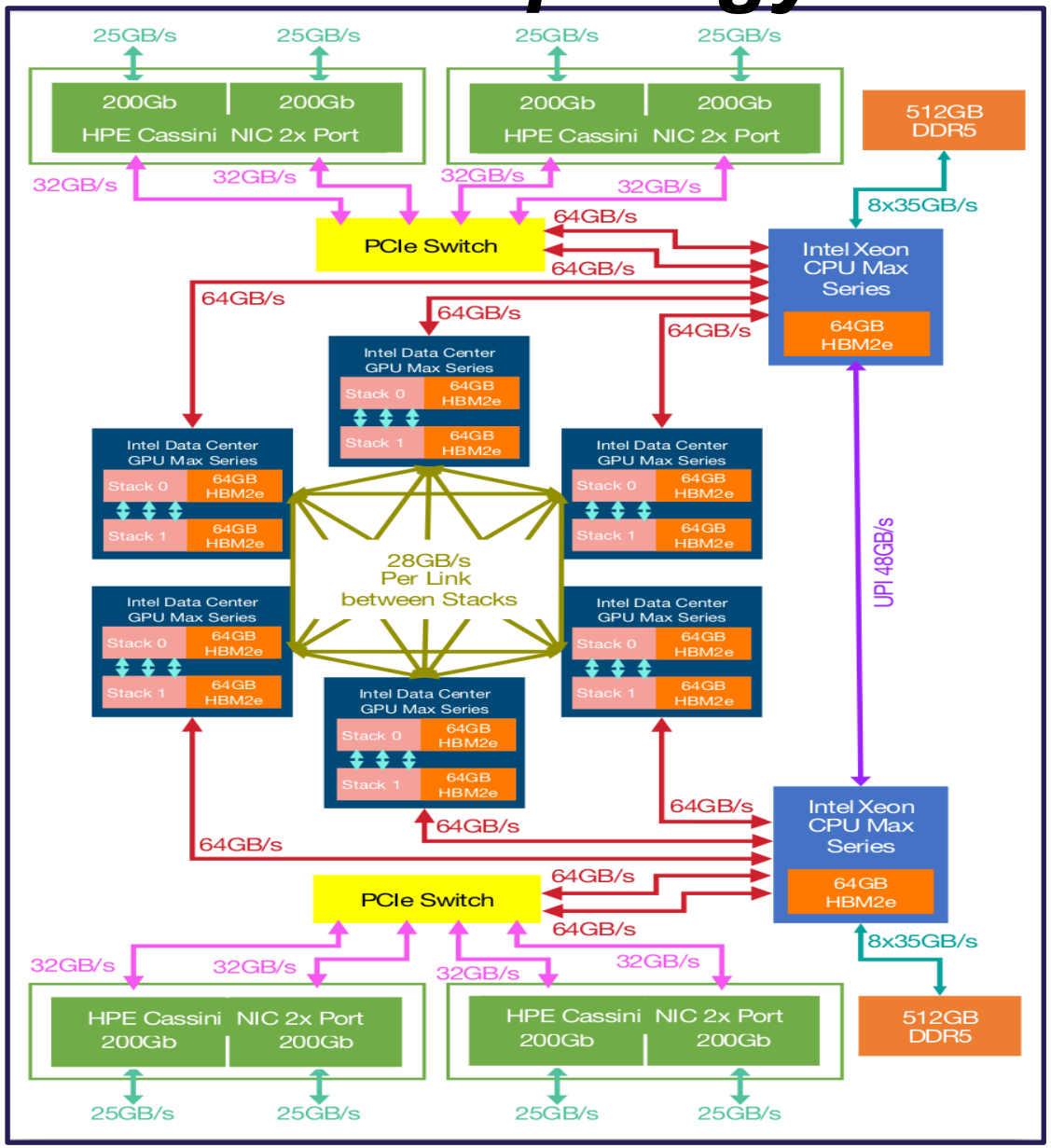
Key points: module restore, gcc-12, oneapi compilers, Aurora mpich, mpicc, mpic++, mpif90

Do Sanity Check: Especially first time

```
morozov@aurora-uan-0010:~> cat hello.c
#include <mpi.h>
#include <stdio.h>

int main()
{
    MPI_Init( NULL, NULL );
    MPI_Finalize();
    return 0;
}

morozov@aurora-uan-0010:~> mpicc -o hello hello.c
morozov@aurora-uan-0010:~> ls -l ./hello
-rwxr-xr-x 1 morozov users 108344 Apr 15 15:01 ./hello
morozov@aurora-uan-0010:~>
```

Key points: make your own tests, identify critical dependencies, test them after each update

Know the topology of the node



- ***Dual socket architecture: socket locality***
  - 52 CPU cores
  - 3 GPUs
  - 4 NICs
  - 512 GB DDR5 memory
  - 64 GB HBM - High-Bandwidth memory
- ***Penalty to access remote-socket hardware***
  - UPI is “only” 48 GB/s
  - Xe-Links could help
  - PCIe Switch could be a bottleneck

Key points:

Locality!!!

Locality!!!

Locality!!!

Know the topology of the node: Sockets and Cores

- ***Dual socket with 52 physical Cores on each socket***
  HyperThreading makes each Core as 2 CPUs

- ***MPI numerates CPUs, not Cores***

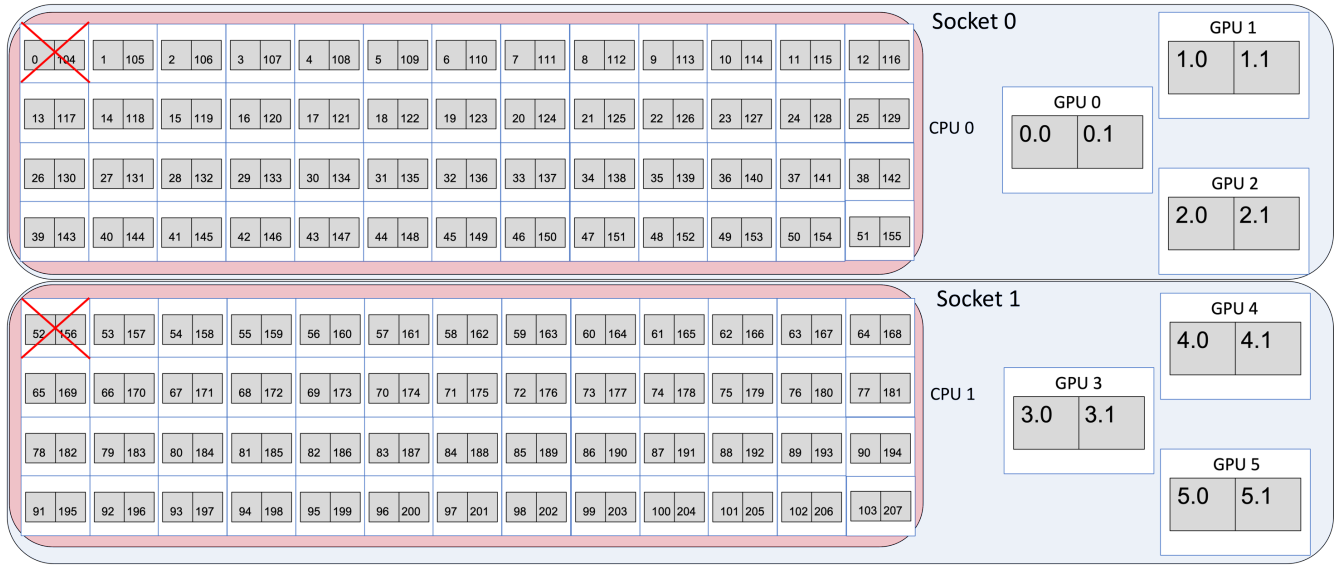
```
CPU0 is Socket0, Core0, Thread0
CPU1 is Socket0, Core1, Thread0
….
CPU51 is Socket0, Core51, Thread0
CPU52 is Socket1, Core0, Thread0
CPU53 is Socket1, Core1, Thread0
…
CPU103 is Socket1, Core51, Thread0
CPU104 is Socket0, Core0, Thread1
CPU105 is Socket0, Core1, Thread1
…
CPU155 is Socket0, Core51, Thread1
CPU156 is Socket1, Core0, Thread1
CPU157 is Socket1, Core1, Thread1
…
CPU206 is Socket1, Core50, Thread1
CPU207 is Socket1, Core51, Thread1
```

Key points:

Core changes first, 0 to 51

Socket changes next, 0 to 1

Thread changes last, 0 to 1



https://docs.alcf.anl.gov/aurora/running-jobs-aurora/#mpi-rank-and-thread-binding-to-cores-and-gpus

Submit a job: Example Scripts

```
NAME=collectives
rm -f ${NAME}.{out,err} core.*

qsub -A Project -N ${NAME} -l nodes=x1921c7s2b0n0+x1921c7s3b0n0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script

or

qsub -A Project -N ${NAME} -l select=64,place=group=tier0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script
```

Key points:

You can ask as the same set of nodes for tuning/debugging/performance evaluation

Submit a job: Example Scripts

```
NAME=collectives
rm -f ${NAME}.{out,err} core.*

qsub -A Project -N ${NAME} -l nodes=x1921c7s2b0n0+x1921c7s3b0n0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script

or

qsub -A Project -N ${NAME} -l select=64,place=group=tier0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script
```

Key points:

You can ask as the same set of nodes for tuning/debugging/performance evaluation
You can ask a full cabinet – 64 nodes compact placement

Submit a job: Example Scripts

```
NAME=collectives
rm -f ${NAME}.{out,err} core.*

qsub -A Project -N ${NAME} -l nodes=x1921c7s2b0n0+x1921c7s3b0n0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script

or

qsub -A Project -N ${NAME} -l select=64,place=group=tier0,walltime=0:10:00 -q prod \
      -l filesystems=home -k doe -o ${NAME}.out -e ${NAME}.err ./pbs-script
```

Key points:

You can ask as the same set of nodes for tuning/debugging/performance evaluation
You can ask a full cabinet – 64 nodes compact placement
Personally, I avoid using interactive jobs

A Script to Run a Job

1. module restore
2. module list
3. echo Working directory is $PBS_O_WORKDIR
4. cd $PBS_O_WORKDIR
5. echo Jobid: $PBS_JOBID
6. echo Running on host `hostname`
7. echo Running on nodes `cat $PBS_NODEFILE`
8. NNODES=`wc -l < $PBS_NODEFILE`
9. cp $PBS_NODEFILE $PBS_O_WORKDIR/nodefile.txt
10. sort -n $PBS_NODEFILE > $PBS_O_WORKDIR/nodelist.sorted.txt
11. ldd ./helloRANKS_PER_NODE=4
12. NRANKS=$((NNODES * RANKS_PER_NODE))
13. echo "NUM_OF_NODES=${NNODES} TOTAL_NUM_RANKS=${NRANKS} RANKS_PER_NODE=${RANKS_PER_NODE}"
14. export MPIR_CVAR_CH4_XPMEM_ENABLE=0
15. echo MPIR_CVAR_CH4_XPMEM_ENABLE=0
16. export MPIR_CVAR_ENABLE_GPU=0
17. echo MPIR_CVAR_ENABLE_GPU=0
18. LIST=4:32:54:68
19. mpiexec --hostfile $PBS_O_WORKDIR/nodelist.sorted.txt --np ${NRANKS} -ppn ${RANKS_PER_NODE} \
    --cpu-bind verbose,list:$LIST ./hello

Use explicit placement and verbose

```
mpiexec --np 8 -ppn 8 --cpu-bind verbose,list:0:1:2:3:52:53:54:55 <exe_file>
```

```
cpubind:list x1922c7s4b0n0 pid 17532 rank 0 0: mask 0x00000001
cpubind:list x1922c7s4b0n0 pid 17533 rank 1 1: mask 0x00000002
cpubind:list x1922c7s4b0n0 pid 17534 rank 2 2: mask 0x00000004
cpubind:list x1922c7s4b0n0 pid 17535 rank 3 3: mask 0x00000008
cpubind:list x1922c7s4b0n0 pid 17536 rank 4 4: mask 0x00100000,0x0
cpubind:list x1922c7s4b0n0 pid 17537 rank 5 5: mask 0x00200000,0x0
cpubind:list x1922c7s4b0n0 pid 17538 rank 6 6: mask 0x00400000,0x0
cpubind:list x1922c7s4b0n0 pid 17539 rank 7 7: mask 0x00800000,0x0
```

Key points:

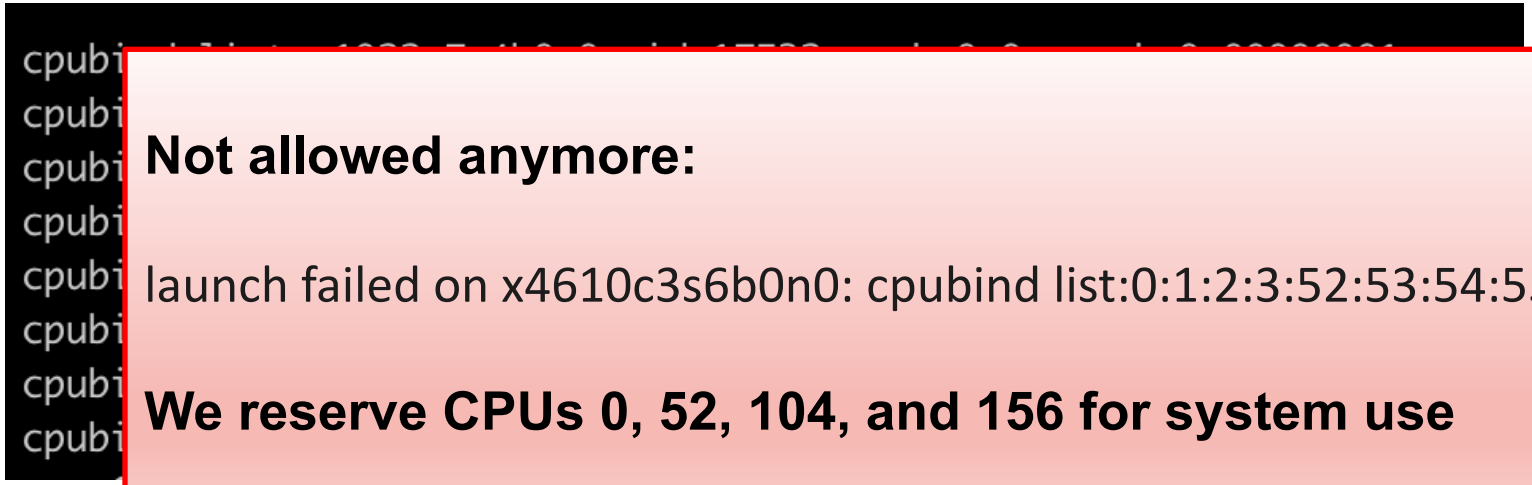
Check the placement

Do not rely on defaults

- ✓ Mask represents cores a task is assigned to. Cores are numerated from 0 to 207
- ✓ Mask is hexadecimal, numerating cores right-to-left
- ✓ One mask digit represents 4 cores, from 0000_b (no cores) to F = 1111_b (all 4 cores)
- ✓ Mask 0x00000001 is 1_b, represents Core0
- ✓ Mask 0x00000002 is 10_b, represents Core1
- ✓ Obviously, Mask 0x00000003 is 11_b, represents Core0 and Core1
- ✓ 0x0 means 0x00000000 no cores from 32 core pool. 0xFFFFFFFF is all 32 cores
- ✓ Example: 0x105,,0x5 means: cores 101b from first 32 core pool, 32 cores empty, cores 1_b 0000_b 0101_b from third pool or Cores 0, 2, 65, 67, 73

Use explicit placement and verbose

```
mpiexec --np 8 -ppn 8 --cpu-bind verbose,list:0:1:2:3:52:53:54:55 <exe_file>
```



Not allowed anymore:

launch failed on x4610c3s6b0n0: cpubind list:0:1:2:3:52:53:54:55[0] has fewer CPUs (0) than depth (1)

We reserve CPUs 0, 52, 104, and 156 for system use

- ✓ Mask represents cores a task is assigned to. Cores are numerated from 0 to 207
- ✓ Mask is hexadecimal, numerating cores right-to-left
- ✓ One mask digit represents 4 cores, from 0000_b (no cores) to F = 1111_b (all 4 cores)
- ✓ Mask 0x00000001 is 1_b, represents Core0
- ✓ Mask 0x00000002 is 10_b, represents Core1
- ✓ Obviously, Mask 0x00000003 is 11_b, represents Core0 and Core1
- ✓ 0x0 means 0x00000000 no cores from 32 core pool. 0xFFFFFFFF is all 32 cores
- ✓ Example: 0x105,,0x5 means: cores 101b from first 32 core pool, 32 cores empty, cores 1_b 0000_b 0101_b from third pool or Cores 0, 2, 65, 67, 73

GPU-aware MPI - default

Programmer can use pointers to GPU device memory in MPI buffers

convenience, but there is a price - performance

GPU-aware MPI - default

```
1 #include <mpi.h>
2 #include <omp.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main( int argc, char * argv[] )
7 {
8     int rank, numdevices, hostid, deviceid;
9     double *val_host, *val_device;
10
11    MPI_Init( &argc, &argv );
12    MPI_Comm_rank( MPI_COMM_WORLD, &rank ) ;
13    hostid = omp_get_initial_device();
14    numdevices = omp_get_num_devices();
15    deviceid = rank % numdevices;
16    omp_set_default_device( deviceid );
17
18    val_host   = (double *)malloc( sizeof( double ) );
19    val_device = (double *)omp_target_alloc( sizeof( double ), deviceid );
20
21    if ( rank == 0 )
22    {
23        *val_host = 15.0; printf( "%3d: My GPUid %1d, I will broadcast value %8.4lf\n", deviceid, rank, *val_host );
24    }
25    else
26    {
27        *val_host = -1.0; printf( "%3d: My GPUid %1d, My initial value was   %8.4lf\n", deviceid, rank, *val_host );
28    }
29
30    omp_target_memcpy( (void *)val_device, (const void *)val_host, sizeof( double ), 0, 0, deviceid, hostid );
31    MPI_Bcast( val_device, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD);
32    omp_target_memcpy( (void *)val_host, (const void *)val_device, sizeof( double ), 0, 0, hostid, deviceid );
33
34    printf( "%3d: I received %8.4lf\n", rank, *val_host );
35
36    free( val_host);
37    omp_target_free( val_device, deviceid );
38    MPI_Finalize();
39    return 0;
40 }
41
```

mory in MPI buffers

val_device is in GPU memory

MPI_Bcast uses it directly

GPU-aware MPI - default

Programmer can use pointers to GPU device memory in MPI buffers

convenience, but there is a price - performance

If you do not use GPU device memory in MPI buffers, unset GPU-awareness

```
export MPI_CVAR_ENABLE_GPU=0
```

This helps to improve latency of MPI operations

Know the NIC assignments

- ✓ 4 NICs on a socket, 8 NICs on a node
- ✓ Round-robin MPI process to NIC assignment on a socket
- ✓ MPI process does not use multiple NICs*

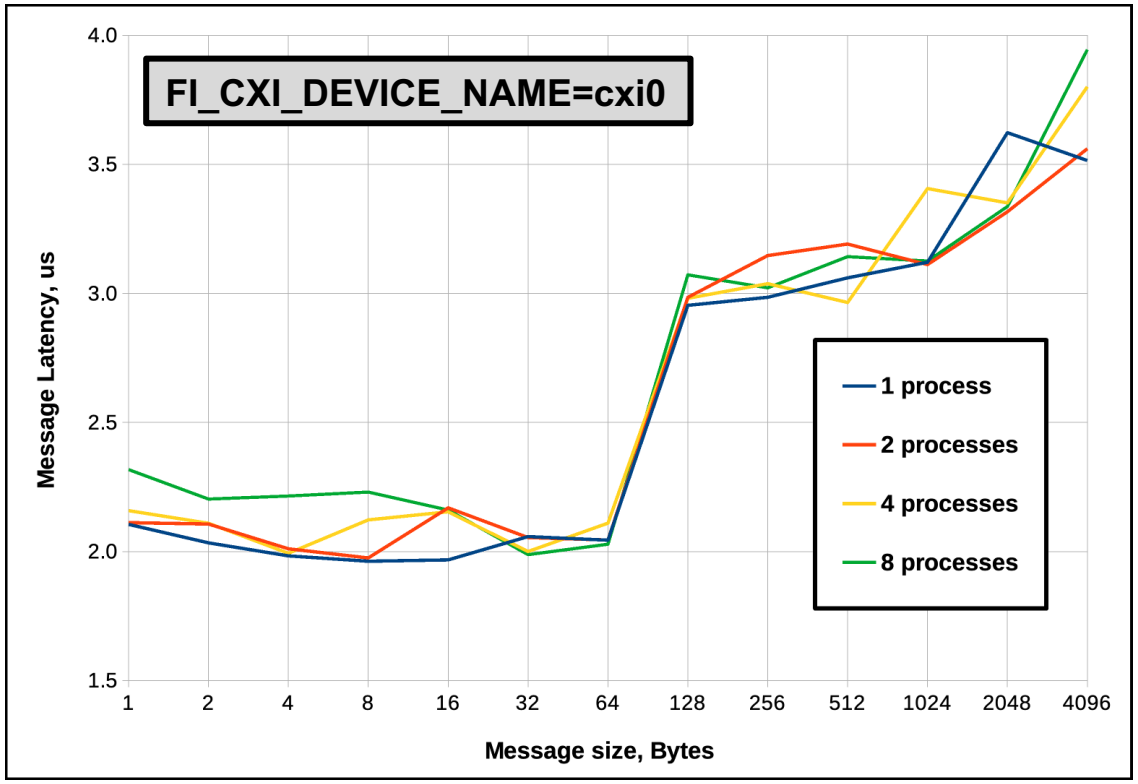
  Link bandwidth might be a limitation
- ✓ MPI processes may share the NICs

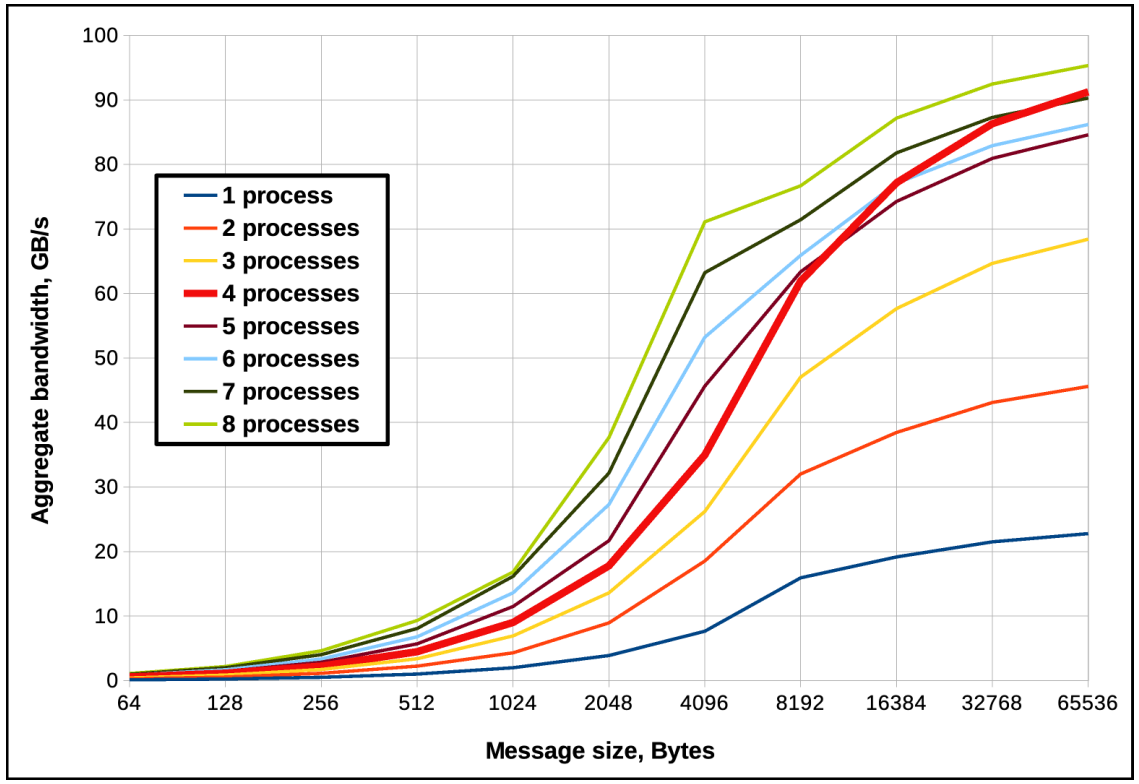
  Running 5 or more processes per socket
- ✓ Sockets are connected by UPI bus – might be a limitation
- ✓ Cores are attached to a Network-on-a-chip – might be a limitation

Key points: Distribute processes across cores and sockets to maximize hardware utilization

*** Default, may be changed in experimental builds**

Maximizing NICs utilization – host memory



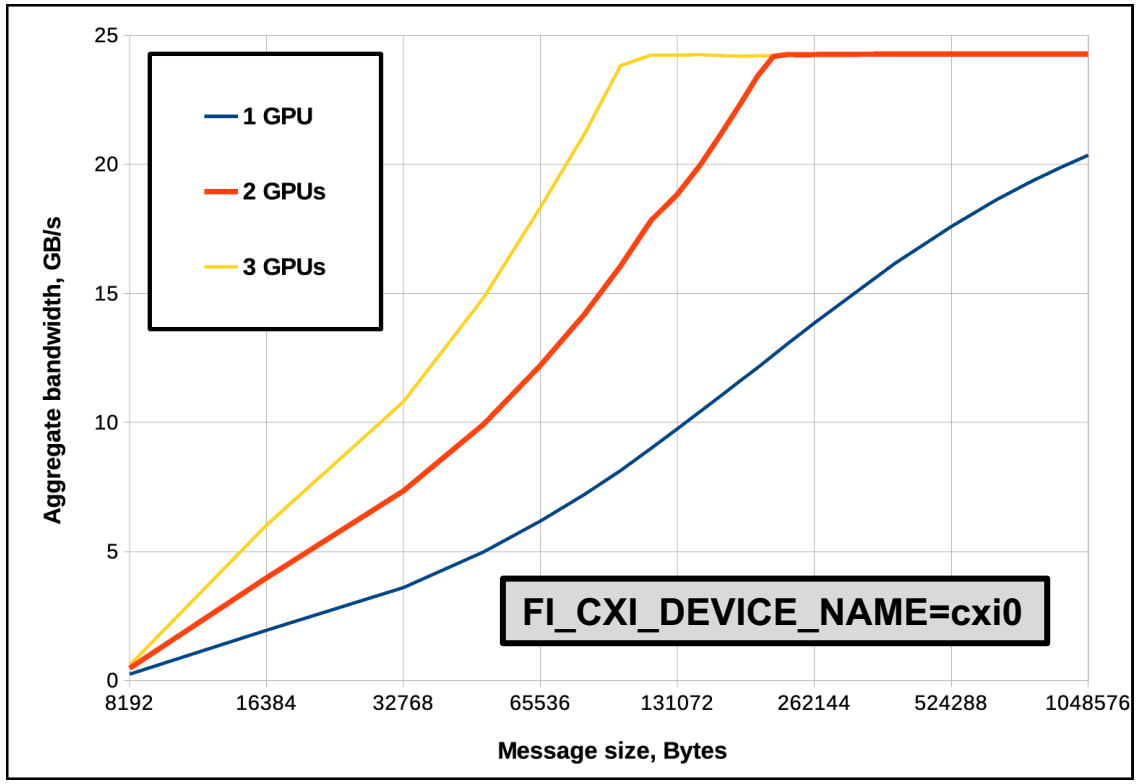


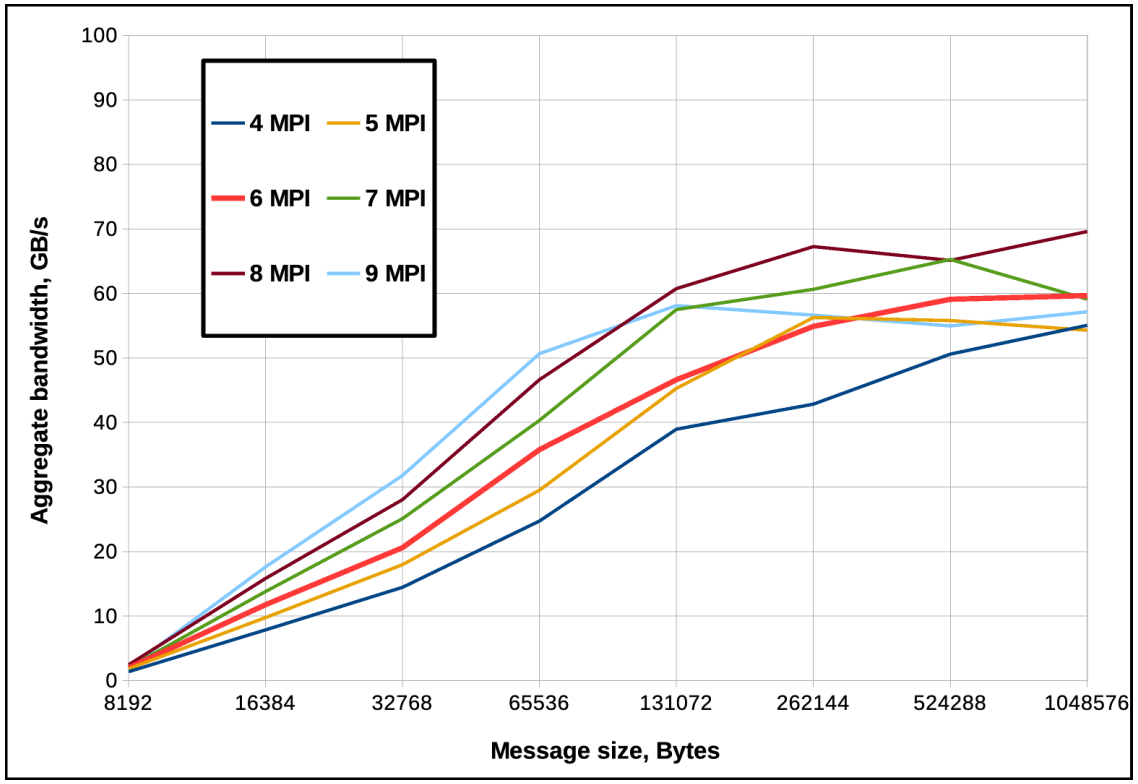
Multiple processes sharing a NIC

Adding processes using more NICs

Key points: Sharing NICs when underutilizing

Maximizing NICs utilization – device memory





Multiple processes sharing a NIC

Adding processes using more NICs

Red line – popular 12 MPI processes per node

Key points: Sharing NICs when underutilizing

Know the memory domains – Default Flat mode

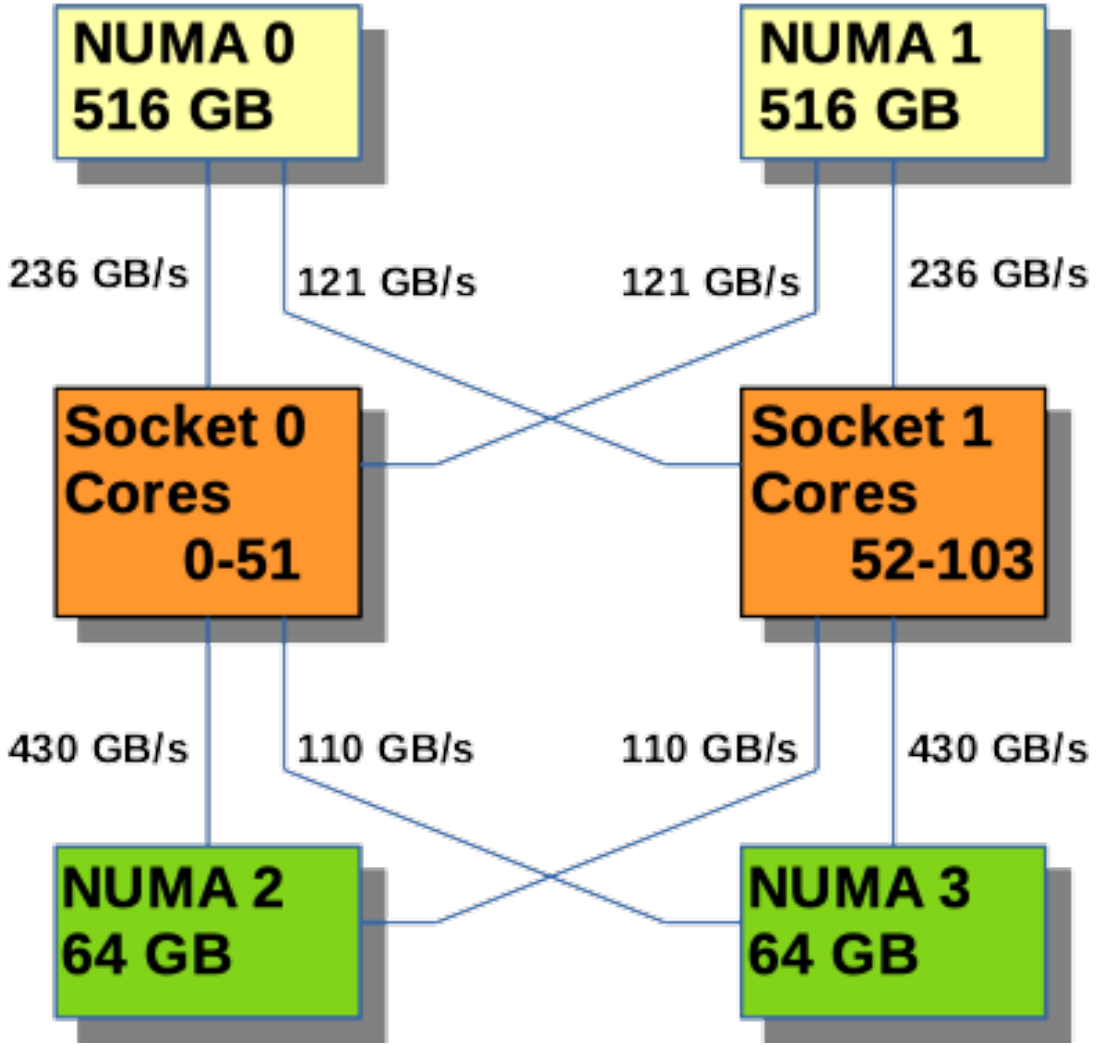
```
morozov@uan-0002:~/ALCF-Benchmark/NUMA> cat NUMA.out
Welcome to sunspot.alcf.anl.gov
module restore
module list
Working directory is /home/morozov/ALCF-Benchmark/NUMA
Jobid: 1341028.amn-0001
Running on host x1922c0s4b0n0
Running on nodes x1922c0s4b0n0.hostmgmt2001.cm.americas.sgi.com
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 \
    32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 104 105 106 107 108 109 110 111 112\
    113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136\
    137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155
node 0 size: 515527 MB
node 0 free: 510000 MB
node 1 cpus: 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80\
    81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 156 157 158 159 160 161\
    162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 \
    186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
node 1 size: 515005 MB
node 1 free: 511001 MB
node 2 cpus:
node 2 size: 65536 MB
node 2 free: 65432 MB
node 3 cpus:
node 3 size: 65536 MB
node 3 free: 65434 MB
node distances:
node   0   1   2   3
  0:  10  21  13  23
  1:  21  10  23  13
  2:  13  23  10  23
  3:  23  13  23  10
```

- ✓ numactl -H before mpiexec
- ✓ 512GB DDR per socket
- ✓ 64GB HBM nodes
- ✓ NUMA domains defined by mode
- ✓ DDR nodes have cores
- ✓ numactl -m 2-3 ./app
- ✓ libnuma, umf, memkind, or similar

```
#include <hbwmalloc.h>
void *hbwmalloc( size_t );
void hbw_free( void *ptr );
-I/home/morozov/include
-L/home/morozov/lib -lmemkind
```

Key points: Meet the system config with your application config

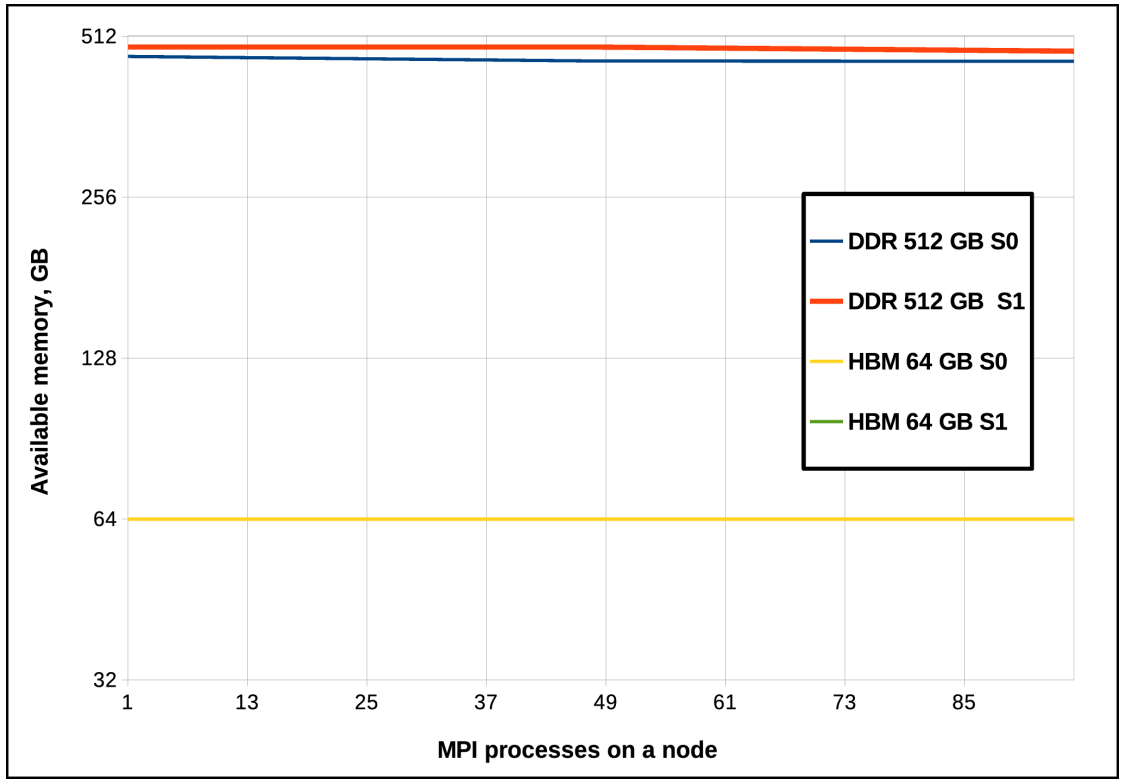
Know the memory domains – Flat mode



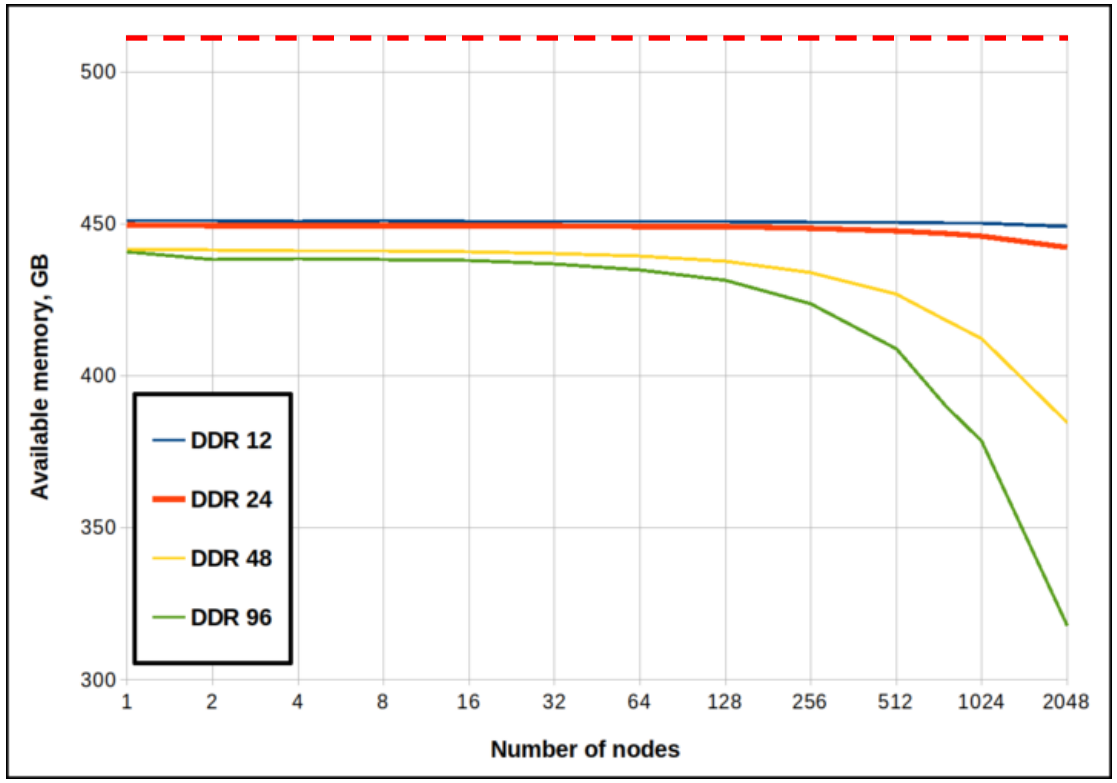
- ✓ numactl -H before mpiexec
- ✓ 512GB DDR per socket
- ✓ 64GB HBM nodes
- ✓ NUMA domains defined by mode
- ✓ DDR nodes have cores
- ✓ numactl -m 2-3 ./app
- ✓ libnuma, umf, memkind

Key points: Meet the system config with your application config

Balance concurrency and available memory



Single Node



Multinode Environment

Key point: Memory is used by runtime

Measured with mpich/opt/4.3.0rc3, libfabric/1.20.1
Might change in future

Local Environment

- ✓ PALS-defined variables – Cray's Parallel Application Launch service provided

  PALS_RANKID – job process ID, PALS_LOCAL_RANKID – node process ID

```
char *ptr = getenv("PALS_LOCAL_RANKID");
if ( ptr != NULL ) local_id = atoi( ptr );
```

- ✓ MPI standard for shared memory region

  MPI_COMM_TYPE_SHARED predefined

```
MPI_Comm_split_type( MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, &shmcomm );
MPI_Comm_size( shmcomm, &shmsize );    // Number of ranks per node
MPI_Comm_rank( shmcomm, &shmrank );    // My ID in a node, might be similar to PALS_LOCAL_RANKID
```

Key point: Use local environment setup constructs to identify itself

OMP example: Assigning a device

```
#include <mpi.h>
#include <omp.h>
#include <stdio.h>
int main( int argc, char * argv[] ) {
    int rank, nsize, res_len, shmrank, shmsize, numdevices, deviceid;
    char name[ MPI_MAX_PROCESSOR_NAME ];    MPI_Comm shmcomm;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &nsize );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank ) ;
    MPI_Get_processor_name( &name[0], &res_len );
    MPI_Comm_split_type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED,
        0, MPI_INFO_NULL, &shmcomm);
    MPI_Comm_size( shmcomm, &shmsize );   // Ranks per node
    MPI_Comm_rank( shmcomm, &shmrank );   // Local rank ID
    numdevices = omp_get_num_devices();
    deviceid = shmrank % numdevices;
    omp_set_default_device( deviceid );
    printf( "Hello from rank %3d of %4d, I am rank %3d of node ranks
        %3d, I know %2d devices, my device id %2d: node %s\n",
        rank, nsize, shmrank, shmsize, numdevices, deviceid, name );
    MPI_Finalize();
    return 0;
}
```

```
mpicc -O2 -g -fiopenmp -fopenmp-targets=spir64  -c hello.c -o hello.o
mpicc -O2 -g -fiopenmp -fopenmp-targets=spir64  hello.o -o hello
```

```
module restore
module list
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR
echo Jobid: $PBS_JOBID
echo Running on host `hostname`
echo Running on nodes `cat $PBS_NODEFILE`
NNODES=`wc -l < $PBS_NODEFILE`
NRANKS=4            # Number of MPI ranks per node
NTOTRANKS=$(( NNODES * NRANKS ))

echo "NUM_OF_NODES=${NNODES}  TOTAL_NUM_RANKS=${NTOTRANKS}
RANKS_PER_NODE=${NRANKS}

export LIBOMPTARGET_DEVICES=SUBDEVICE  # DEVICE | SUBDEVICE
export LIBOMPTARGET_PLUGIN=LEVEL0      # DEFAULT

mpiexec --np ${NTOTRANKS} -ppn ${NRANKS} \
    --cpu-bind=verbose,list:2:34:54:86 ./hello
```

WARNING: LIBOMPTARGET_DEVICES=SUBDEVICE – not portable

Example: MPI process to GPU binding

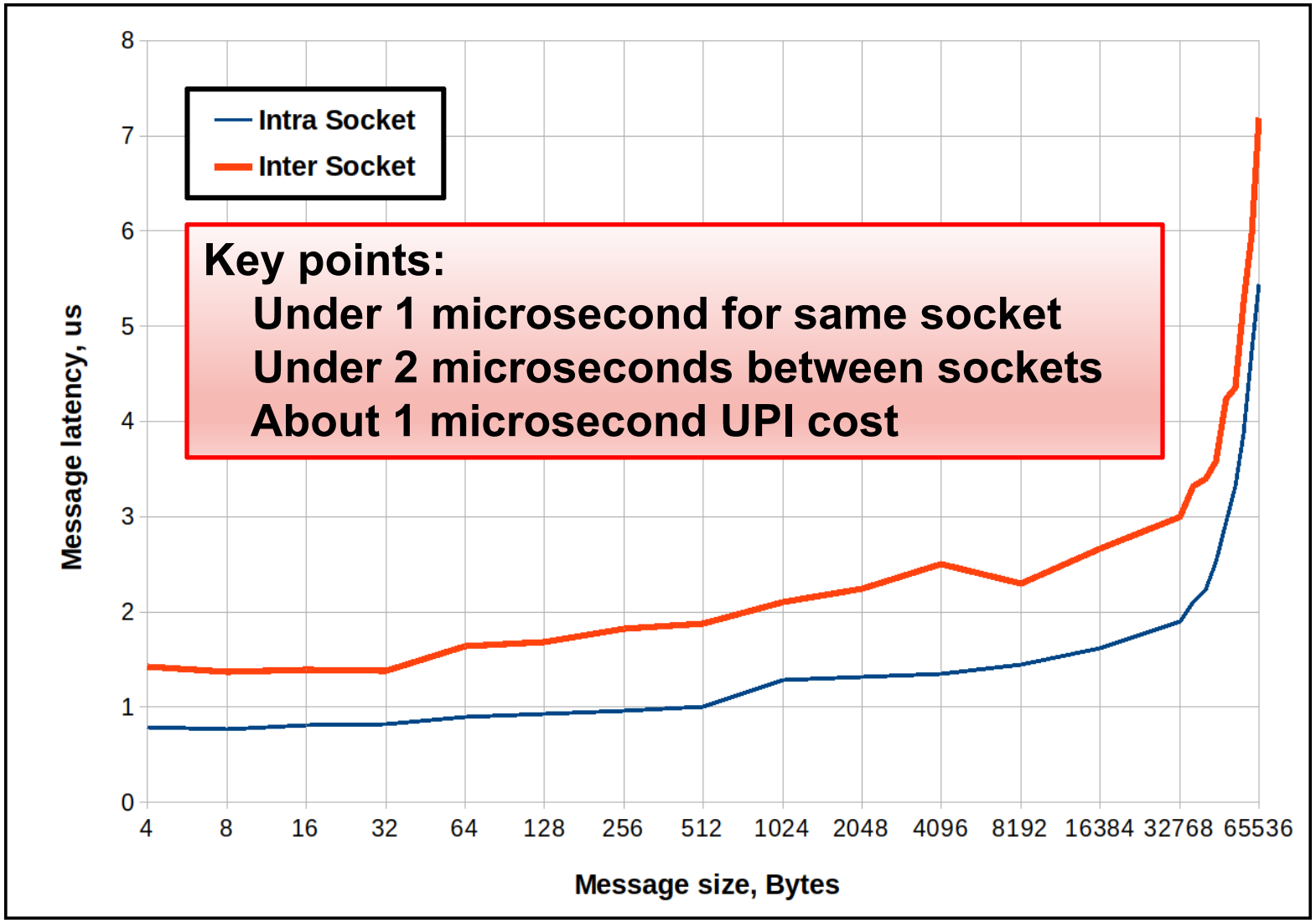
The "official" way is to use gpu_tile_compact.sh – see Thomas Applencourt for details.

```
morozov@uan-0002:~> cat set_ze_mask.sh
#!/bin/bash

rpn=${RANKS_PER_NODE}
ranks_per_tile=${RANKS_PER_TILE}

##### 0.0 0.1 0.0 0.1 until full 1.0 1.1 1.0 1.0 until full
##### My GPU first, tiles alternate
((g=PALS_LOCAL_RANKID/2/ranks_per_tile))
((t=PALS_LOCAL_RANKID%2))
export ZE_AFFINITY_MASK=$g.$t

echo "[I am rank $PALS_RANKID on node `hostname`]  Localrank=$PALS_LOCAL_RANKID, ZE_AFFINITY_MASK=$ZE_AFFINITY_MASK"

# Launch the executable:
$*
```

```
mpiexec --np ${NRANKS} -ppn ${RANKS_PER_NODE} --cpu-bind verbose,list:${ranks} \
./set_ze_mask.sh \
./<exe_name> <exe_ards>
```

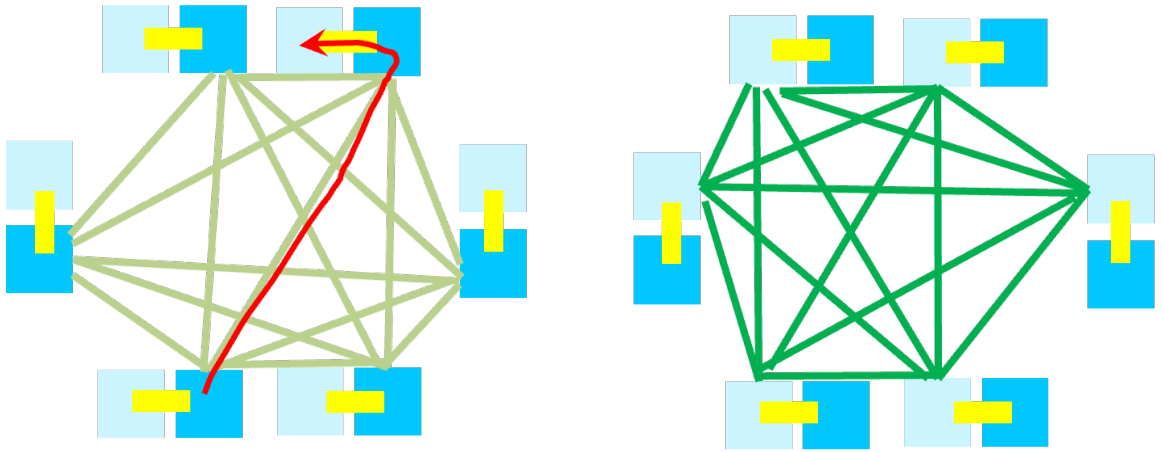
Key point:
Each task should only see the device it uses

Benchmarks: DDR Socket to Socket on a node



Benchmarks: GPU to GPU on a node

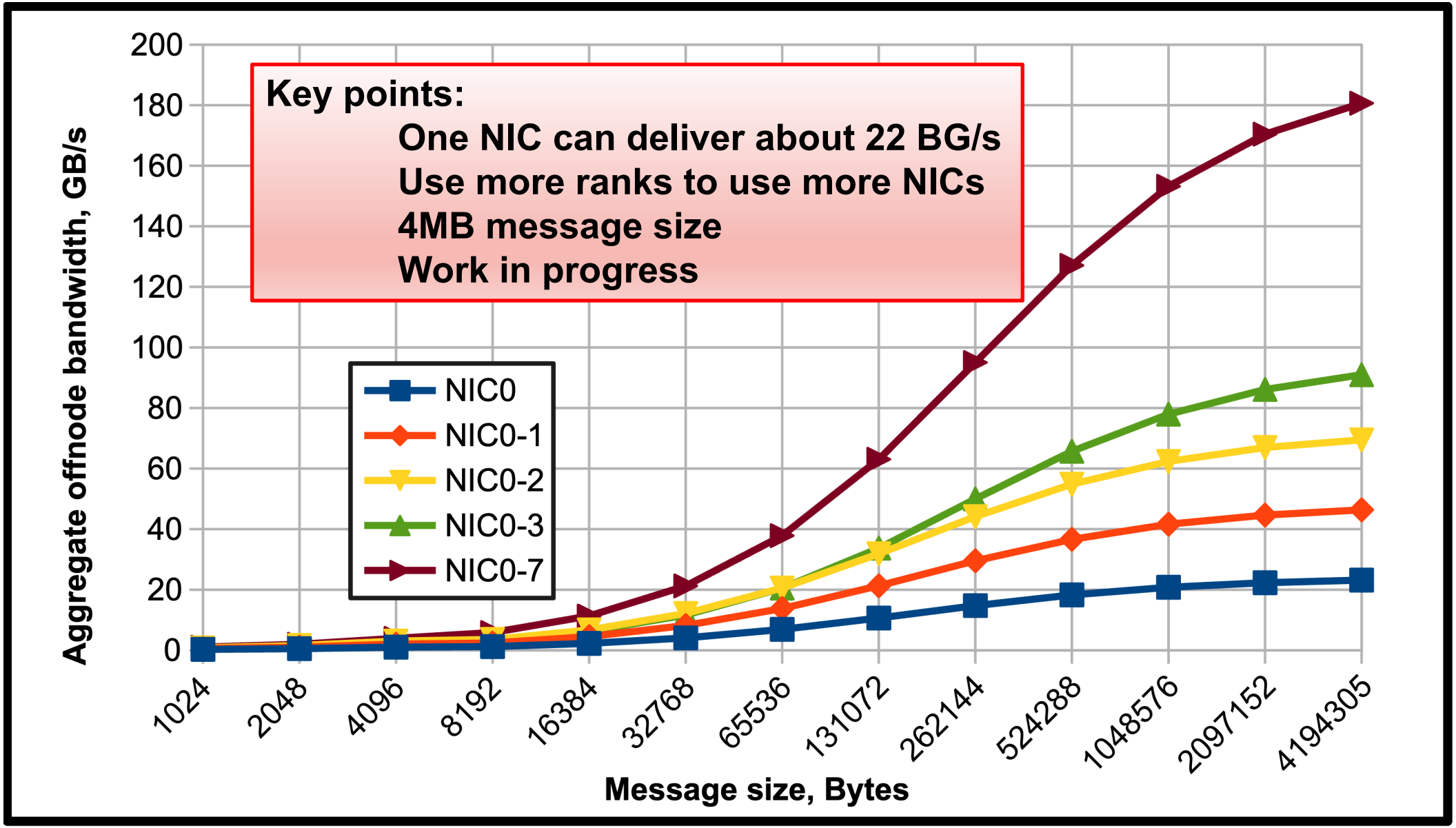
```
Welcome to aurora.alcf.anl.gov
Working directory is /home/morozov/OSU_MPI_IntelZE-runs/OSU_pt2pt_1n_bw.Z
Jobid: 4394.amn-0001
Running on host x1921c0s6b0n0

Intra-node: Core 0-25 device 0 -> Core 26-33 device 1
Rank 0/2, device_id = 0/6
Rank 1/2, device_id = 1/6
# OSU MPI-ZE Bandwidth Test v5.6.2
# Send Buffer on DEVICE (D) and Receive Buffer on DEVICE (D)
# Size      Bandwidth (MB/s)
1024                 49.83
2048                174.80
4096                343.54
8192                678.89
16384              1345.91
32768              2505.71
65536              4465.87
131072             7310.41
262144            10664.40
524288            13929.75
1048576           16370.99
2097152           17954.20
4194304           18874.38
8388608           19370.02
16777216          19436.62
33554432          19273.23
```



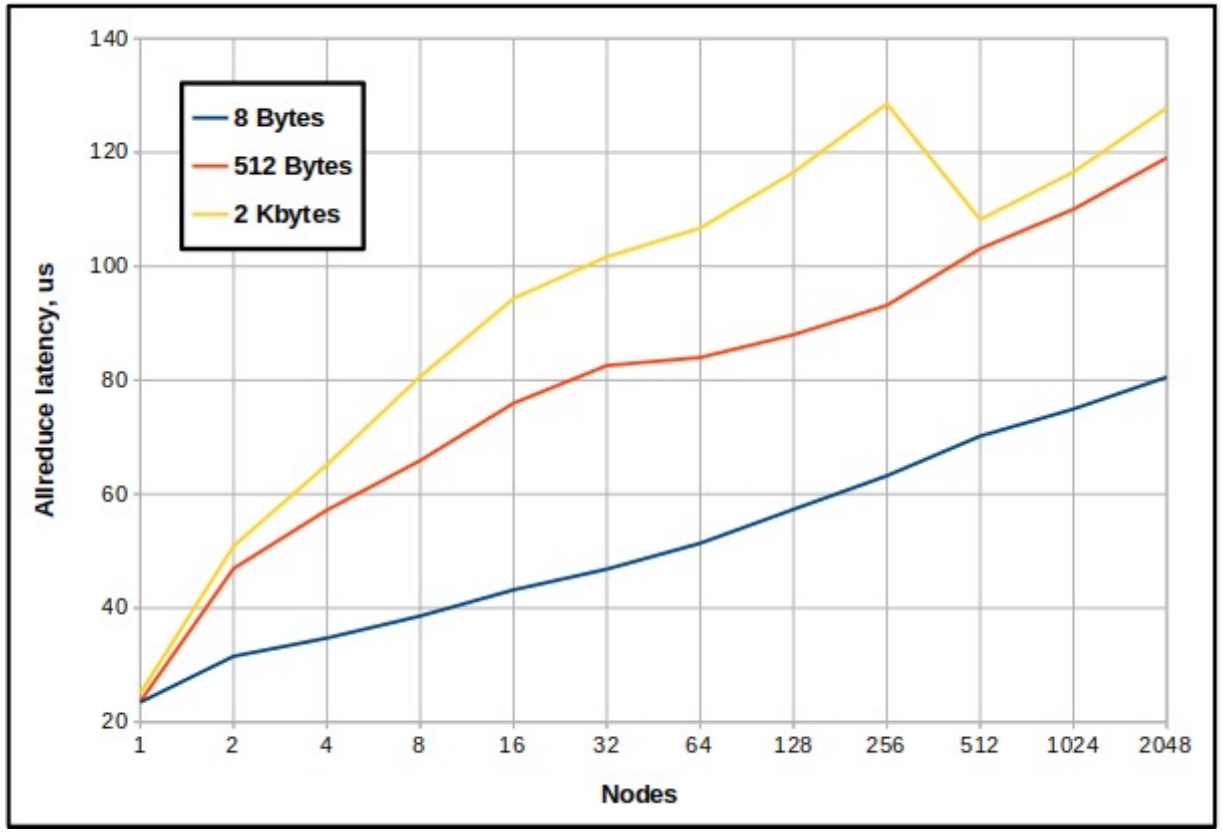
Key points:

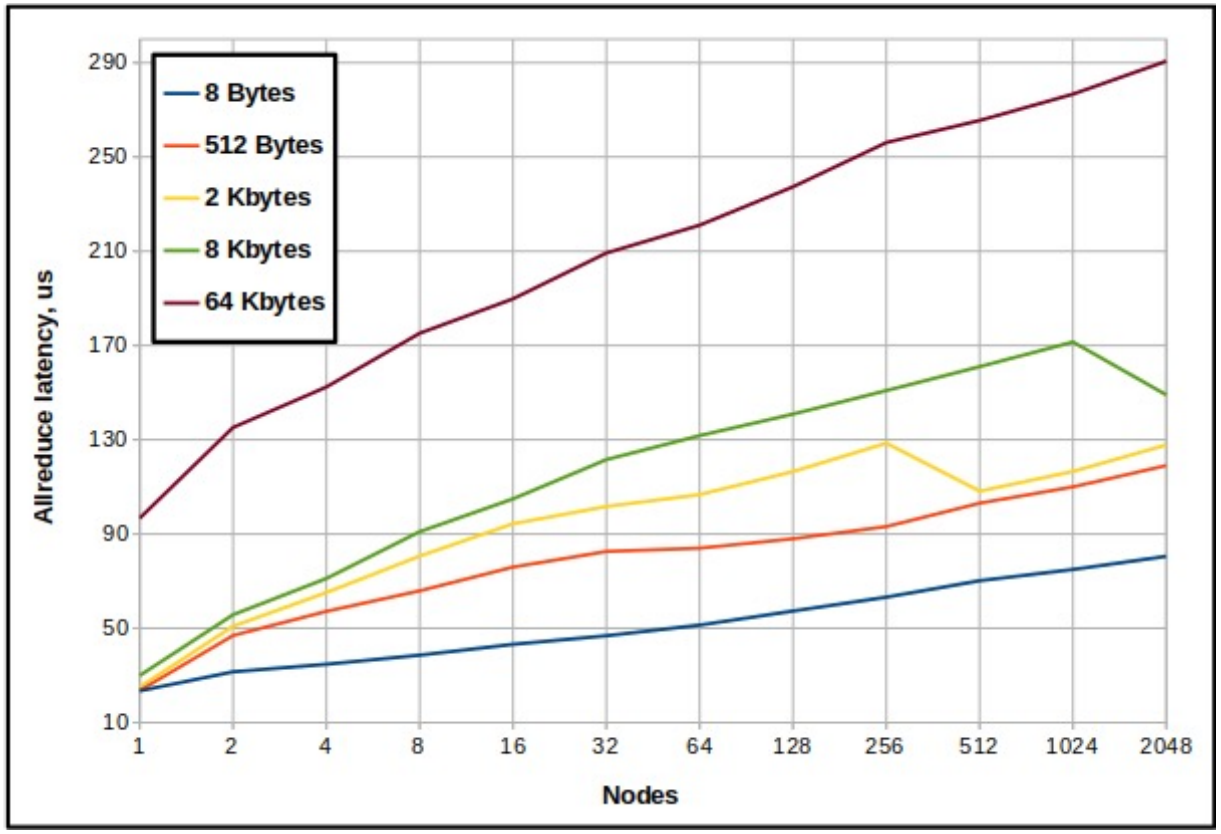
- About 20 GB/s/link
- All Links can be used

Benchmarks: Aggregate offnode bandwidth



Benchmarks: Allreduce latency from GPU buffers





Key points:

- 1 MPI process per GPU
- Many optimization options
- Topology-aware
- Mapping improves

Aurora-Sunspot Bob Walkup's MPI profiler

```
Link:  -L/home/morozov/mpitrace/hpmprof -lhpmprof_c \
       -L/home/morozov/binutils-2.39/lib -lbfd -liberty \
       -L/home/morozov/zlib-1.2.13/lib -lz
```

```
Data for MPI rank 0 of 192:
Times and statistics from MPI_Init() to MPI_Finalize().
-----------------------------------------------------------------
MPI Routine                        #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_rank                          10            0.0          0.000
MPI_Comm_size                           8            0.0          0.000
MPI_Isend                            1092     16424526.9          0.025
MPI_Recv                               52      2508532.2          0.095
MPI_Irecv                            1040     17119641.6          0.027
MPI_Wait                             1456            0.0         10.786
MPI_Waitall                           312            0.0          7.634
MPI_Barrier                            79            0.0          2.338
MPI_Allreduce                          87          373.9          5.652
-----------------------------------------------------------------
total communication time = 26.556 seconds.
total elapsed time       = 150.842 seconds.
user cpu time            = 140.157 seconds.
system time              = 11.229 seconds.
max resident set size    = 1739.879 MBytes.
```

Key points:
Familiar tools are ported

Final checklist

- ✓ If you are happy with the results, continue scaling to bigger hardware
- ✓ Socket: 52 cores, 3 devices, 4 NICs, Memory domains – complete system
  - ➢ Maximize resource utilization, balance, minimize sharing
- ✓ Mapping: ranks to cores, devices to ranks
  - ➢ `mpiexec …-ppn 3  --cpu-bind verbose,list:1-15:16-31:32-51` (1 NIC is unused)
  - ➢ `mpiexec …-ppn 12 --cpu-bind list:4:8:12:32:36:40:48:50:51`
  - ➢ Check your program API to expose GPUs

```
if ((PALS_LOCAL_RANKID==3)); then
    export ZE_AFFINITY_MASK=3
fi
```

- ✓ Always make a sanity check
- ✓ Use mpich/dbg for debugging

Key point:
Search for info: slack, support@alcf.anl.gov