Hewlett Packard Enterprise

Advanced Placement

Comprehensive General LUMI Course

April 23–26, 2024

Goal: Speed Up Your Application Without Changing or Recompiling Code (1/3)

“Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway”

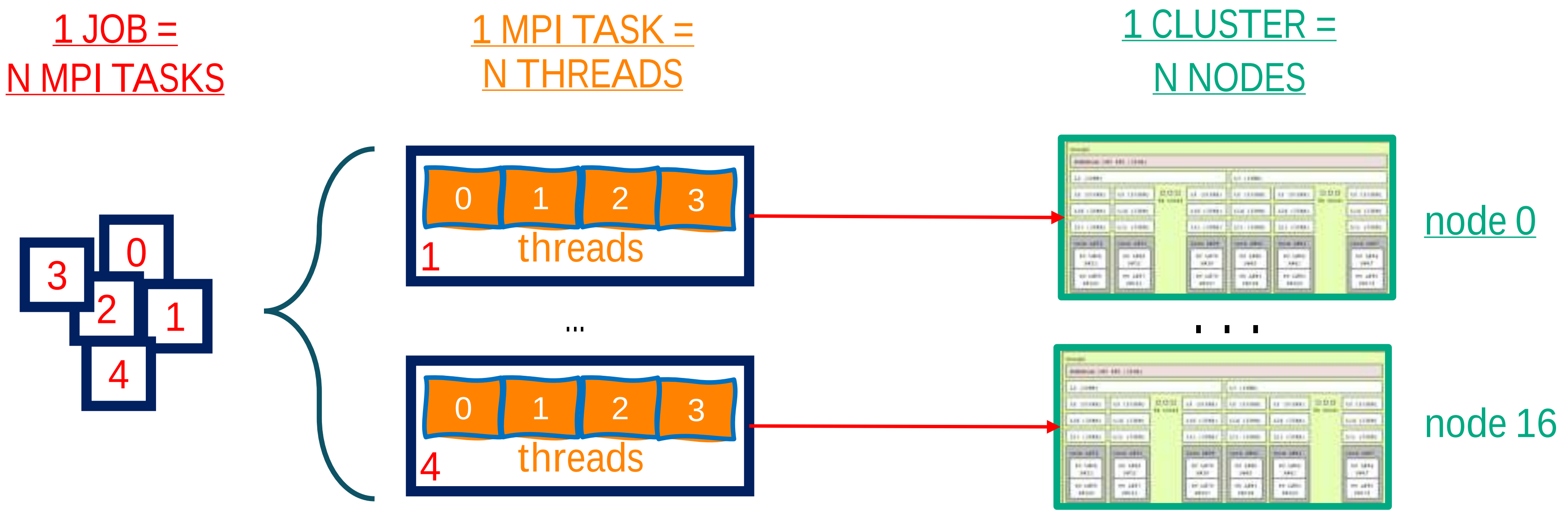
— Andrew S. Tanenbaum

Extending the metaphor, this talk is about configuring a “GPS” to:

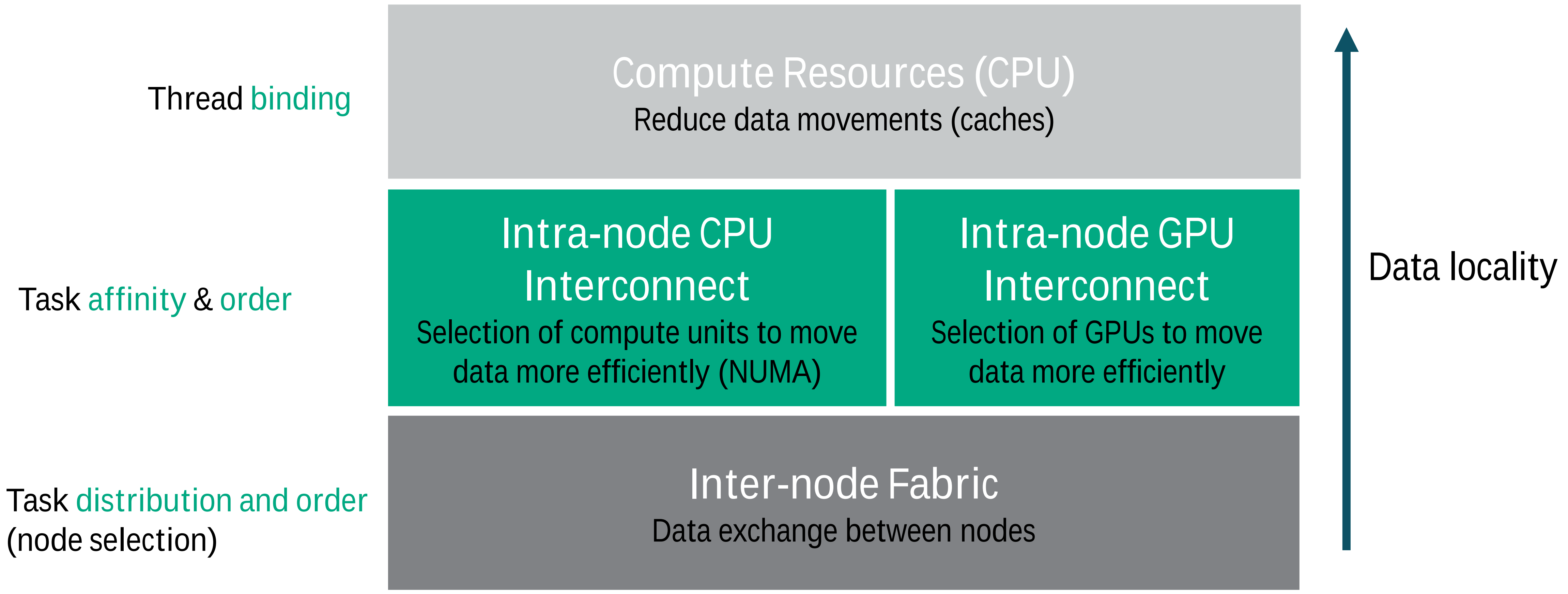
1. Use the fastest routes from point A to point B (reduce latency)
2. Avoid traffic jams (hitting hardware limitation due to link saturation)

Goal: Speed Up Your Application Without Changing or Recompiling Code (2/3)

- Task distribution (and ordering): distribution of MPI tasks among the nodes.
- Task affinity: assign each MPI task a set of CPU cores.
- Thread binding: pin/attach each thread in each MPI task to one or many cores.



Goal: Speed Up Your Application Without Changing or Recompiling Code (3/3)


Thread binding
Compute Resources (CPU)
Reduce data movements (caches)
Task affinity & order
Intra-node CPU Interconnect
Selection of compute units to move data more efficiently (NUMA)
Intra-node GPU Interconnect
Selection of GPUs to move data more efficiently
Task distribution and order (node selection)
Inter-node Fabric
Data exchange between nodes
Data locality

Agenda

- Placement Check tools
- Inter-node Placement (fabric)
- Intra-node Placement (memory affinity)
- Compute Resources Affinity (OpenMP thread binding)
- Intra-node Placement for GPUs

PLACEMENT CHECK TOOLS

Tools and options to check distributions and bindings

PLACEMENT CHECK TOOLS – hybrid_check

```
module load LUMI/23.09
module load partition/C
module load lumi-CPEtools
srun -c2 --nodes=1 --tasks-per-node=8 hybrid_check
```

```
Running 8 MPI ranks with 2 threads each (total number of threads: 16).

++ hybrid_check: MPI rank   0/8   OpenMP thread   0/2   on cpu   0/256 of nid002052
++ hybrid_check: MPI rank   0/8   OpenMP thread   1/2   on cpu   1/256 of nid002052
++ hybrid_check: MPI rank   1/8   OpenMP thread   0/2   on cpu   2/256 of nid002052
++ hybrid_check: MPI rank   1/8   OpenMP thread   1/2   on cpu   3/256 of nid002052
++ hybrid_check: MPI rank   2/8   OpenMP thread   0/2   on cpu   4/256 of nid002052
++ hybrid_check: MPI rank   2/8   OpenMP thread   1/2   on cpu   5/256 of nid002052
++ hybrid_check: MPI rank   3/8   OpenMP thread   0/2   on cpu   6/256 of nid002052
++ hybrid_check: MPI rank   3/8   OpenMP thread   1/2   on cpu   7/256 of nid002052
++ hybrid_check: MPI rank   4/8   OpenMP thread   0/2   on cpu  64/256 of nid002052
++ hybrid_check: MPI rank   4/8   OpenMP thread   1/2   on cpu  65/256 of nid002052
++ hybrid_check: MPI rank   5/8   OpenMP thread   0/2   on cpu  67/256 of nid002052
++ hybrid_check: MPI rank   5/8   OpenMP thread   1/2   on cpu  66/256 of nid002052
++ hybrid_check: MPI rank   6/8   OpenMP thread   0/2   on cpu  68/256 of nid002052
++ hybrid_check: MPI rank   6/8   OpenMP thread   1/2   on cpu  69/256 of nid002052
++ hybrid_check: MPI rank   7/8   OpenMP thread   0/2   on cpu  70/256 of nid002052
++ hybrid_check: MPI rank   7/8   OpenMP thread   1/2   on cpu  71/256 of nid002052
```

PLACEMENT CHECK TOOLS – gpu_check

```
module load LUMI/23.09
module load partition/G
module load lumi-CPEtools
srun -n 8 -c 7 gpu_check
```

```
MPI 000 - OMP 000 - HWT 001 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 001 - HWT 002 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 002 - HWT 003 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 003 - HWT 004 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 004 - HWT 005 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 005 - HWT 006 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 000 - OMP 006 - HWT 007 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc

MPI 001 - OMP 000 - HWT 009 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 001 - OMP 001 - HWT 010 - Node nid005014 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
…
```

PLACEMENT CHECK TOOLS – Summary and Other Options

- You should be aware WHAT is doing the binding for you!
- Various software components may try do this for you
  - WLM (SLURM, ALPS, …)
  - MPI (MVAPICH, …)
  - Compiler (CCE,GNU,…)
  - OpenMP
  - Tools (numactl, taskset)

Cheat sheet

`ml lumi-CPEtools; srun … hybrid_check`	Show hybrid MPI/OpenMP placement
`ml lumi-CPEtools; srun … gpu_check`	Show hybrid MPI/OpenMP placement + GPU affinities
`srun --cpu-bind=verbose,<mode>`	Report task affinity set by Slurm
`export OMP_DISPLAY_AFFINITY=TRUE`	Show OpenMP affinities
`export MPICH_CPUMASK_DISPLAY=1`	Report MPI task affinities
`export MPICH_RANK_REORDER_DISPLAY=1`	Show MPI Rank reordering
`export MPICH_OFI_NIC_VERBOSE=2`	Show information about network interface selection

INTER-NODE PLACEMENT

Distributing tasks between nodes to better harness the fabric

INTER-NODE PLACEMENT – What can be done?

Selection of nodes (almost no control)

Slurm allocates sets of nodes which are roughly consecutive (likely to be in the same cabinet with a few nodes).

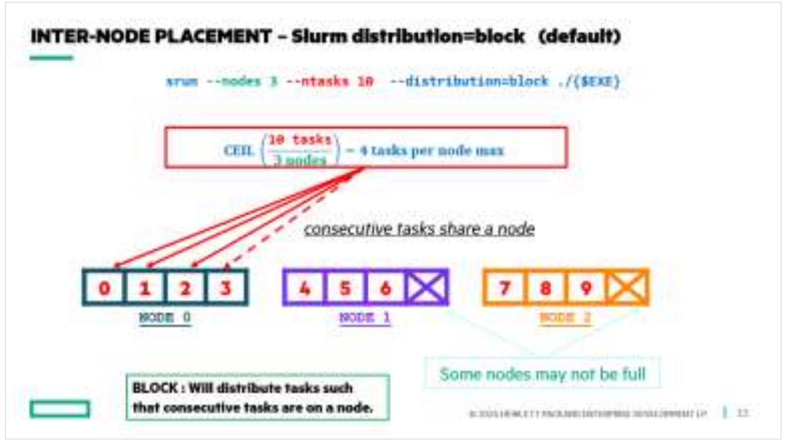
Task placement on allocated nodes

If possible, place ranks exchanging highest volume of data on the same compute. Two complementary options:

- Distribution pattern: Coarse grain control on placement with Slurm (srun `--distribution` argument).
- Reordering of MPI tasks (`MPICH_RANK_REORDER_METHOD`): More details during the *MPI rank reordering* talk and hints provided by the Perftools.

INTER-NODE PLACEMENT– Distributing tasks between nodes (coarse grain control)

- To control the distribution of the MPI ranks (tasks) across nodes
- In *srun* : use the `'--distribution/-m'` with either { block | cyclic | plane=<size> }


INTER-NODE PLACEMENT – Slurm distribution=block (default)

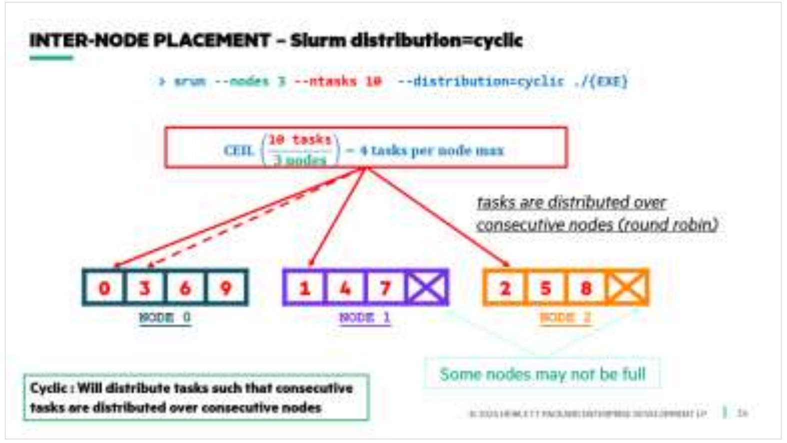
```
srun --nodes 3 --ntasks 10  --distribution=block ./($EXE)
```

$CEIL\left(\frac{10\ tasks}{3\ nodes}\right) = 4$ tasks per node max

consecutive tasks share a node

Some nodes may not be full

BLOCK : Will distribute tasks such that consecutive tasks are on a node.


INTER-NODE PLACEMENT – Slurm distribution=cyclic

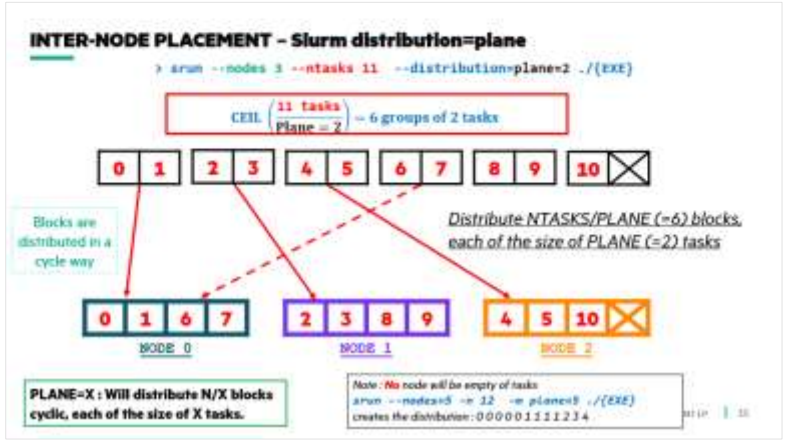
```
> srun --nodes 3 --ntasks 10  --distribution=cyclic ./(EXE)
```

$CEIL\left(\frac{10\ tasks}{3\ nodes}\right) = 4$ tasks per node max

tasks are distributed over consecutive nodes (round robin)

Some nodes may not be full

Cyclic : Will distribute tasks such that consecutive tasks are distributed over consecutive nodes


INTER-NODE PLACEMENT – Slurm distribution=plane

```
> srun --nodes 3 --ntasks 11  --distribution=plane=2 ./(EXE)
```

$CEIL\left(\frac{11\ tasks}{Plane = 2}\right) = 6$ groups of 2 tasks

Blocks are distributed in a cycle way

Distribute NTASKS/PLANE (=6) blocks, each of the size of PLANE (=2) tasks

PLANE=X : Will distribute N/X blocks cyclic, each of the size of X tasks.

Note : No node will be empty of tasks

```
srun --nodes=5 -n 12  -m plane=5 ./(EXE)
```

creates the distribution : 0 0 0 0 0 1 1 1 1 1 2 3 4

--distribution=block

(default)

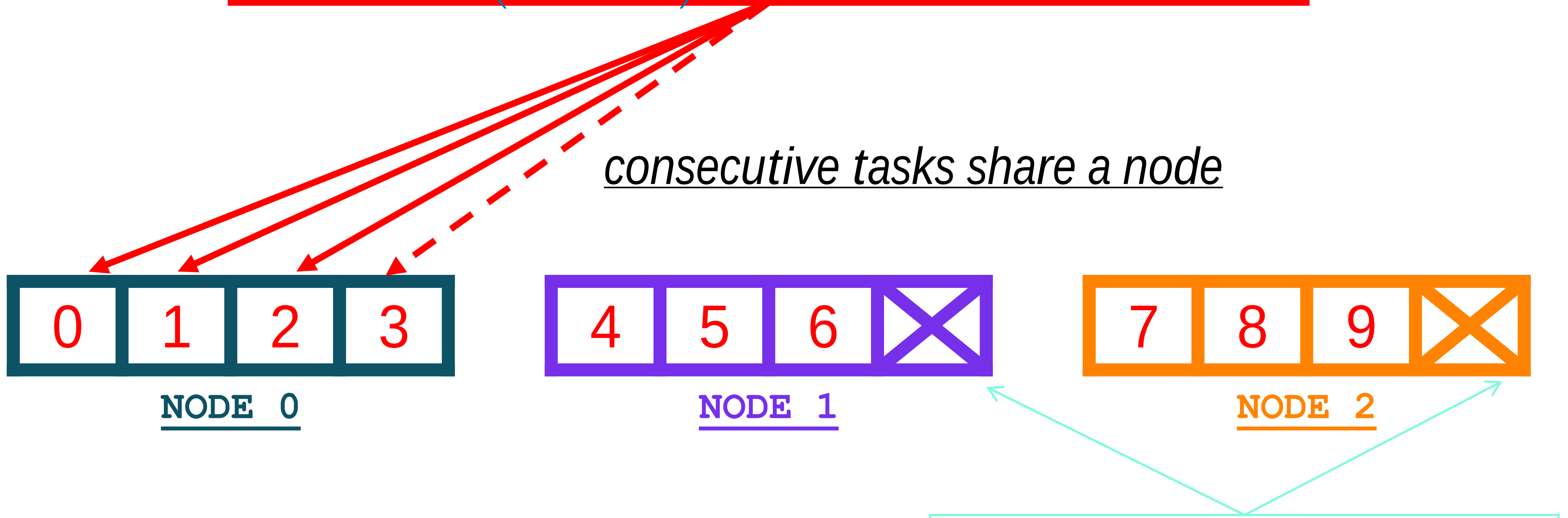
--distribution=cyclic

--distribution=plane=X

INTER-NODE PLACEMENT – Slurm distribution=block (default)

```
srun --nodes 3 --ntasks 10  --distribution=block ./{$EXE}
```

$$\text{CEIL}\left(\frac{\text{10 tasks}}{\text{3 nodes}}\right) = \text{4 tasks per node max}$$

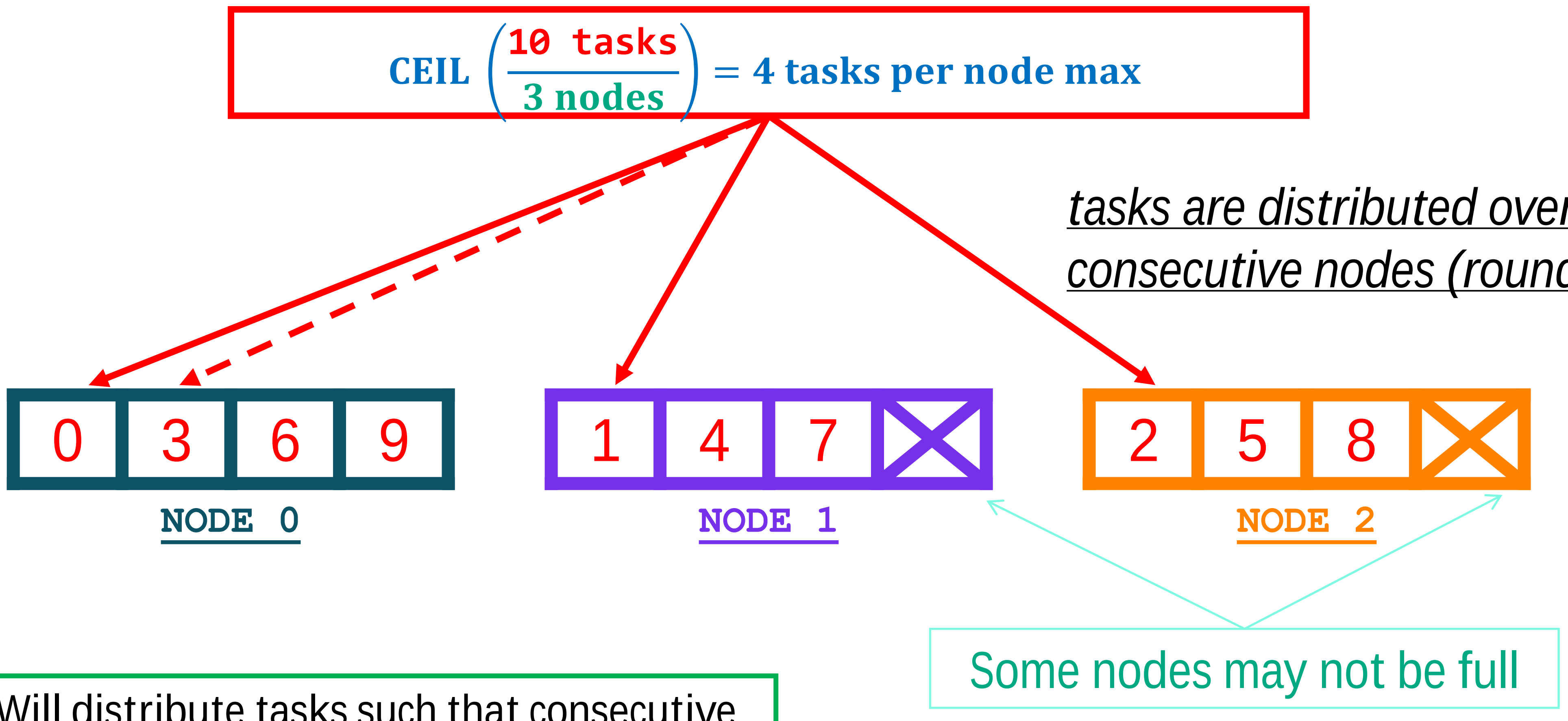


Some nodes may not be full

BLOCK : Will distribute tasks such that consecutive tasks are on a node.

INTER-NODE PLACEMENT – Slurm distribution=cyclic

```
> srun --nodes 3 --ntasks 10  --distribution=cyclic ./{EXE}
```


$\text{CEIL}\left(\frac{\text{10 tasks}}{\text{3 nodes}}\right) = \text{4 tasks per node max}$
0
3
6
9
NODE 0
1
4
7
NODE 1
2
5
8
NODE 2
Some nodes may not be full

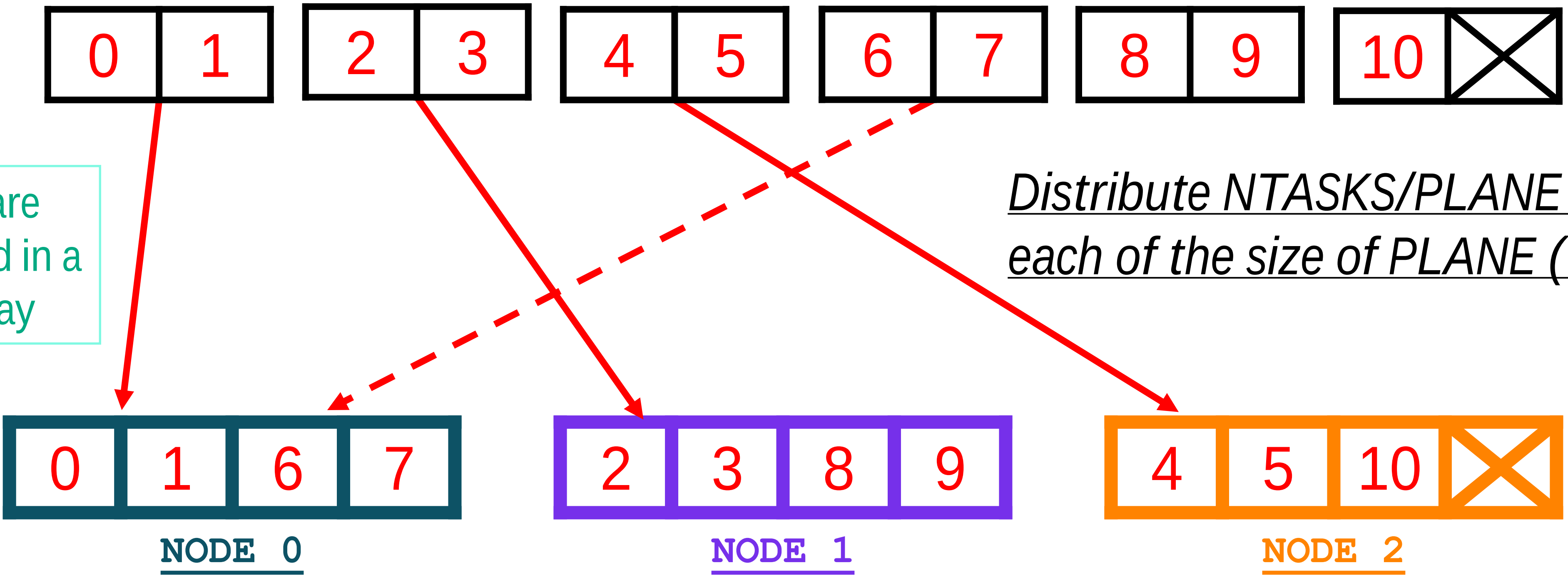
tasks are distributed over consecutive nodes (round robin)

Cyclic : Will distribute tasks such that consecutive tasks are distributed over consecutive nodes

INTER-NODE PLACEMENT – Slurm distribution=plane

```
> srun --nodes 3 --ntasks 11  --distribution=plane=2 ./{EXE}
```

$$\text{CEIL}\left(\frac{11\ \text{tasks}}{\text{Plane} = 2}\right) = 6\ \text{groups of 2 tasks}$$



Blocks are distributed in a cycle way

Distribute NTASKS/PLANE (=6) blocks, each of the size of PLANE (=2) tasks

PLANE=X : Will distribute N/X blocks cyclic, each of the size of X tasks.

Note : No node will be empty of tasks

```
srun --nodes=5 -n 12  -m plane=5 ./{EXE}
```

creates the distribution : 0 0 0 0 0 1 1 1 1 2 3 4

INTER-NODE PLACEMENT – Takeaways

- Keep in mind variations in network performance might be observed. Slurm may allocate nodes which are not in the same cabinet.

- Default Slurm distribution across nodes (block) is well suited in most cases.

- MPI rank reordering might be useful to reduce pressure on the fabric (perftools may provide hints)

Cheat sheet

`salloc/sbatch -x <nodelist>`	Exclude specific nodes
`srun --distribution=block|cyclic|plane=<X>`	Modify task distribution pattern across nodes

`export MPICH_RANK_REORDER_METHOD=3`
`export MPICH_RANK_REORDER_FILE=<file>`
	Manually defining rank reordering
	(more details in another presentation)

INTRA-NODE PLACEMENT (NUMA NODES)

Distributing tasks to better harness shared memory communications (node interconnect)

INTRA-NODE PLACEMENT – Lumi-G and Lumi-C Nodes

Accelerator blade

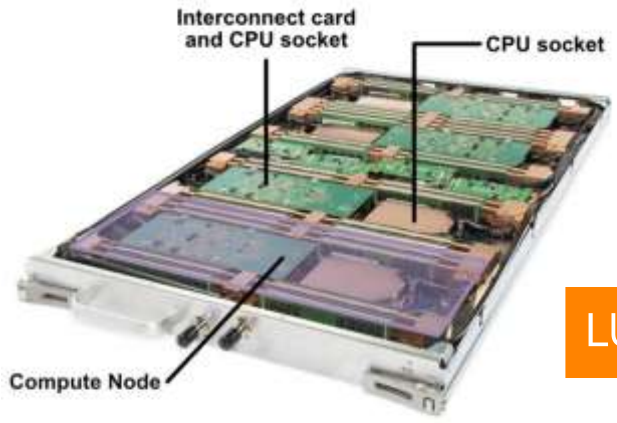
LUMI-G

Module GPU - HPE Cray EX235a

- Bard Peak Module
  - 2 nodes per module

- Each Bard Peak Node
  - 1 processor AMD Trento 64 cores
  - 4 GPUs AMD MI250x
  - 4 network interfaces Slingshot-11 (200Gbps each)

Compute blade



LUMI-C

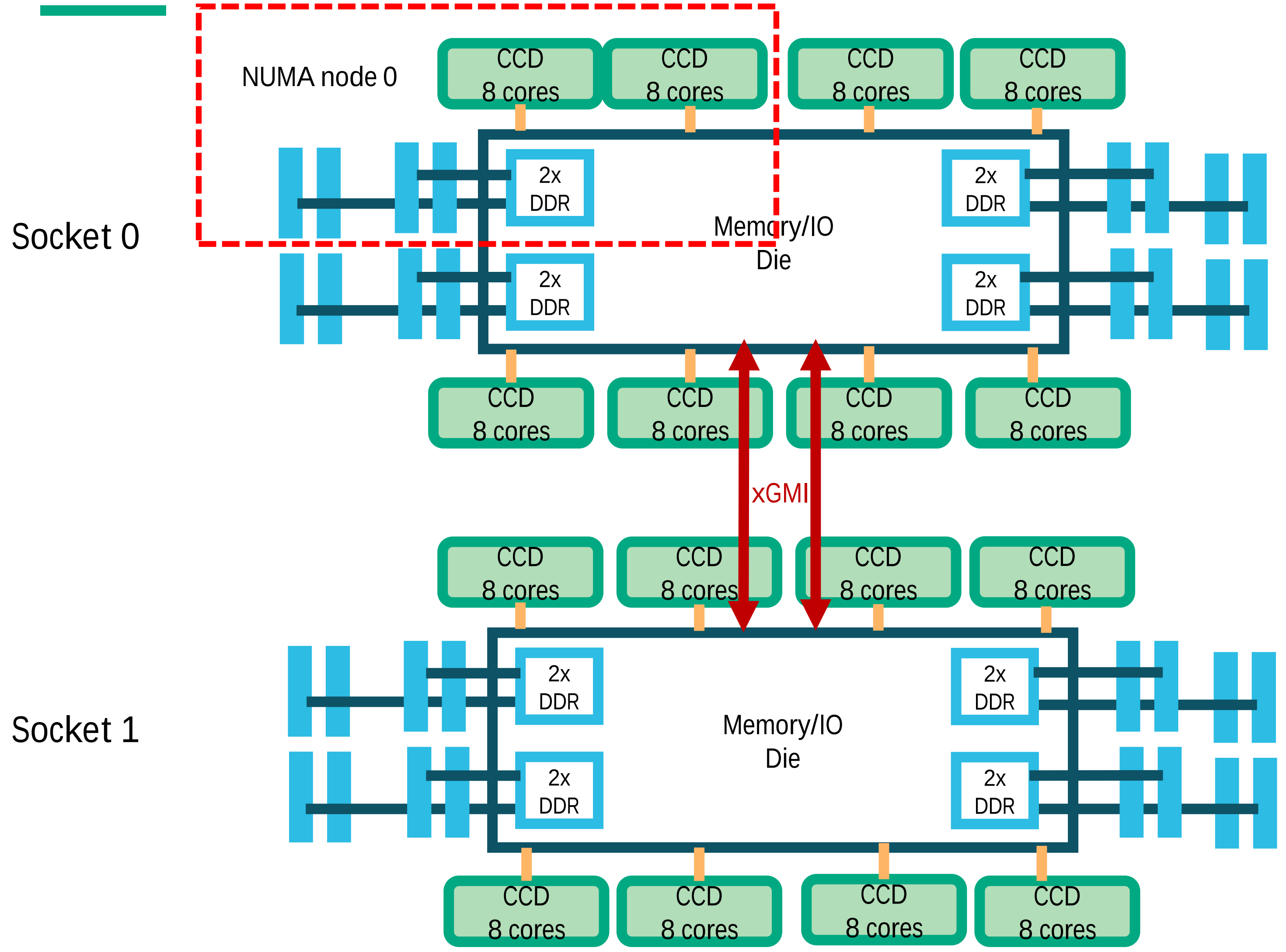
Module CPU - HPE Cray EX425

- Antero Module
  - 4 nodes per module

- Each Anero Node
  - 2 processors AMD Milan 64 cores
  - 1 network interface Slingshot-11 (200Gbps)

Photos: Orian Louant

INTRA-NODE PLACEMENT – Interconnect With Two CPU Sockets



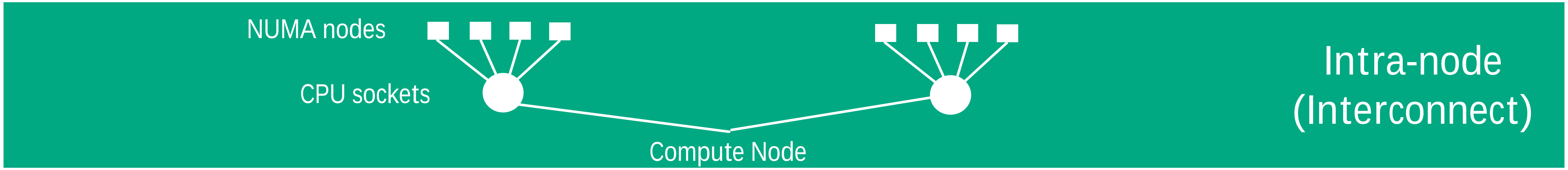
- CPUs in a compute node can be seen a small cluster of interconnect and cache coherent compute groups called NUMA (Non-Uniform Memory Access) nodes.

- A NUMA node is composed of a set cores associated with a memory controller and dedicated affinity to devices such as network interface(s) and/or GPU(s).

- Each LUMI-C node is divided into eight NUMA nodes.

INTRA-NODE PLACEMENT – Potential Performance Issues

Quick reminder: MPI tasks in the same node may exchange data via shared memory communications.



Interconnect

Higher latency:

When fetching data in a different NUMA node ("NUMA effects"), especially on a different CPU socket (higher "NUMA distance").

Saturation:

Heavily using the interconnect (additional hops or link saturation) leads to suboptimal utilization of memory / IO throughputs.

Memory

Saturation:

Collocating more tasks on one NUMA node may lead to unbalanced memory usage (performance and capacity).

INTRA-NODE PLACEMENT – What Can Be Done?

Different strategies depending on the workload. You may want to profile the application first.

A few guidelines:

- Task placement limiting NUMA effects: If possible, place ranks exchanging highest volume of data on the same NUMA node or NUMA nodes with lowest NUMA distance (same CPU socket). Reordering MPI tasks might be an option.

- Avoid making each task affinity spread across multiple NUMA nodes (without specific reason): by default, data get assigned to a NUMA node on first access (first touch policy) based on NUMA the CPU core used. Using multiple NUMA nodes with one task may increase NUMA effects.

- Task placement with balanced NUMA node usage: use same number of tasks per NUMA node to avoid saturating one memory controller and slow down the whole application.

Controlling Intra-Node Task Placement

- Manually reordering MPI tasks
- Coarse grain control using the `--distribution` *(srun)* argument (second level) and `–c` to define the quantity of cores/hyperthreads per task.
- Fine grain control using a CPU map (`--cpu-bind=map_cpu`) or a CPU mask (`--cpu-bind=mask_cpu`) to select CPU cores/hyperthreads accessible by each task.

INTRA-NODE PLACEMENT – Getting Node Information, lscpu

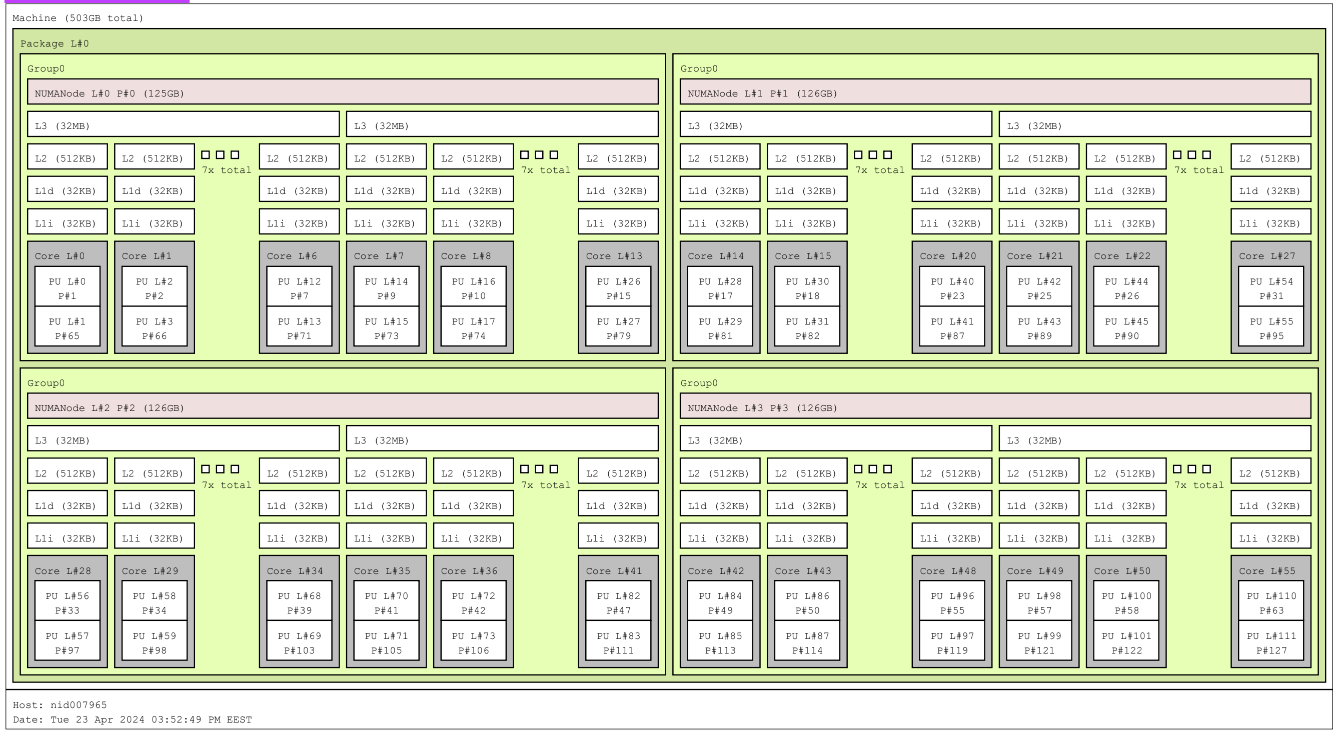
```
lscpu | grep -Ei "CPU\(s\)|Thread|Core\(s\)
 |Socket\(s\)|Numa|Model\ name| MHz|cache"
```

CPU(s):	128
On-line CPU(s) list:	0-127
Thread(s) per core:	2
Core(s) per socket:	64
Socket(s):	1
NUMA node(s):	4
Model name:	AMD EPYC 7A53
CPU MHz:	1925.019
CPU max MHz:	3541.0149
CPU min MHz:	1500.0000
L1d cache:	32K
L1i cache:	32K
L2 cache:	512K
L3 cache:	32768K
NUMA node0 CPU(s):	0-15,64-79
NUMA node1 CPU(s):	16-31,80-95
NUMA node2 CPU(s):	32-47,96-111
NUMA node3 CPU(s):	48-63,112-127

- Check `/proc/cpuinfo` ON the compute nodes
- If `lscpu` is installed on a system, it will list the configuration
- Hyperthreading (aka SMT) is turned ON
  - From a binding point of view, here we have CORES=CPUs

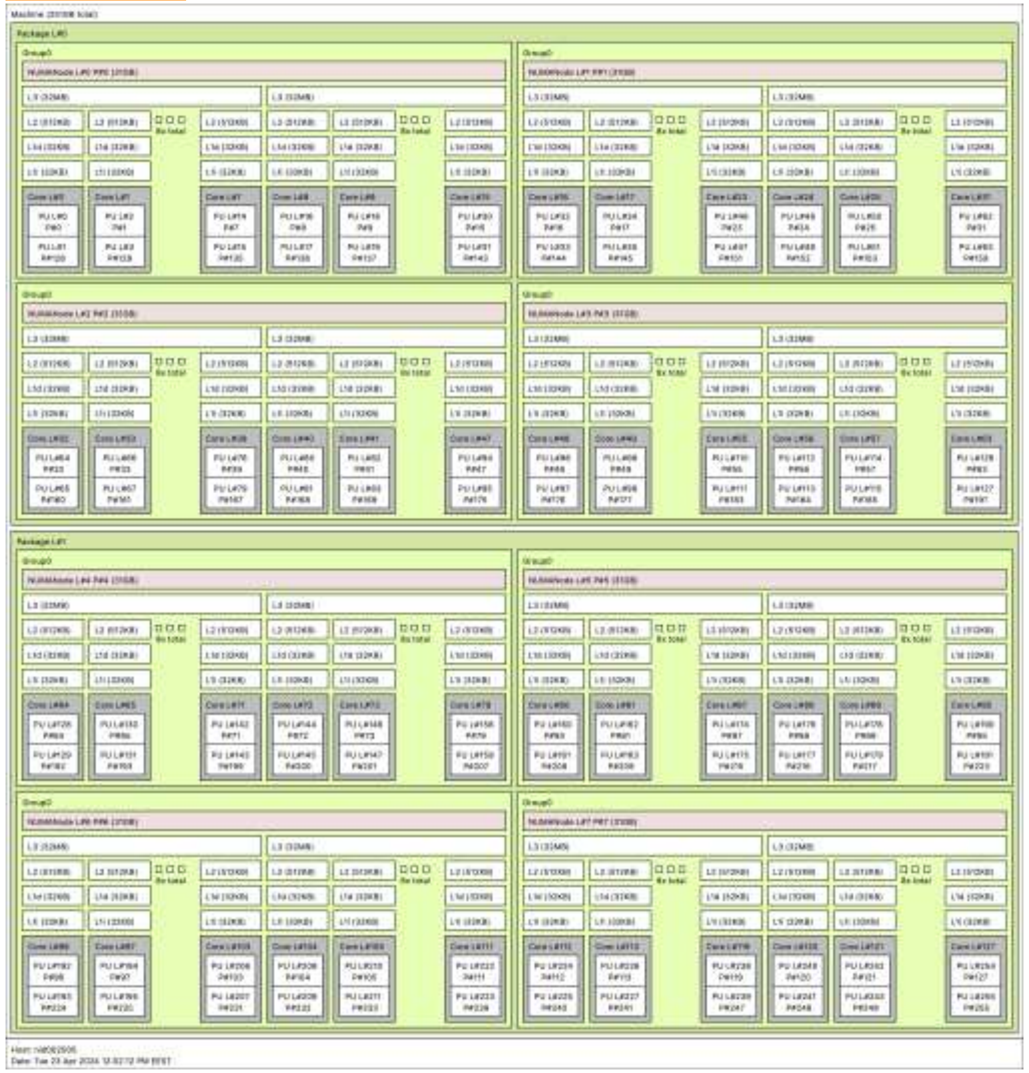
INTRA-NODE PLACEMENT – Getting Node Information with lstopo

LUMI-G



```
Host: nid007965
Date: Tue 23 Apr 2024 03:52:49 PM EEST
```

LUMI-C



```
lstopo --output-format svg -v --no-io > cpu.svg
lstopo-no-graphics -.ascii --only pu
```

INTRA-NODE PLACEMENT – Getting Node Information, numactl

```
numactl --hardware | grep -E "nodes|cpus"
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 64 65 66 67 68 69 70 71 72 73 74 75 76  ...
node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 80 81 82 83 84 85 86 87 88 89 ...
...
```

Each NUMA NODE (4 in total) has

- 16 physical core
- Each core has hardware threads – aka Hyperthreads or Simultaneous MultiThreading (SMT)

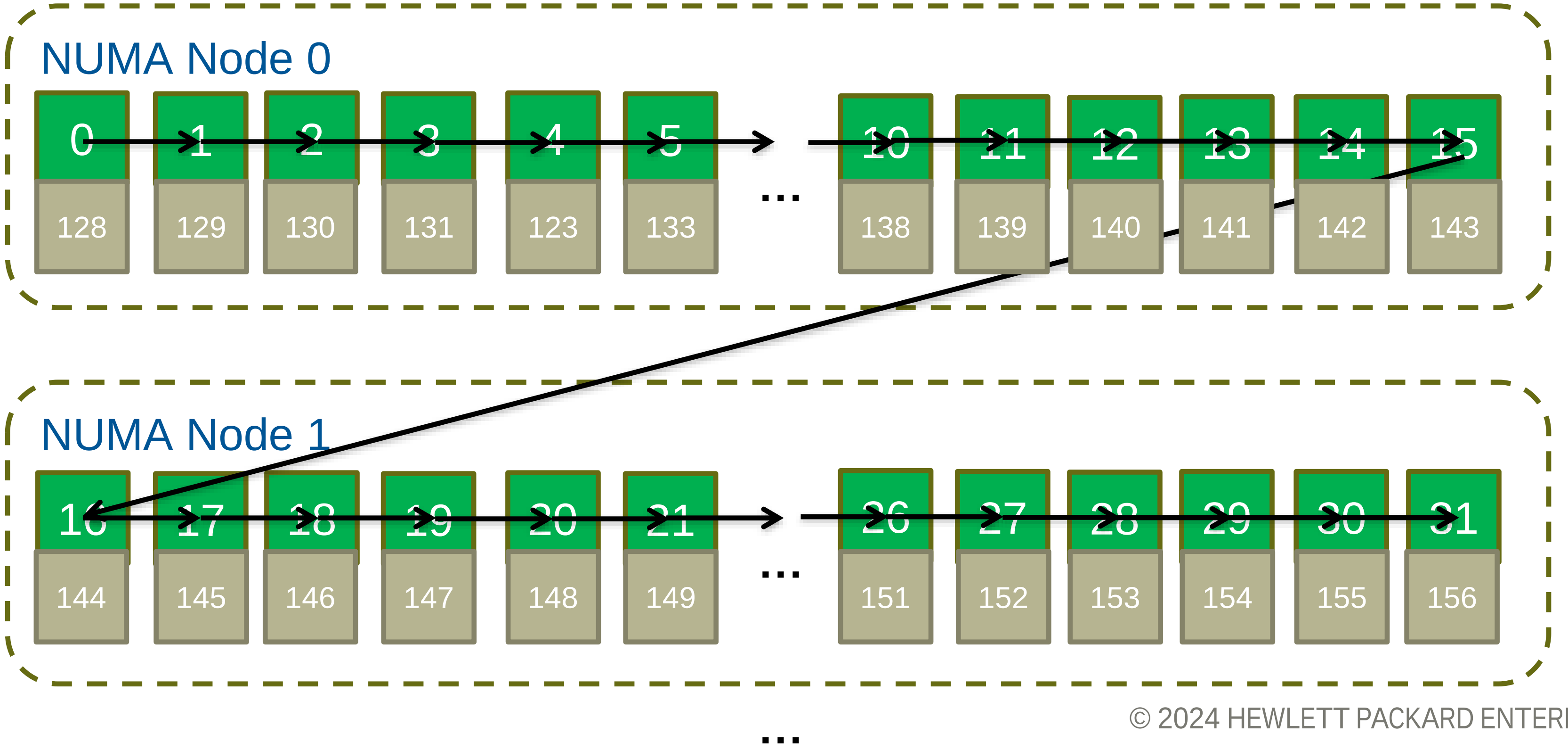
```
lscpu | grep NUMA
NUMA node(s):          4
NUMA node0 CPU(s):     0-15,64-79
NUMA node1 CPU(s):     16-31,80-95
NUMA node2 CPU(s):     32-47,96-111
NUMA node3 CPU(s):     48-63,112-127
```

INTRA-NODE PLACEMENT – Hyperthreads and Numbering

- The numbering of Lumi-C
  - *Actual cores* from 0-127
  - Hyperthreads from 128-255.
- It is not mandatory to use the hyperthreads (disabled by default)
  - The hyperthreads are still there but not utilized.

Option for salloc and srun

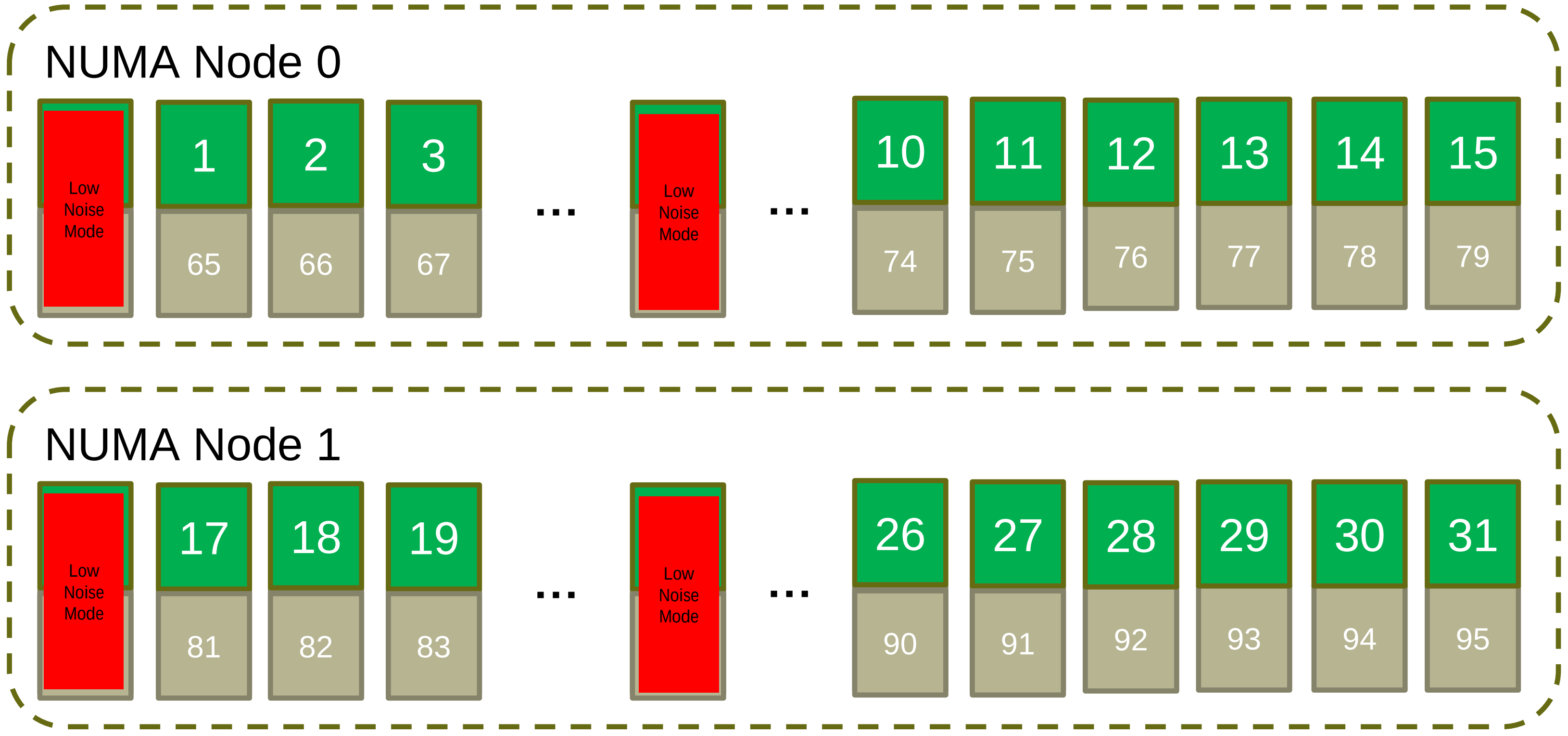
```
--hint=[no]multithread
```



CPUs 0-127 are available,
CPUs 128-255 are ignored

INTRA-NODE PLACEMENT – Low Noise Mode Configuration (1/2)

- The LUMI-G compute nodes have the low-noise mode activated.
  - Helps reduce jitter and variability from run to run
  - This mode reserve 1 core of each CCD to the operating system (8 cores in total)



…

INTRA-NODE PLACEMENT – Low Noise Mode Configuration (2/2)

Jobs requesting > 56 cores (112 SMT threads) per node will never run.

➔Only 56 cores / 112 SMT threads available for user applications

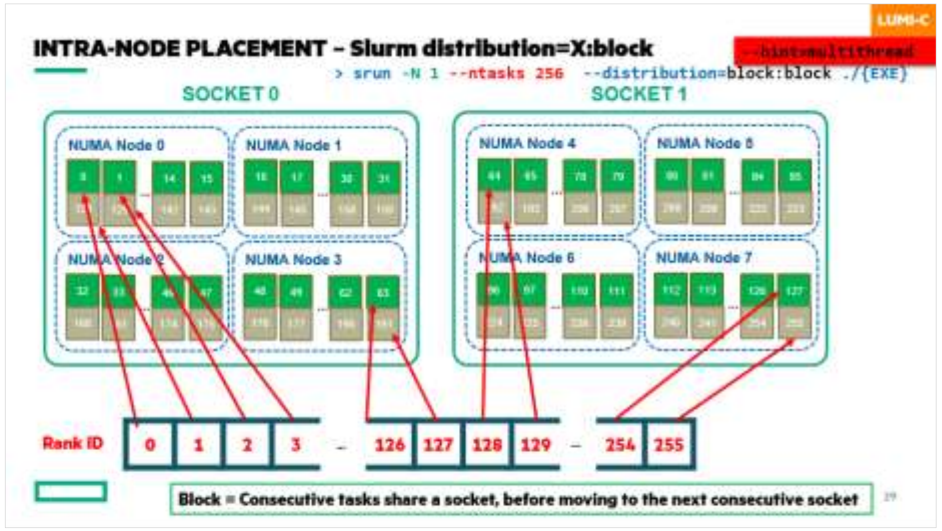
```
> srun -p small-g --nodes=1 --hint=nomultithread sh -c 'echo $SLURM_JOB_CPUS_PER_NODE’
56
> srun -p small-g --nodes=1 --hint=multithread   sh -c 'echo $SLURM_JOB_CPUS_PER_NODE’
112
> srun -p small-g --nodes=1 --hint=multithread   --cpus-per-task=113 ./myApp
srun: error: Unable to allocate resources: Requested node configuration is not available
```

- In general, applications are bandwidth bound (memory or transfers with GPUs)

INTRA-NODE PLACEMENT – Slurm task distribution (level 2)

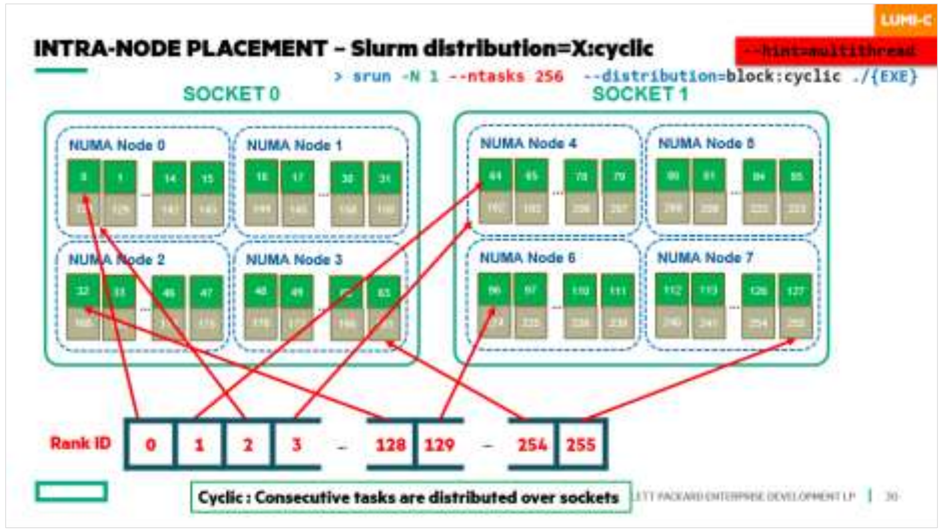
For the second distribution method, the ranks collected in a node in the first distribution step, can be distributed over:

- Sockets on LUMI-C
- NUMA nodes on LUMI-G (NUMA nodes as declared as sockets on LUMI-G)



```
> srun -N 1 --ntasks 256  --distribution=block:block ./{EXE}
```

Block = Consecutive tasks share a socket, before moving to the next consecutive socket



```
> srun -N 1 --ntasks 256  --distribution=block:cyclic ./{EXE}
```

Cyclic : Consecutive tasks are distributed over sockets

`--distribution=[block|cyclic]:block`

consecutive tasks share a CPU socket

`--distribution=[block|cyclic]:cyclic`

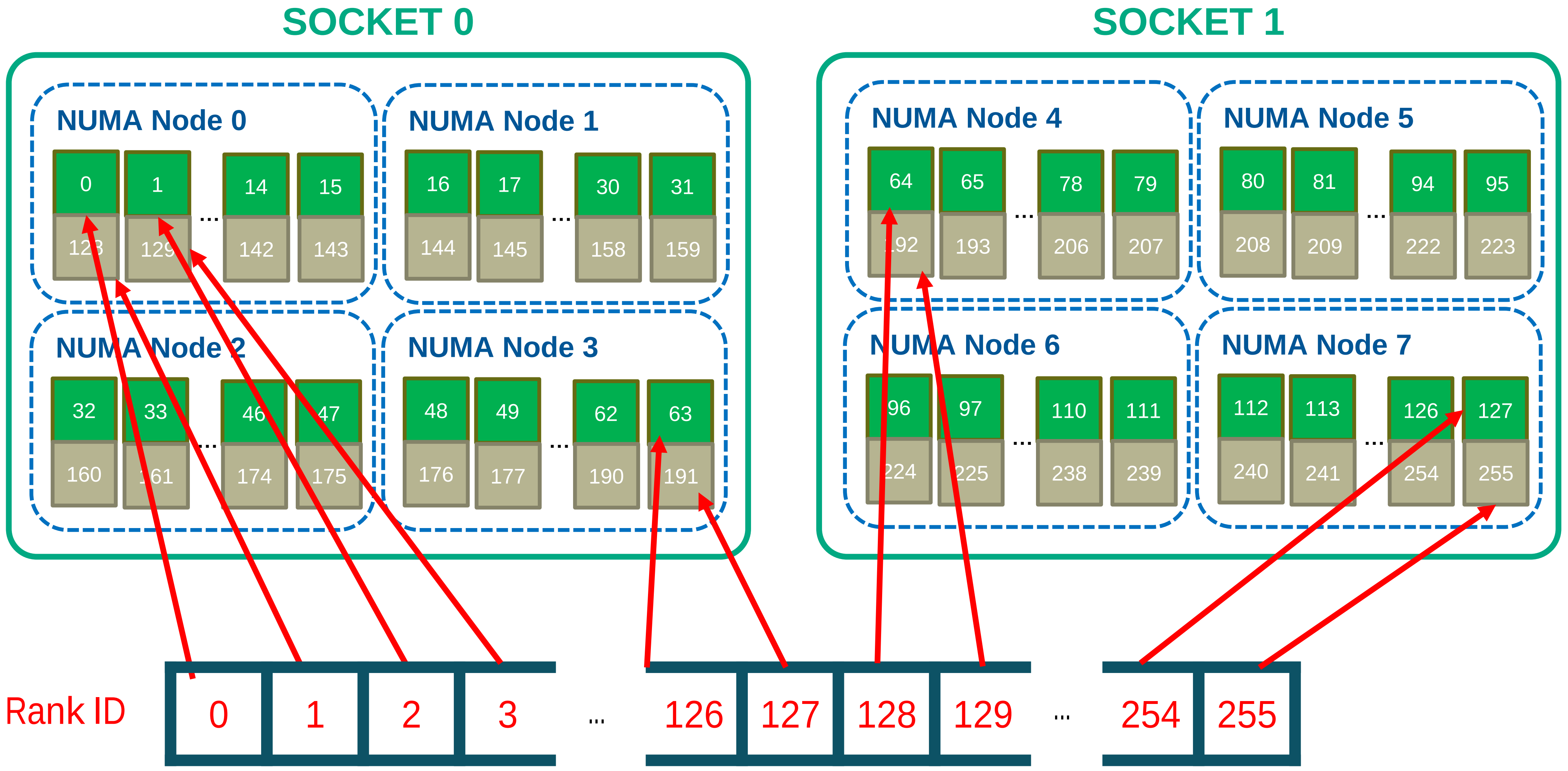
consecutive tasks are distributed over CPU sockets

LUMI-C

--hint=multithread

INTRA-NODE PLACEMENT – Slurm distribution=X:block

```
> srun -N 1 --ntasks 256  --distribution=block:block ./{EXE}
```



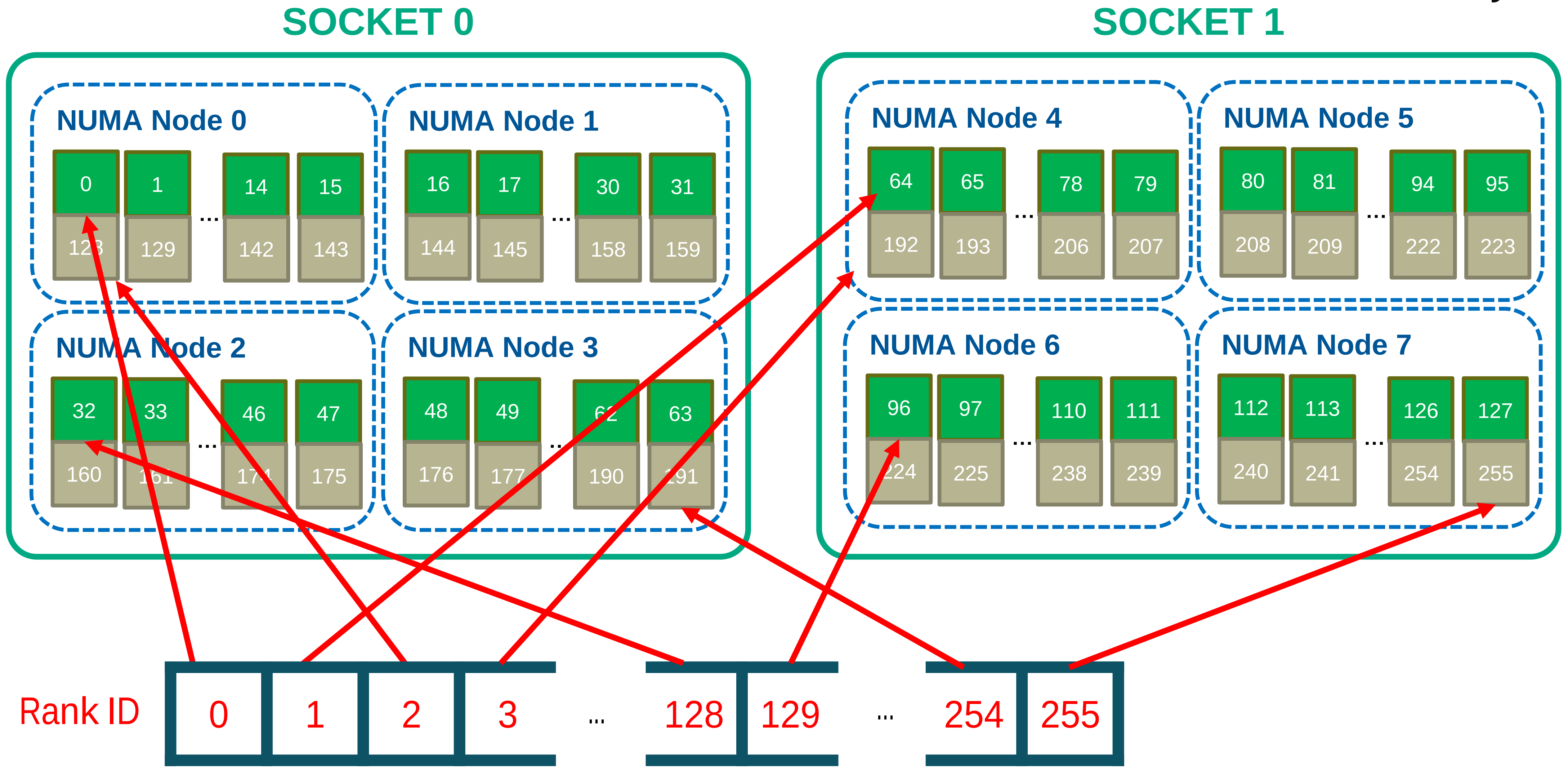
Block = Consecutive tasks share a socket, before moving to the next consecutive socket

LUMI-C

INTRA-NODE PLACEMENT – Slurm distribution=X:cyclic

--hint=multithread

```
> srun -N 1 --ntasks 256  --distribution=block:cyclic ./{EXE}
```

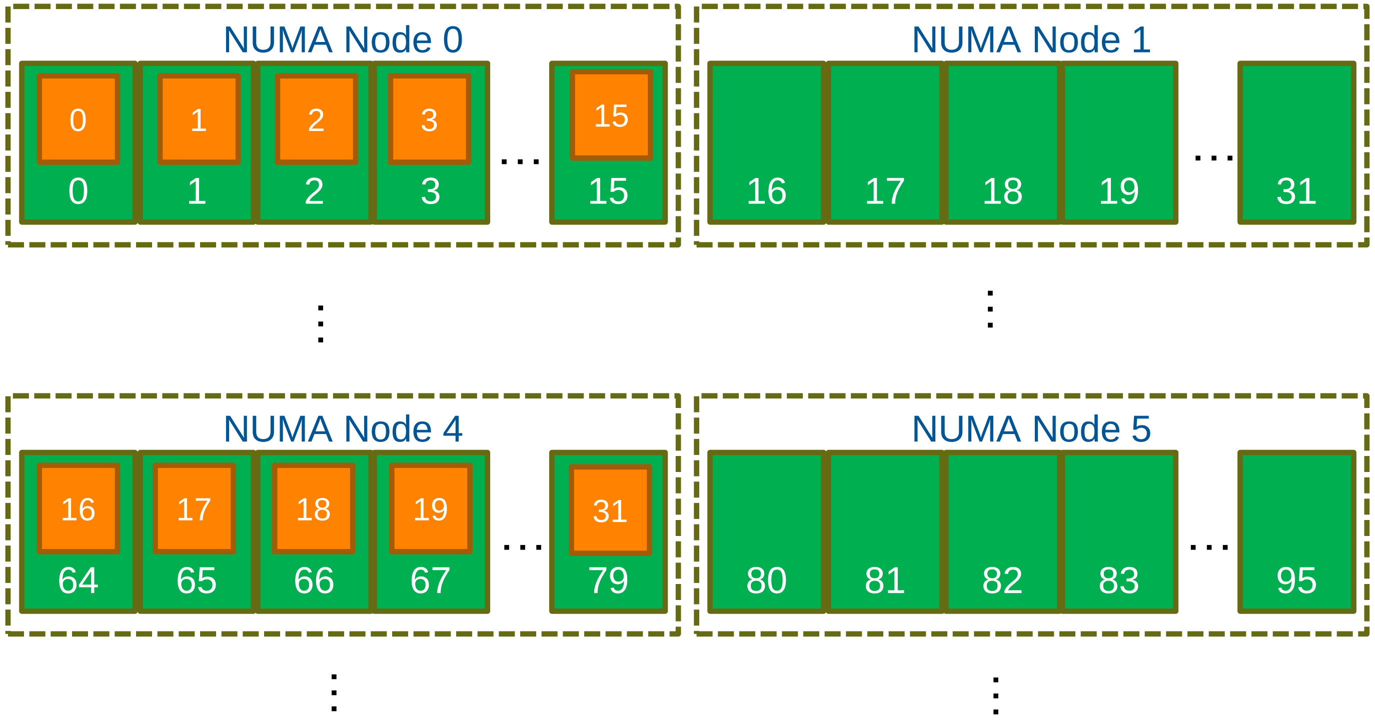


Cyclic : Consecutive tasks are distributed over sockets

INTRA-NODE PLACEMENT – Specifying Quantity of Tasks per Socket

- The number of tasks per socket can be limited

```
srun --nodes 1 -n 32 --ntasks-per-socket=16 --hint=nomultithread ./${EXE}
```



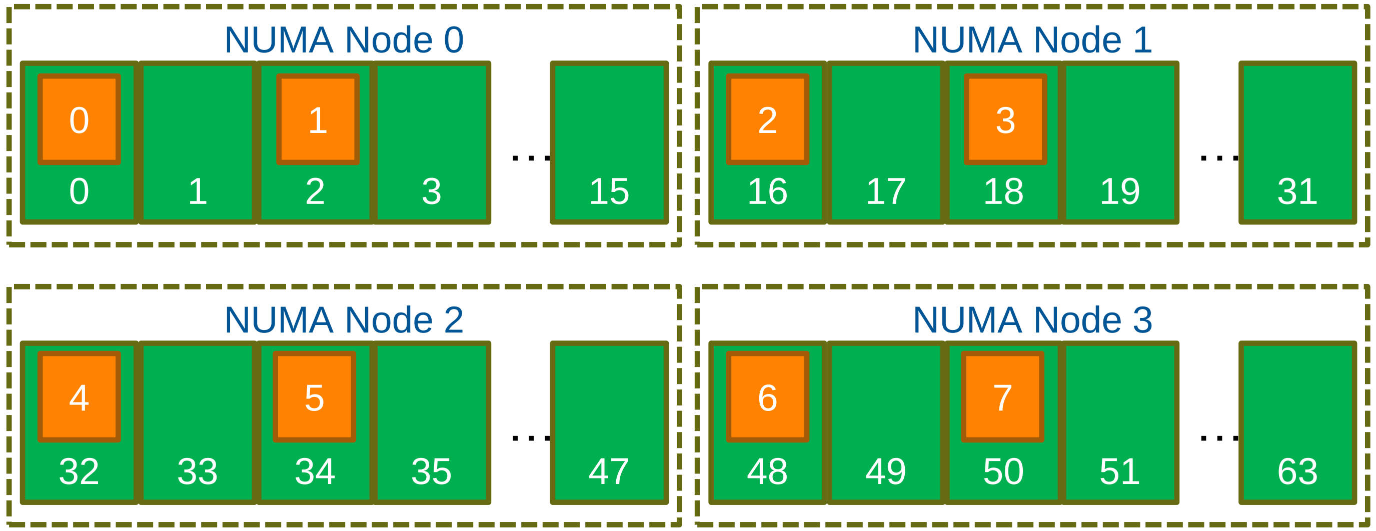
```
host rank thr  count mask
 nid002459    0    0   1 cpu 0
              1    0   1 cpu 1
...
             14    0   1 cpu 14
             15    0   1 cpu 15
             16    0   1 cpu 64
             17    0   1 cpu 65
...
             30    0   1 cpu 78
             31    0   1 cpu 79
```

INTRA-NODE PLACEMENT – Custom Mapping, One Core per Task

- The user may explicitly specific a core/hyperthread accessible by each task on the node.
- The mapping is the same for all nodes
- Useful when underpopulating a node to access more memory bandwidth per task

```
export bind=0,2,16,18,32,34,48,50
srun -N 1 -n 8 --cpu-bind=map_cpu:${bind} ./${EXE}
```

Only makes sense if the node is reserved in --exclusive mode



INTRA-NODE PLACEMENT – Custom Mapping, Multiple Cores per Task (1/2)

- The user may explicitly specific a CPU mask to define multiple cores/hyperthreads accessible by each task on the node.
- The mapping is the same for all nodes
- A mask is a bitmap. When converted to binary, the position (from right to left hand side) of the ones defines the core IDs. For instance: 0xFE -> 1111 1110 (core ID 0 would be skipped and cores from ID 1 to ID 7 selected)
- Following aliases (add to your .bashrc) might be useful for conversions:

```
# Convert hexa binary
0x () {
    local val=$(tr '[a-z]' '[A-Z]' <<< $1)
    echo "binary: `BC_LINE_LENGTH=0 bc <<< \"ibase=16;obase=2;$val\"`"
}

# Convert binary to hexa
0b () {
    local val=$(tr '[a-z]' '[A-Z]' <<< $1)
    echo "hexa: `BC_LINE_LENGTH=0 bc <<< \"ibase=2;obase=10000;$val\"`“
}
```

Example:

```
> 0x ef
binary: 11101111

> 0b 11101111
hexa: EF
```

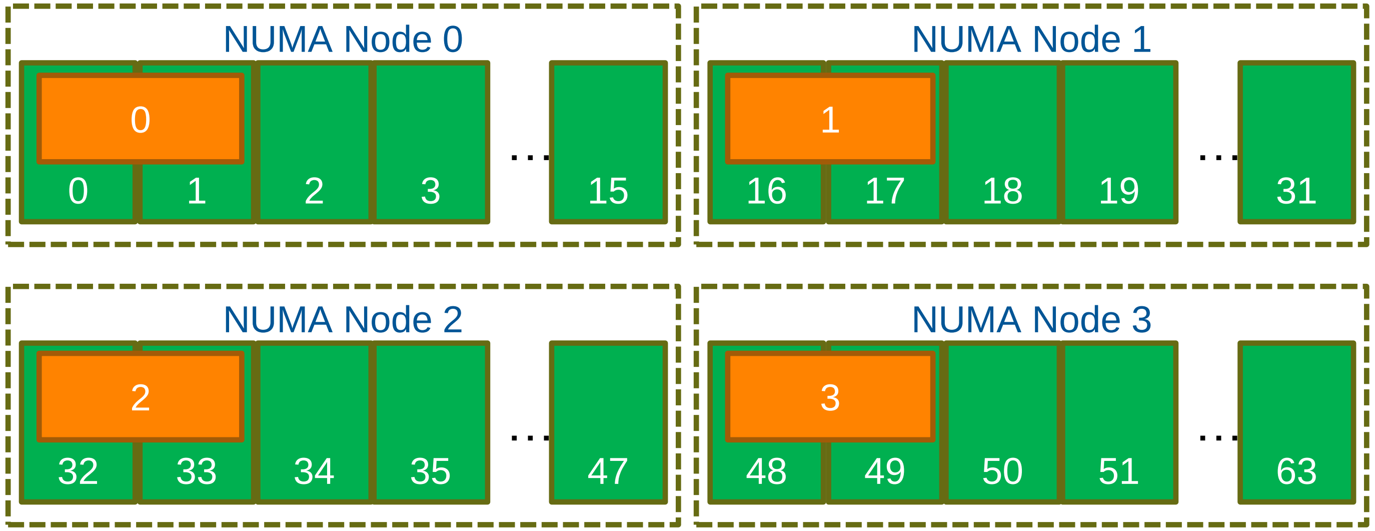
INTRA-NODE PLACEMENT – Custom Mapping, Multiple Cores per Task (2/2)

```
export bind=0x3,0x30000,0x300000000,0x3000000000000
srun -N 1 -n 4 --cpu-bind=mask_cpu:${bind} ./${EXE}
```

Only makes sense if the node is reserved in --exclusive mode

srun will not inherit the --cpus-per-task value requested by salloc or sbatch

```
Cores 0-1:   11 -> 0x3
Cores 16-17: 110000000000000000 -> 0x30000
Cores 32-33: 1100000000000000000000000000000000 -> 0x300000000
Cores 48-49: 11000000000000000000000000000000000000000000000000 -> 0x3000000000000
```



INTRA-NODE PLACEMENT – Takeaways

- Understand node hierarchy and core to NUMA node mapping to
  - Reduce NUMA effects
  - Avoid unbalances (overloading one specific NUMA node)
- 8 cores are not available on LUMI-G nodes

Cheat sheet

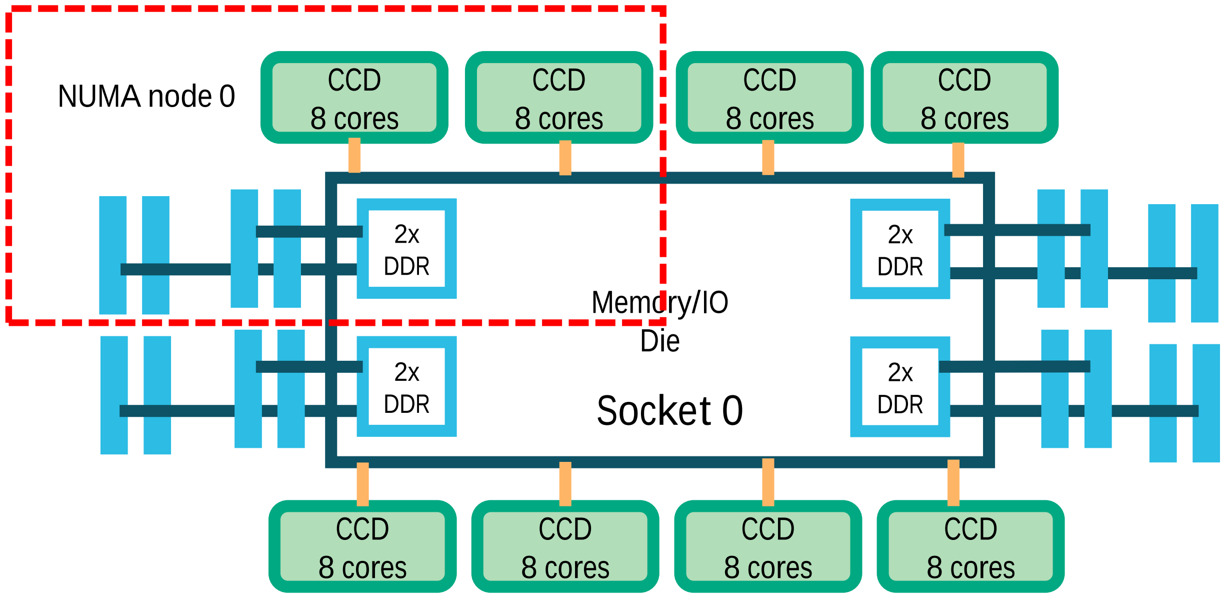
`lscpu, lstopo, numactl –hardware`	Display node information
`salloc sbatch --hint=[no]multithread`	Use or hide hardware threads (hyperthreads)
`srun --distribution=X:block|cyclic`	Modify task distribution pattern across CPU sockets
`srun –-ntasks-per-socket=X`	Specify how many tasks per socket
`srun --cpu-bind=map_cpu:<core_list>`	Provide a CPU core affinity per task
`srun –c X --cpu-bind=mask_cpu:<mask_list>`	Provide multiple CPU core affinities per task

`export MPICH_RANK_REORDER_METHOD=3`
`export MPICH_RANK_REORDER_FILE=<file>`
	Manually defining rank reordering
	(more details in another presentation)

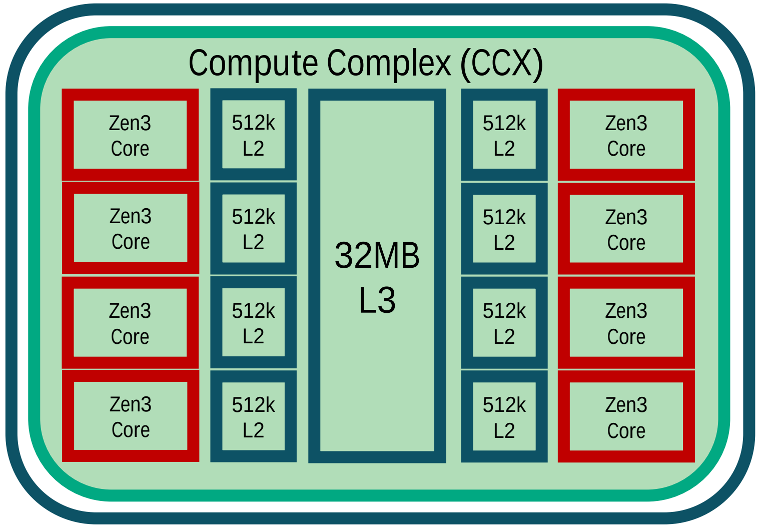
COMPUTE RESOURCES PLACEMENT (CPU CORE AFFINITY)

Hybrid MPI/OpenMP, thread binding to reduce data movements

CORE AFFINITY – AMD CPU, CCD Zoom in



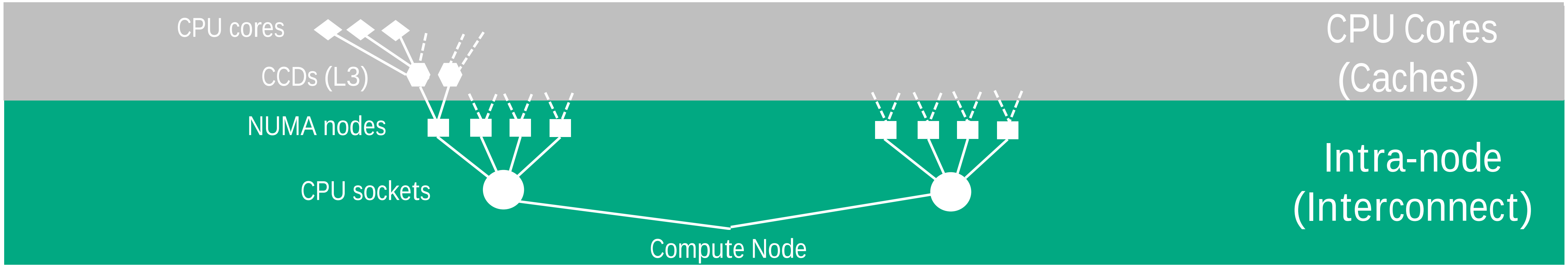
Compute Complex Die (CCD)



Compute Complex Dies (CCDs)

- host cores and L2/L3 cache
  - L1 cache 32kB / core
  - L2 cache 512kB / core
  - L3 cache 32MB / 8-cores

CORE AFFINITY – Binding And Potential Performance Issues



A task may have an affinity to multiple cores (`--cpu-bind=mask_cpu`). The same may apply to threads. We call this binding a process or a thread to CPUs.

BINDING only makes sense if the node is reserved in --exclusive mode

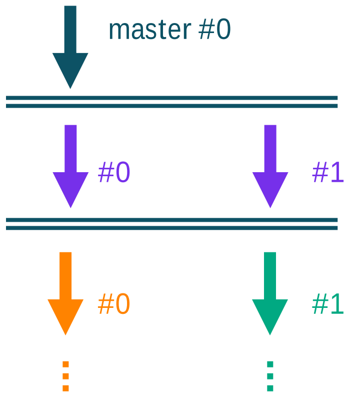
- The concept binding is crucial for optimal performance :
  - memory/cache locality (minimize the data movement internally on the processor)
  - Make sure threads runs only on one core/hyperthread
    - To reduce OS costs when moving it (less locality: NUMA effects, different cache hierarchy)
    - To make sure no core/hyperthread is oversubscribed (resource sharing + context switches)

- It can be hard to spot performance problems relating to binding

Binding Threads with OpenMP
(in hybrid context MPI + OpenMP)

OPENMP – Set the Number of OpenMP Threads

```
export OMP_NUM_THREADS=2
```

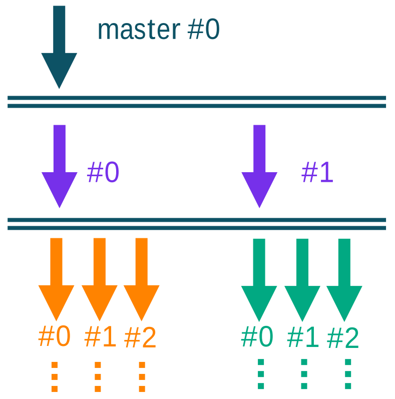


Outer : create 1 new thread
to make 1 team of 2 threads

Inner : No new threads created
to make 2 teams of 1 threads

```
outer: thread_num = 0 ... level = 1
outer: thread_num = 1 ... level = 1
inner: thread_num = 0 ... level = 2
inner: thread_num = 0 ... level = 2
```

```
export OMP_NUM_THREADS=2,2
```



Outer : create 1 new thread
to make 1 team of 2 threads

Inner : 4 new threads to
make 2 teams of 3 threads

```
outer: thread_num = 0 ... level = 1
outer: thread_num = 1 ... level = 1
inner: thread_num = 0 ... level = 2
inner: thread_num = 0 ... level = 2
inner: thread_num = 1 ... level = 2
inner: thread_num = 1 ... level = 2
inner: thread_num = 2 ... level = 2
inner: thread_num = 2 ... level = 2
```

```
int main() {
    #pragma omp parallel
        printf("outer: ...");
        #pragma omp parallel
            printf("inner: ... ");
}
```

OPENMP BINDING – OMP_PLACES

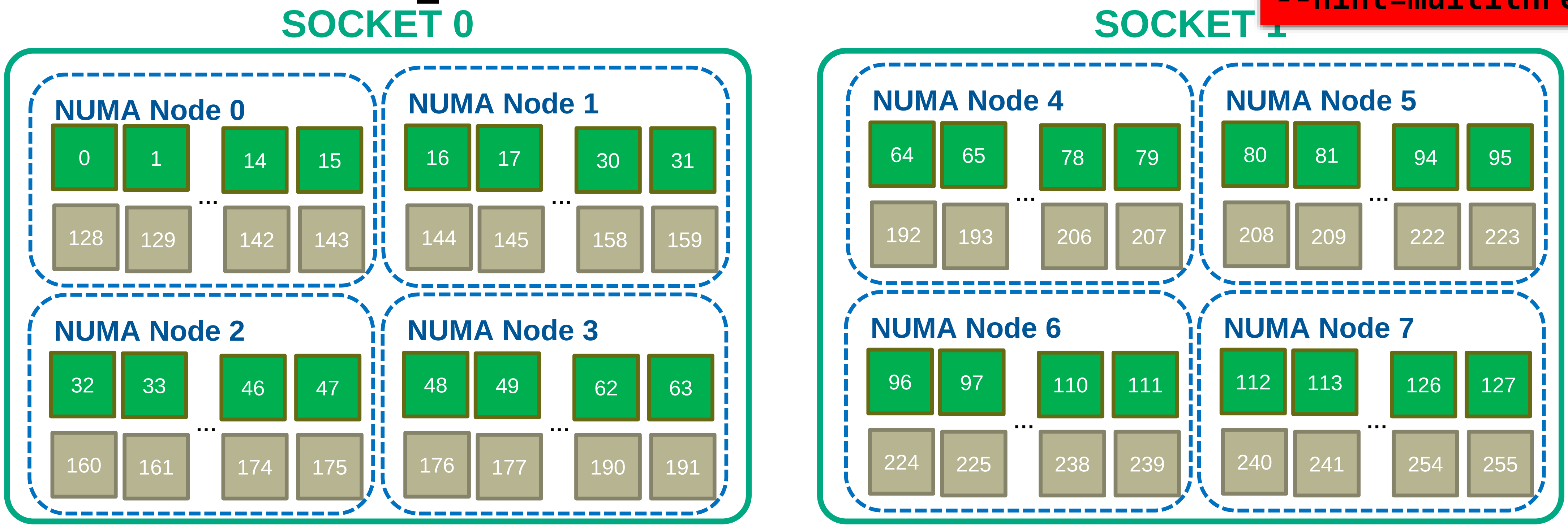
- A list of places (affinity) that threads can be pinned on.
- Each "Place" defines a location where a **thread can "float"**
- Places do NOT depend on the number of threads (OMP_NUM_THREADS)
- Keep in mind that that the MPI task already has affinity which may restrict an OMP places

- `OMP_PLACES=`*`values`* where possible values are:
  - threads: Each place corresponds to a single hardware thread (hyperthread) on the target machine.
  - cores: Each place corresponds to a single core (having one or more hardware threads)
  - sockets: Each place corresponds to a single socket (consisting of one or more cores)
  - Or *a list* with explicit values e.g., "{0:4}:4:4 = {0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}"

LUMI-C

OPENMP BINDING – OMP_PLACES=threads

--hint=multithread



```
> srun -N 1 --ntasks 8 –c 32
```

```
OMP_DISPLAY_ENV=true
```

```
OMP_PLACES = '{0},{128},{1},{129},{2},{130},{3} ...
OMP_PLACES = '{16},{144},{17},{145},{18},{146},{19} ...
OMP_PLACES = '{32},{160},{33},{161},{34},{162},{35} ...
OMP_PLACES = '{48},{176},{49},{177},{50},{178},{51} ...
OMP_PLACES = '{64},{192},{65},{193},{66},{194},{67} ...
OMP_PLACES = '{80},{208},{81},{209},{82},{210},{83} ...
OMP_PLACES = '{96},{224},{97},{225},{98},{226},{99} ...
OMP_PLACES = '{112},{240},{113},{241},{114},{242},{115} ...
```

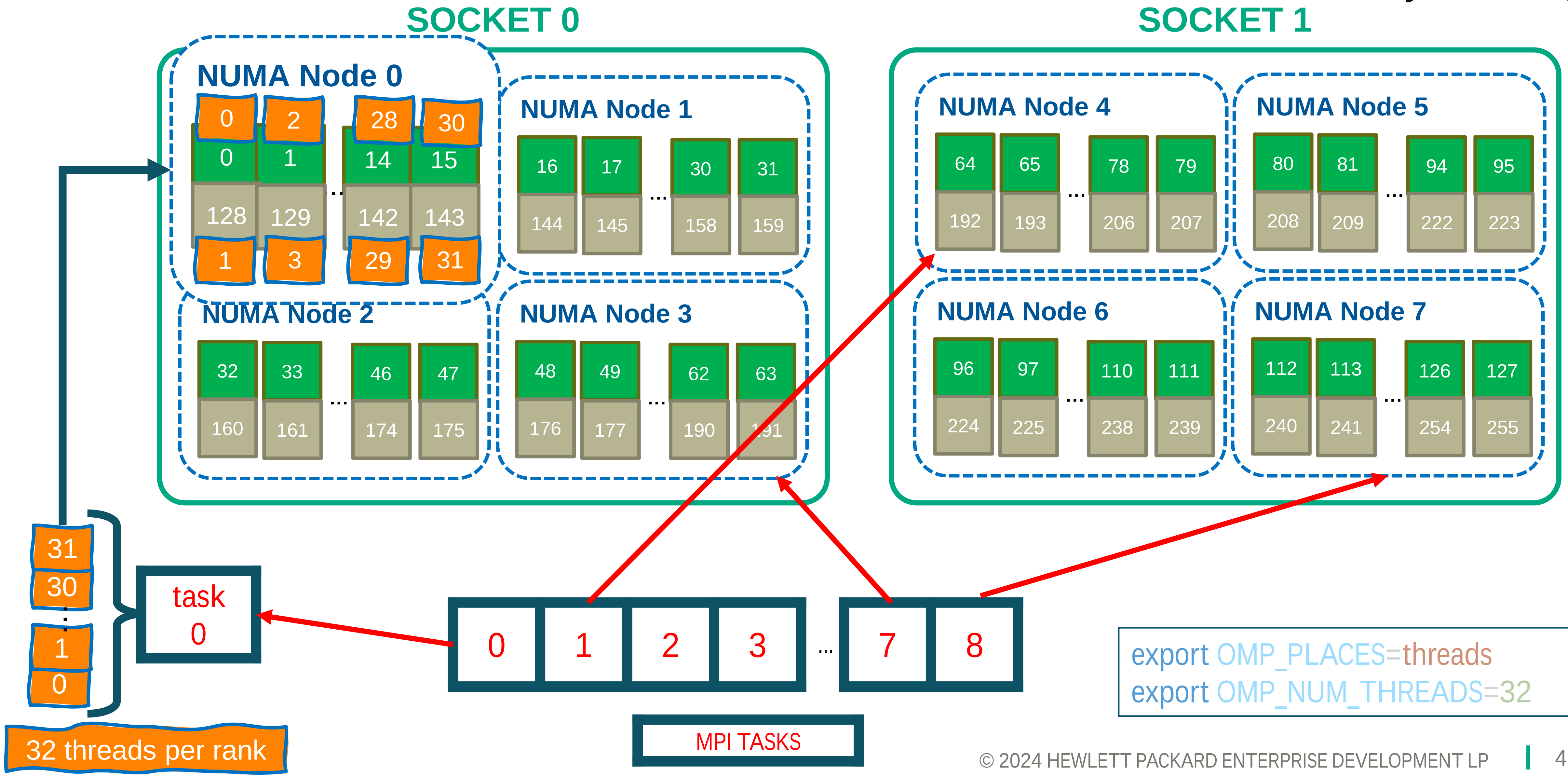
OMP_PLACES=threads

1 place = a single hardware thread

OPENMP BINDING – OMP_PLACES=threads

--hint=multithread

```
> srun -N 1 --ntasks 8 -c 32 --distribution=block:cyclic ./${EXE}
```

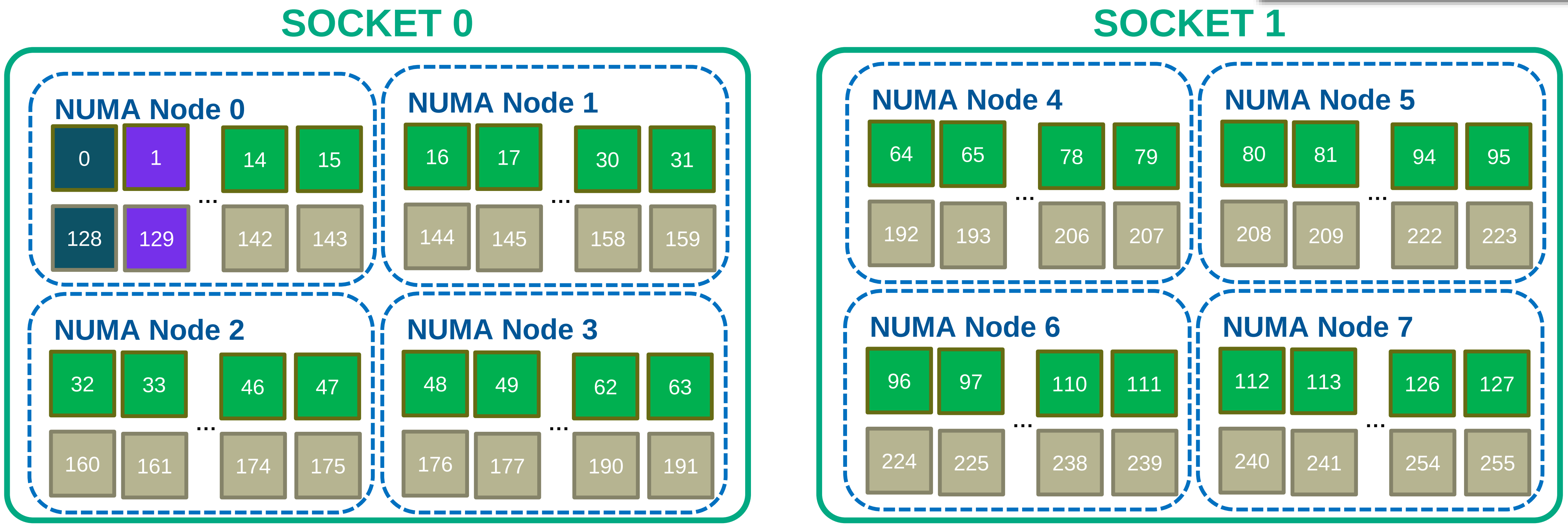


```
export OMP_PLACES=threads
export OMP_NUM_THREADS=32
```

OPENMP BINDING – OMP_PLACES=cores

LUMI-C

--hint=multithread



```
> srun -N 1 --ntasks 8 -c 32
```

OMP_DISPLAY_ENV=true

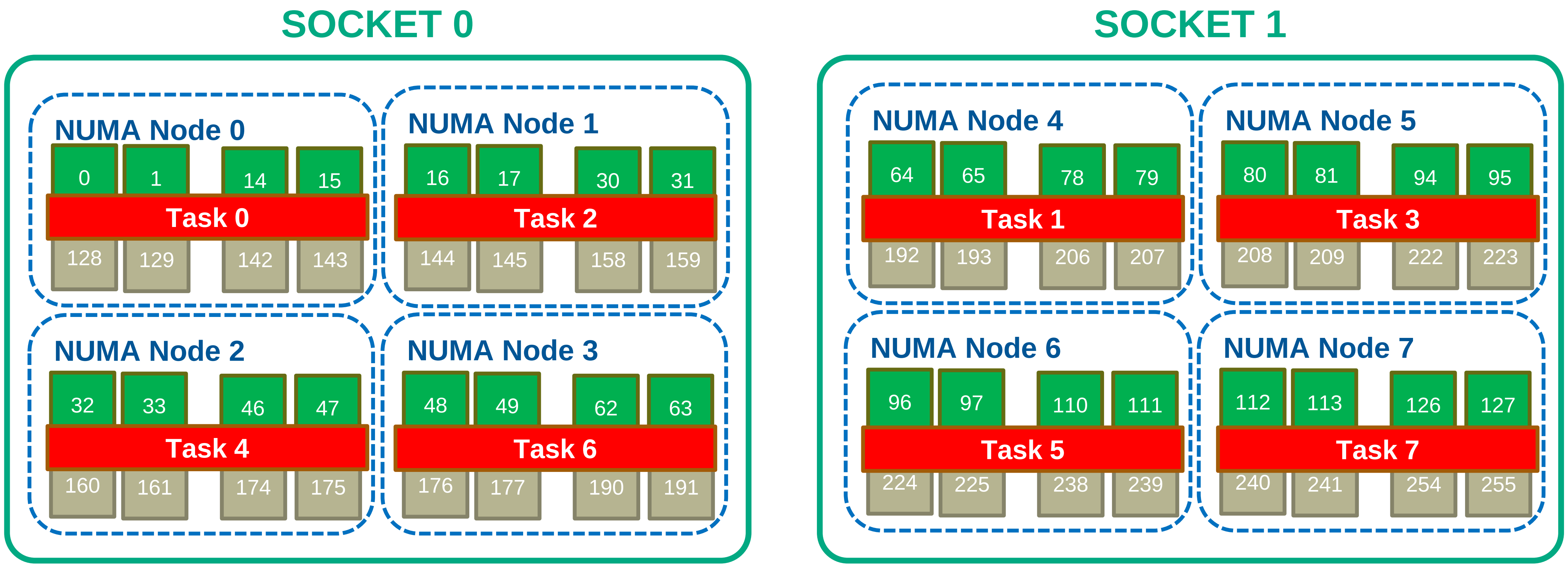
```
OMP_PLACES = '{0,128},{1,129},{2,130},{3,131} ...
OMP_PLACES = '{16,144},{17,145},{18,146},{19,147} ...
OMP_PLACES = '{32,160},{33,161},{34,162},{35,163} ...
OMP_PLACES = '{48,176},{49,177},{50,178},{51,179} ...
OMP_PLACES = '{64,192},{65,193},{66,194},{67,195} ...
OMP_PLACES = '{80,208},{81,209},{82,210},{83,211} ...
OMP_PLACES = '{96,224},{97,225},{98,226},{99,227} ...
OMP_PLACES = '{112,240},{113,241},{114,242},{115,243} ...
```

OMP_PLACES=cores
1 place = a single core

--hint=multithread LUMI-C

OPENMP BINDING – OMP_PLACES=sockets

```
> srun --nodes 1 -n 8 -c 32 --distribution=block:cyclic --hint=multithread
```



```
OMP_PLACES = '{0:16,128:16}'
OMP_PLACES = '{16:16,144:16}'
OMP_PLACES = '{32:16,160:16}'
OMP_PLACES = '{48:16,176:16}'
OMP_PLACES = '{64:16,192:16}'
OMP_PLACES = '{80:16,208:16}'
OMP_PLACES = '{96:16,224:16}'
OMP_PLACES = '{112:16,240:16}'
```

```
host      rank thr  count   mask
nid002241    0    0   32 cpus 0-15 128-143
             1    0   32 cpus 64-79 192-207
             2    0   32 cpus 16-31 144-159
             3    0   32 cpus 80-95 208-223
             4    0   32 cpus 32-47 160-175
             5    0   32 cpus 96-111 224-239
             6    0   32 cpus 48-63 176-191
             7    0   32 cpus 112-127 240-255
```

```
OMP_PLACES=sockets
OMP_NUM_THREADS=1
```

threads can “float”

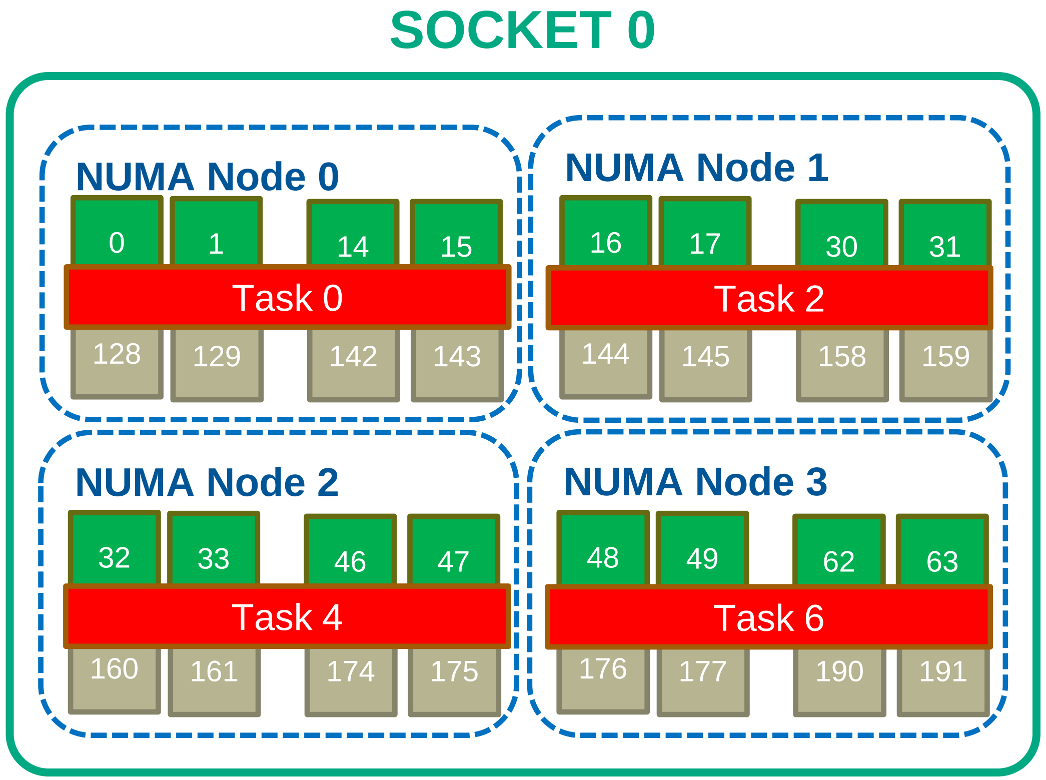
OPENMP BINDING – OMP_PROC_BIND

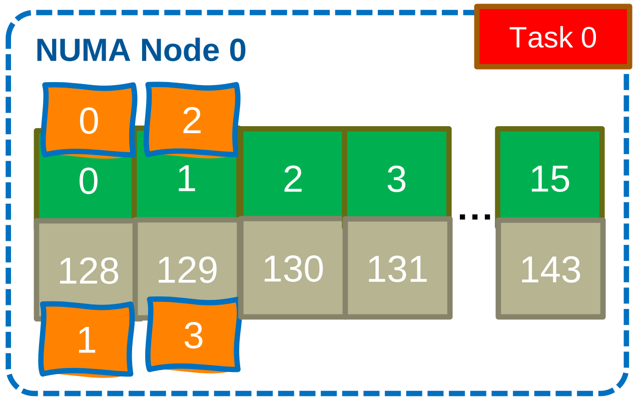
- Sets the binding of threads to processors.
  - close: Bind threads close to the master thread while still distributing for load balancing.
  - spread: Bind threads as evenly distributed (spread) as possible.
  - master: Bind threads to the same place as the master thread.
  - false: turns off OMP binding

OPENMP BINDING – OMP_PROC_BIND=close

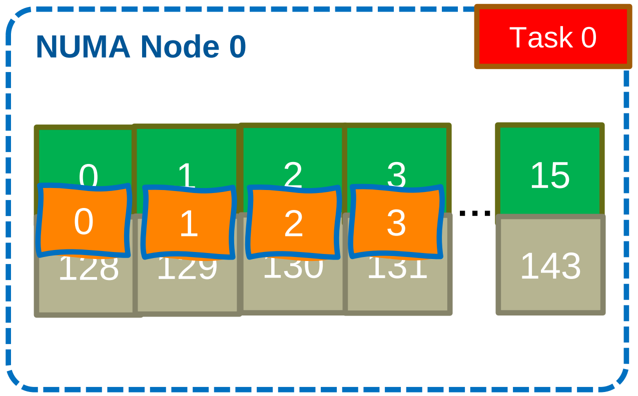
--hint=multithread

- Here we specify 8 MPI tasks with 4 OpenMP threads with places = threads or cores





```
export OMP_PROC_BIND=close
export OMP_PLACES=threads
export OMP_NUM_THREADS=4

      host rank thr  count mask
 nid002241    0   0  1 cpu 0
                  1  1 cpu 128
                  2  1 cpu 1
                  3  1 cpu 129
              1   0  1 cpu 64
...
```



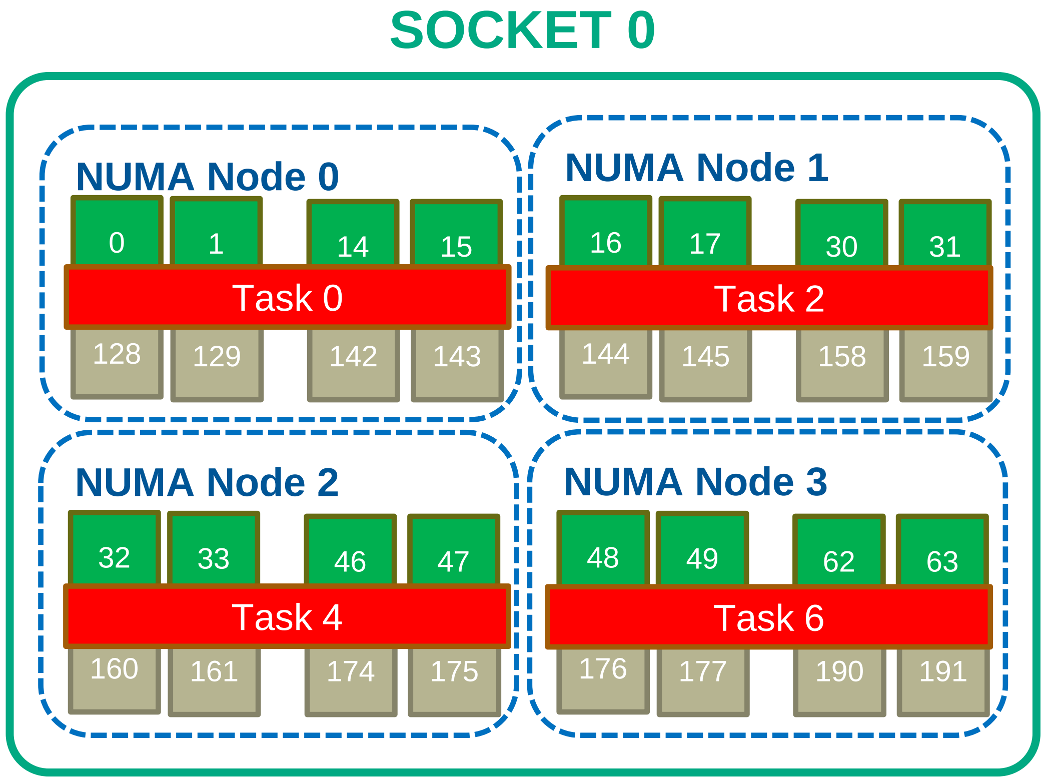
```
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export OMP_NUM_THREADS=4

      host rank thr  count  mask
 nid002241    0   0  2 cpus 0 128
                  1  2 cpus 1 129
                  2  2 cpus 2 130
                  3  2 cpus 3 131
              1   0  2 cpus 64 192
...
```

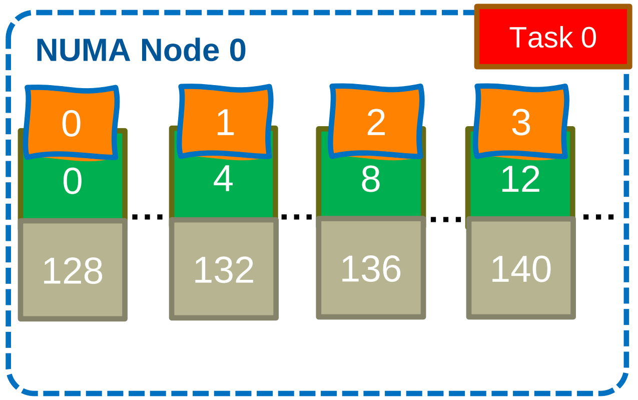
```
> srun -N 1 --ntasks 8 -c 32--distribution=block:cyclic --hint=multithread ./${EXE}
```

OPENMP BINDING – OMP_PROC_BIND=spread

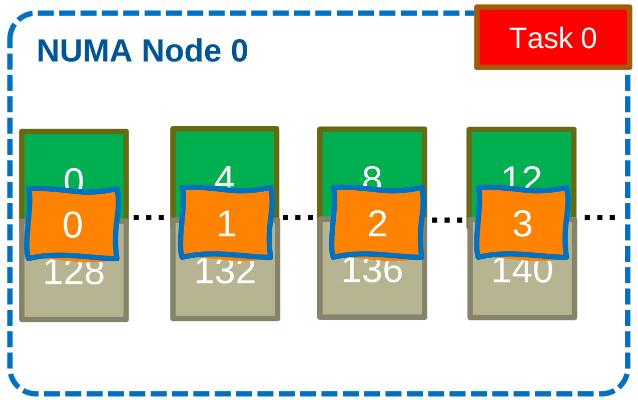
--hint=multithread

- Here we specify 8 MPI tasks with 4 OpenMP threads with places = threads or cores





```
export OMP_PROC_BIND=spread
export OMP_PLACES=threads
export OMP_NUM_THREADS=4

       host rank thr  count mask
  nid002241    0   0   1 cpu 0
                   1   1 cpu 4
                   2   1 cpu 8
                   3   1 cpu 12
               1   0   1 cpu 64
...
```



```
export OMP_PROC_BIND=spread
export OMP_PLACES=cores
export OMP_NUM_THREADS=4

       host rank thr  count  mask
  nid002241    0   0   2 cpus 0 128
                   1   2 cpus 4 132
                   2   2 cpus 8 136
                   3   2 cpus 12 140
               1   0   2 cpus 64 192
...
```

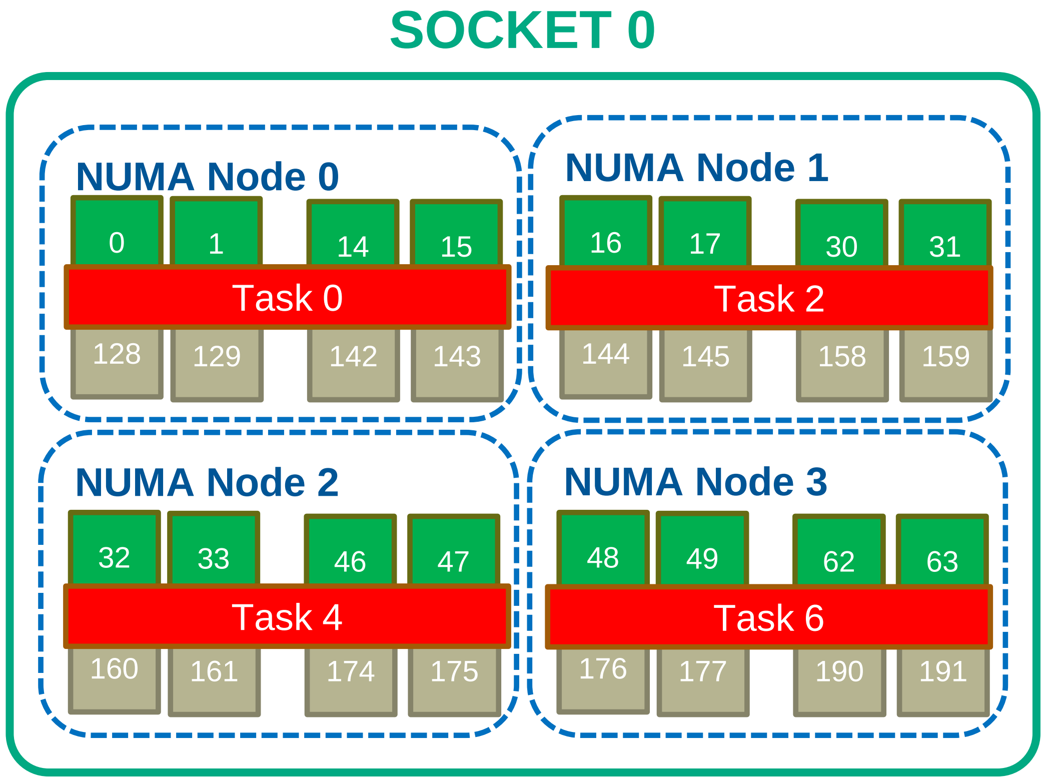
```
> srun -N 1 --ntasks 8 –c 32--distribution=block:cyclic --hint=multithread ./{EXE}
```

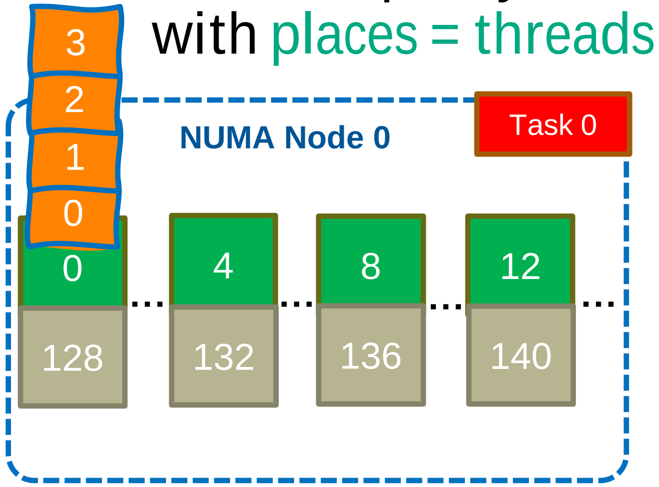
LUMI-C

OPENMP BINDING – OMP_PROC_BIND=master

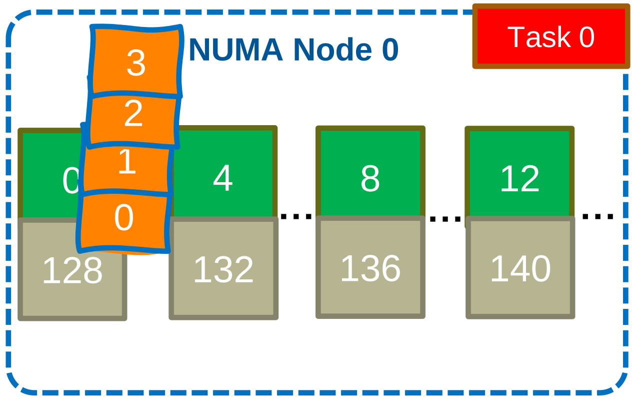
--hint=multithread

- Here we specify 8 MPI tasks with 4 OpenMP threads with places = threads or cores





```
export OMP_PROC_BIND=master
export OMP_PLACES=threads
export OMP_NUM_THREADS=4

     host rank thr  count mask
nid002241    0   0  1 cpu 0
                 1  1 cpu 0
                 2  1 cpu 0
                 3  1 cpu 0
             1   0  1 cpu 64
...
```



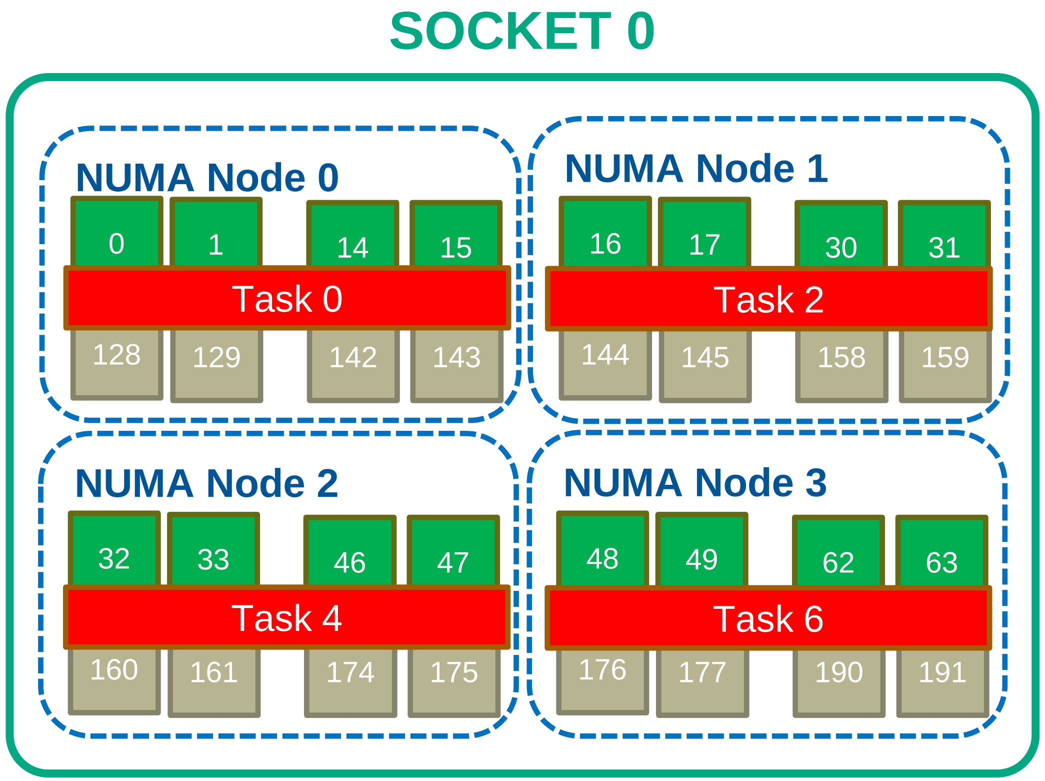
```
export OMP_PROC_BIND=master
export OMP_PLACES=cores
export OMP_NUM_THREADS=4

     host rank thr  count  mask
nid002241    0   0  2 cpus 0 128
                 1  2 cpus 0 128
                 2  2 cpus 0 128
                 3  2 cpus 0 128
             1   0  2 cpus 64 192
...
```

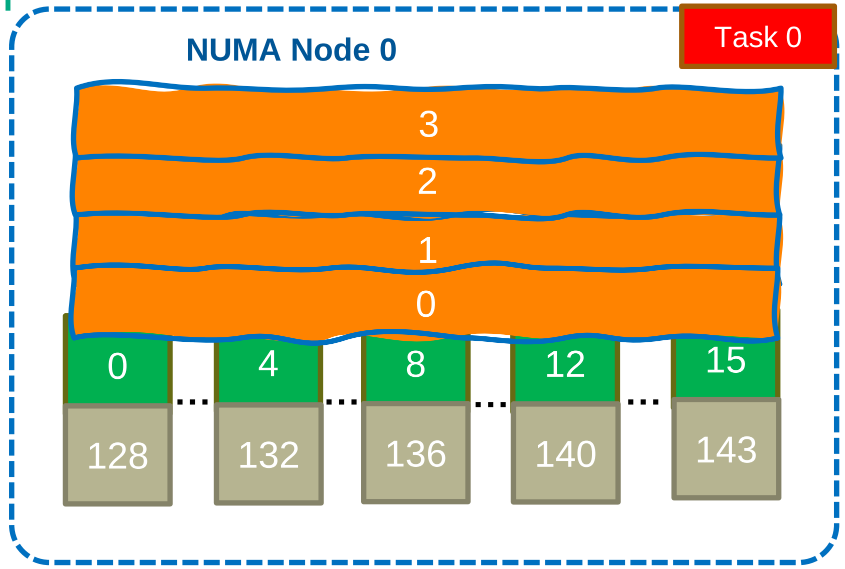
```
> srun -N 1 --ntasks 8 -c 32--distribution=block:cyclic --hint=multithread ./{EXE}
```

OPENMP BINDING – OMP_PROC_BIND=false

--hint=multithread

- Here we specify 8 MPI tasks with 4 OpenMP threads with places = threads or cores





```
export OMP_PROC_BIND=false
export OMP_PLACES=threads | cores | sockets #same results
export OMP_NUM_THREADS=4
 host rank thr  count   mask
 nid002241    0    0   32 cpus 0-15 128-143
                   1   32 cpus 0-15 128-143
                   2   32 cpus 0-15 128-143
                   3   32 cpus 0-15 128-143
              1    0   32 cpus 64-79 192-207
```

```
> srun -N 1 --ntasks 8 -c 32--distribution=block:cyclic --hint=multithread ./{EXE}
```

CORE AFFINITY – Takeaways

- Multiple levels of setting affinities (ex. MPI Ranks, OpenMP): OpenMP threads affinities are restricted by the parent process (MPI task), finer control can be set using OMP environment variables.
- Ensure cores or hyperhtreads are not oversubscribed (shared compute resource, OS context switches)
- Prefer binding a thread to a single core: Allowing threads to float between cores may impact cache locality and induce NUMA effects

Cheat sheet

`export OMP_NUM_THREADS=X`	Set number of OpenMP threads per task
`export OMP_PLACES=threads|cores|sockets|<list>`	The way the threads are distributed
`export OMP_PROC_BIND=close|spread|master|false`	Assinging a core/hyperthread to each thread

INTRA-NODE PLACEMENT FOR GPUS

Distributing tasks to better harness GPUs

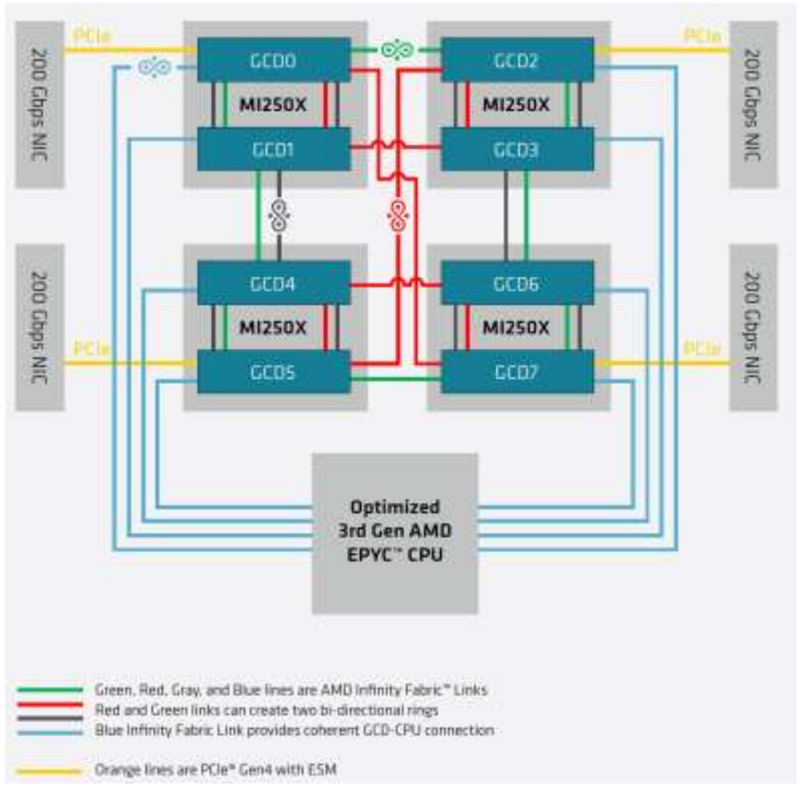
PLACEMENT FOR GPUS – Resources on Lumi-G

On Lumi-G:

- 4 MI250X per node = 8 GCDs (Graphics Complex Die) in total
- Will show as 8 separate GPUs according to Slurm terminology

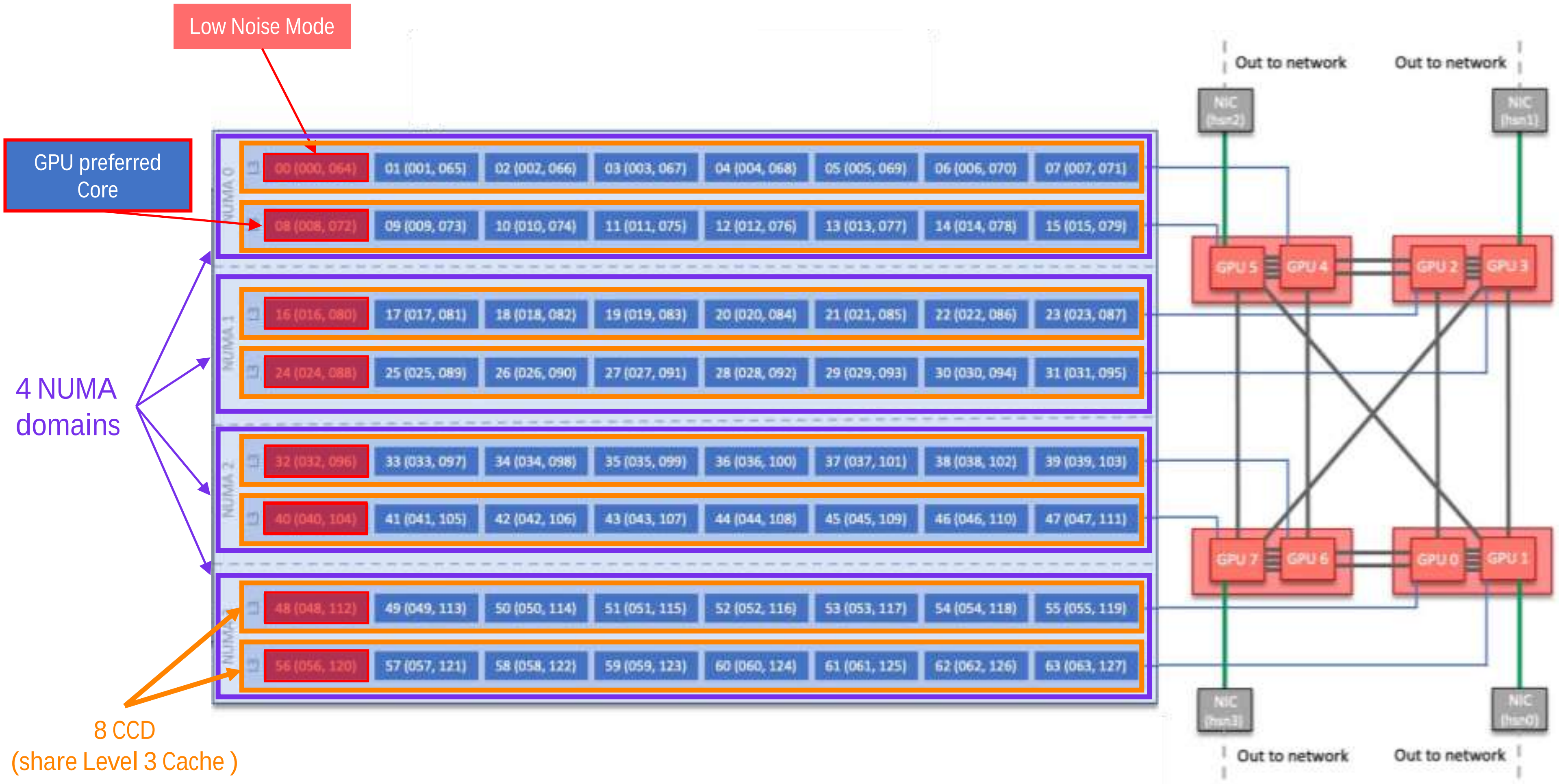
Note :

- Just Keep in mind that 2 GDCs can be on the same physical package = better connectivity


200 Gbps NIC
200 Gbps NIC
200 Gbps NIC
200 Gbps NIC
PCIe
PCIe
PCIe
PCIe
GCD0
MI250X
GCD1
GCD2
MI250X
GCD3
GCD4
MI250X
GCD5
GCD6
MI250X
GCD7
Optimized 3rd Gen AMD EPYC™ CPU
Green, Red, Gray, and Blue lines are AMD Infinity Fabric™ Links
Red and Green links can create two bi-directional rings
Blue Infinity Fabric Link provides coherent GCD-CPU connection
Orange lines are PCIe® Gen4 with ESM
One MI250X GPU is spitted in Two GCDs

```
uan02:~> sinfo -o "%10R  %10c  %10m  %20G  %20F" -p standard,standard-g
PARTITION   CPUS        MEMORY      GRES                  NODES(A/I/O/T)
standard    256         229376      (null)                1151 / 317 / 260 / 1728
standard-g  128         491520      gpu:mi250:8(S:0)      2346 / 125 / 92 / 2563
```

PLACEMENT FOR GPUS – CPU / GPU Affinity



MPI Ranks >= 8

- Pinned to the 8 CCD

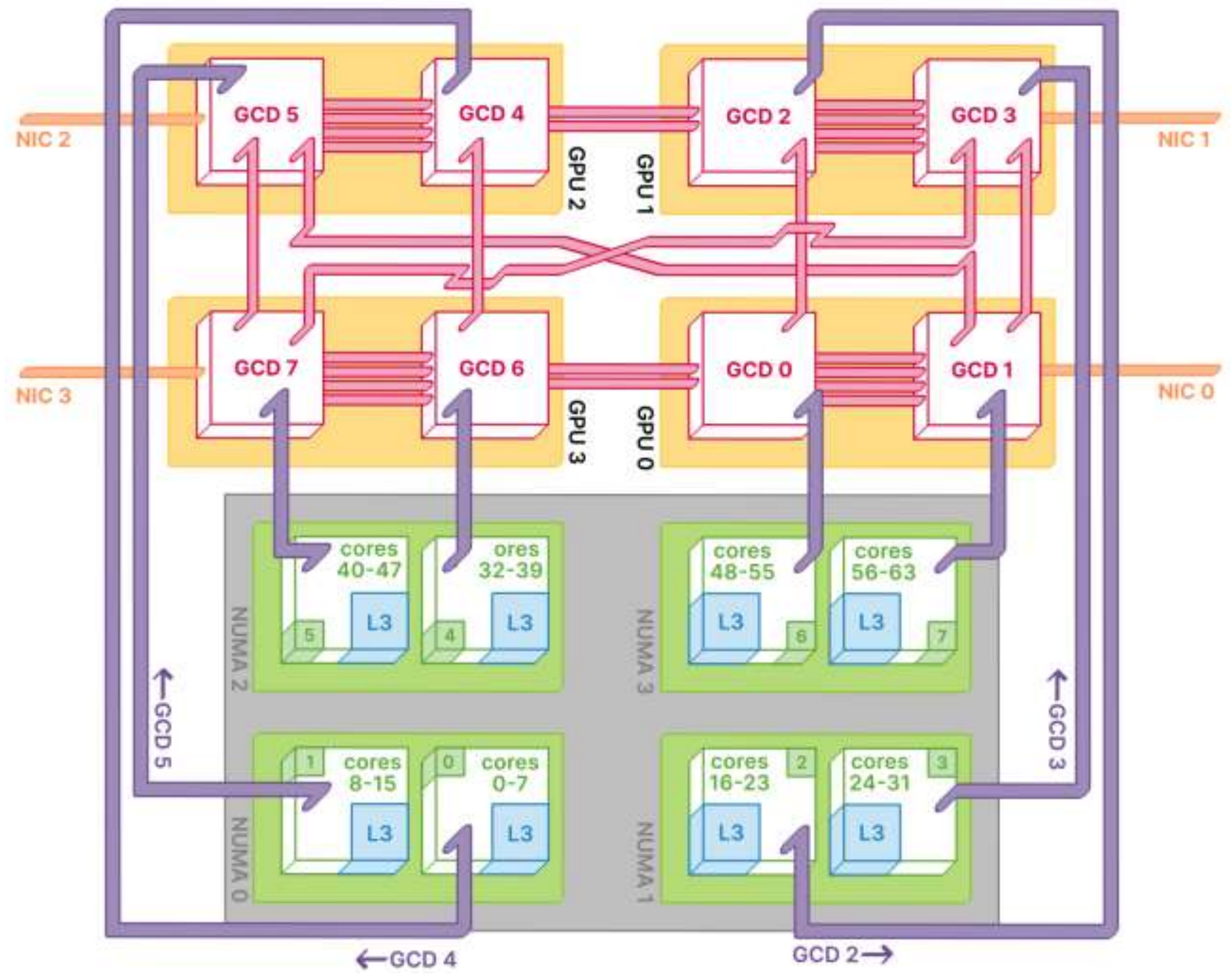
OpenMP Threads <= 7

- Cores 0,8,16,24,32,40,48,56 not available

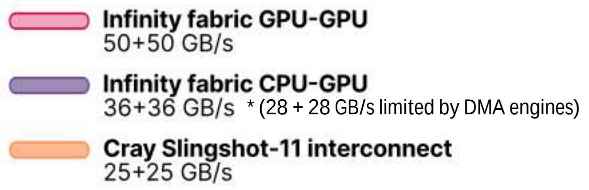
The first cores in each CDD are reserved for
- servicing the OS + GPU management

Credit : ORNL

PLACEMENT FOR GPUS –Potential Performance Issues



- NUMA affinity for each GPU: Improper task placement may impact transfers between the GCDs and main memory (aka Host to Device and Device to Host transfers).
- Asymmetric interconnect between GPUs (GCDs): Task ordering maters to ensure faster inter-GCD communications (aka IPC).
- Network interfaces directly attached to the GPUs: Transfers could be slightly slower when initiated by the CPU.


Infinity fabric GPU-GPU
50+50 GB/s
Infinity fabric CPU-GPU
36+36 GB/s * (28 + 28 GB/s limited by DMA engines)
Cray Slingshot-11 interconnect
25+25 GB/s

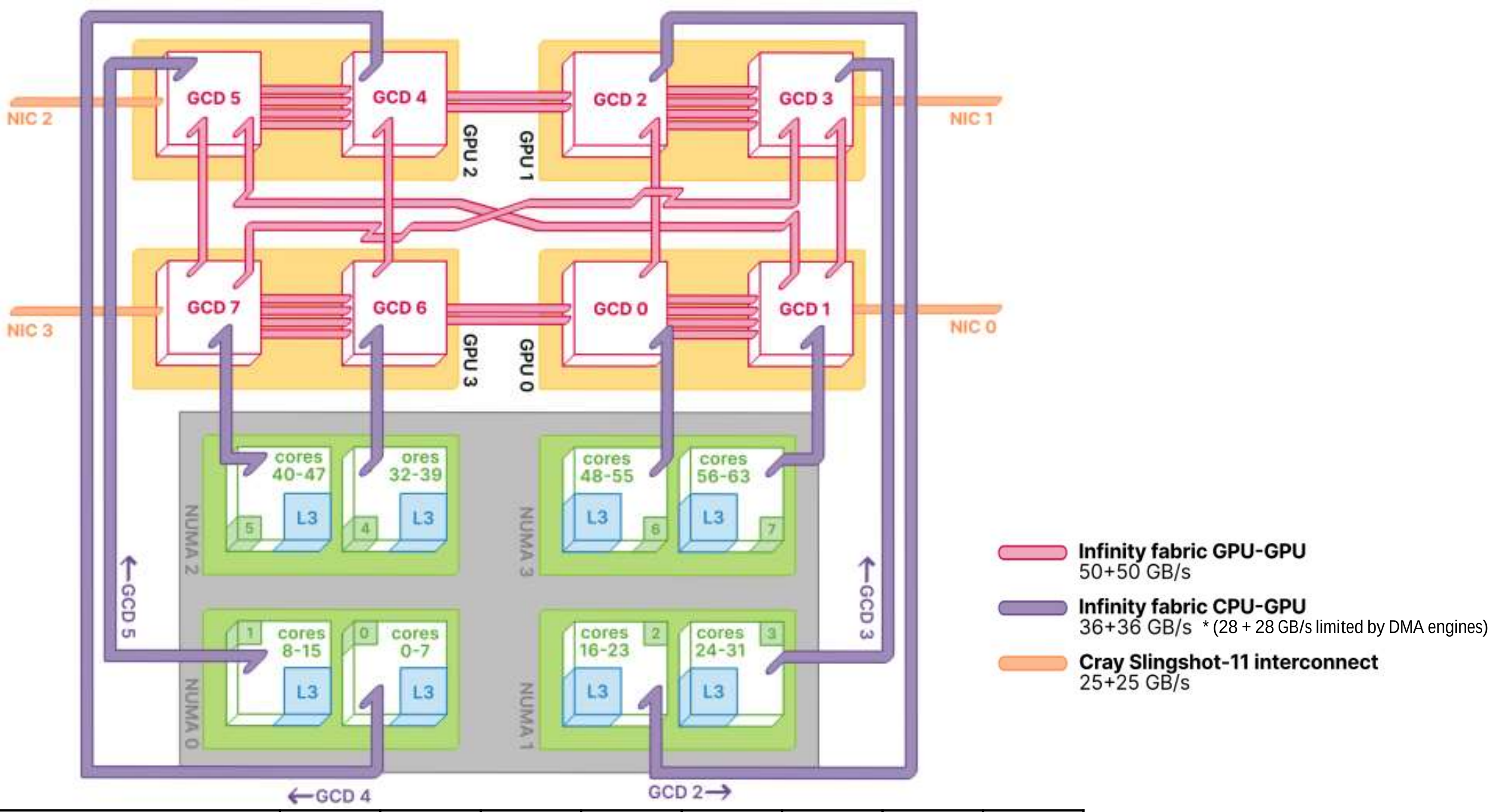
source : https://docs.lumi-supercomputer.eu/hardware/lumig/

PLACEMENT FOR GPUS – GPUs to NUMA Domains Mapping

- GPUs are associated to NUMA nodes
- Can use rocm-smi to see the topology

```
> srun --nodes=1 -t "00:10:00" --ntasks=1 --gres=gpu:8 rocm-smi –showtoponuma | grep Affinity
…
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Affinity: 2
```

PLACEMENT FOR GPUS – GPUs to NUMA Domains Mapping



NUMA ID	0	0	1	1	2	2	3	3
Optimal GPU ID	4	5	2	3	6	7	0	1

PLACEMENT FOR GPUS – Assuming one GPU per MPI Rank

- Don't use `--gpus-per-task` or `--gpu-bind` (Slurm issue with cgroups preventing to use intra-node GPU to GPU communications).
- Associate each MPI rank to a given GPU with two environment variables:
  - **`ROCR_VISIBLE_DEVICES`**
    – Limit the number of GPU devices that are available for a given process
  - **`SLURM_LOCALID`**
    – Node local task ID for the process within a job

- Remember, you can check GPU bindings with the gpu_check tool

```
MPI 000 - OMP 000 - HWT 001 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 000 - OMP 001 - HWT 002 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 000 - OMP 002 - HWT 003 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 000 - OMP 003 - HWT 004 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
…
MPI 007 - OMP 004 - HWT 060 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 007 - OMP 005 - HWT 061 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 007 - OMP 006 - HWT 062 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
MPI 007 - OMP 007 - HWT 063 - Node nid005015 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,de
```

PLACEMENT FOR GPUS – Naive Approach (1/2)

- Let’s consider the example of the 8 MPI tasks/node shown before
- Use a script to properly set `ROCR_VISIBLE_DEVICES` during the job submission

`job.slurm`

```
#!/bin/bash
#SBATCH -p <partition>
#SBATCH -A <your_project>
#SBATCH --time=00:02:00
#SBATCH --nodes=2
#SBATCH --gres=gpu:8
#SBATCH --exclusive
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=7
#SBATCH --hint=nomultithread
...
${ASRUN} ./select_gpu.sh <my_app>
```

`select_gpu.sh`

```
#!/bin/bash

export ROCR_VISIBLE_DEVICES=$SLURM_LOCALID

exec $*
```

- Naïve approach example, Each MPI rank :
  - Detects the availability of a single GPU device
  - Is essentially associated with a different GPU device to ensure that two MPI ranks do not share the same GPU device

PLACEMENT FOR GPUS – Naive Approach (2/2)

- What we want

Local Task ID	0	1	2	3	4	5	6	7
NUMA ID	0	0	1	1	2	2	3	3
Optimal GPU ID	4	5	2	3	6	7	0	1

- Naïve approach binding map

Local Task ID	0	1	2	3	4	5	6	7
Cores ID	1 - 7	9 - 15	17 - 23	25 - 31	33 - 39	41 - 47	49 - 55	57 - 63
NUMA ID	0	0	1	1	2	2	3	3
GPU ID	0	1	2	3	4	5	6	7

- This mapping is not optimal

PLACEMENT FOR GPUS – Affinity Script

- Adapt `select_gpu.sh` script for the optimal GPU ID

```
#!/bin/bash
GPUSID="4 5 2 3 6 7 0 1"
GPUSID=(${GPUSID})
if [ ${#GPUSID[@]} -gt 0 -a -n "${SLURM_NTASKS_PER_NODE}" ]; then
   if [ ${#GPUSID[@]} -gt $SLURM_NTASKS_PER_NODE ]; then
        export ROCR_VISIBLE_DEVICES=${GPUSID[$(($SLURM_LOCALID))]}
    else
        export ROCR_VISIBLE_DEVICES=${GPUSID[$(($SLURM_LOCALID / ($SLURM_NTASKS_PER_NODE / ${#GPUSID[@]})))]}
    fi
fi
exec $*
```

```
MPI 000 - OMP 000 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - GPU_ID 7 - Bus_ID de
MPI 006 - OMP 000 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - GPU_ID 1 - Bus_ID c6
```

Local Task ID	0	1	2	3	4	5	6	7
NUMA ID	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**
Optimal GPU ID	**4**	**5**	**2**	**3**	**6**	**7**	**0**	**1**

PLACEMENT FOR GPUS – Combining MPI tasks, OpenMP and GPUs (1/2)

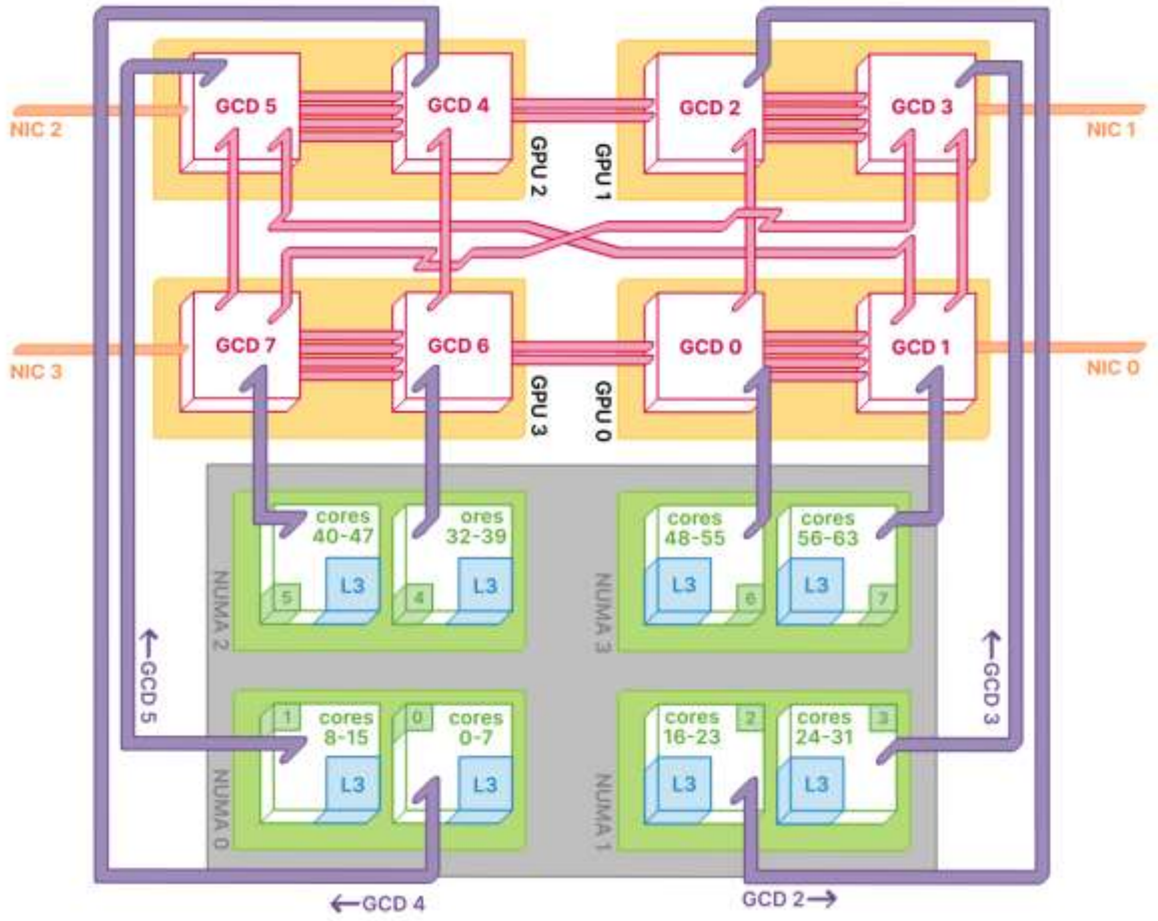
- 8 tasks/node
- 7 threads/task
- 8 GPU/node
- 1 GPU/task

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --gres=gpu:8
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=7
#SBATCH --hint=nomultithread
#SBATCH --exclusive

export OMP_PLACES=cores #replace this by sockets
export OMP_PROC_BIND=close
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun ${cpu_bind} ./select_gpu.sh gpu_check
```

```
command : gpu_check | awk '{ gsub("- Node nid[0-9]+ - RT_GPU_ID [0-9]+ - ", ""); print }'
```

PLACEMENT FOR GPUS – Combining MPI tasks, OpenMP and GPUs (2/2)

```
MPI 000 - OMP 000 - HWT 001 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 001 - HWT 002 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 002 - HWT 003 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 003 - HWT 004 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 004 - HWT 005 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 005 - HWT 006 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 000 - OMP 006 - HWT 007 (CCD0) GPU_ID 4 - Bus_ID d1(GCD4/CCD0)
MPI 001 - OMP 000 - HWT 009 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 001 - HWT 010 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 002 - HWT 011 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 003 - HWT 012 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 004 - HWT 013 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 005 - HWT 014 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 001 - OMP 006 - HWT 015 (CCD1) GPU_ID 5 - Bus_ID d6(GCD5/CCD1)
MPI 002 - OMP 000 - HWT 017 (CCD2) GPU_ID 2 - Bus_ID c9(GCD2/CCD2)
...
```



🟥	**Infinity fabric GPU-GPU** 50+50 GB/s
🟪	**Infinity fabric CPU-GPU** 36+36 GB/s
🟧	**Cray Slingshot-11 interconnect** 25+25 GB/s

PLACEMENT FOR GPUS – Mapping Processes to Network Interfaces

- On compute nodes that offer multiple GPU devices and multiple Network Interface Controllers (NIC), Cray MPI offers a flexible way to offer the ideal mapping between a process and the default NIC
  - We have a NIC per each Mi250X, i.e. 4 NICs per node

- For GPU-enabled parallel applications that involve MPI operations that access application arrays resident are on GPU-attached memory regions, users can set **`MPICH_OFI_NIC_POLICY`** to **“GPU”**
  - In this case, for each MPI process, Cray MPI strives to select a NIC device that is closest to the GPU device being used

- To display information pertaining to NIC selection:

```
export MPICH_OFI_NIC_VERBOSE=2
```

- More info in `mpi` man page

PLACEMENT FOR GPUS – Takeaways

- Don't use `--gpus-per-task` or `--gpu-bind` (Slurm issue with cgroups preventing to use intra-node GPU to GPU communications)
- Use instead a binding script to ensure proper NUMA affinity (impact transfers between Host and Device)
- When using GPU to GPU communications (different compute nodes), set MPICH_GPU_SUPPORT_ENABLED=1 and MPICH_OFI_NIC_POLICY=GPU
- Potentially reorder tasks to better harness the asymmetric interconnect between GPUs in the same node

Cheat sheet

`rocm-smi -showtopo`	Display GPU topology
`export ROCR_VISIBLE_DEVICES=<list>`	Set GPUs visible by the task (tools and applications)
`export HIP_VISIBLE_DEVICES=<list>`	Set GPUs visible by the task (HIP only)
`export MPICH_GPU_SUPPORT_ENABLED=1`	Enable MPI communications between GPUs
`export MPICH_OFI_NIC_POLICY=GPU`	Pick network interconnect closer to GCD

`export MPICH_RANK_REORDER_METHOD=3`
`export MPICH_RANK_REORDER_FILE=<file>`
	Manually defining rank reordering
	(more details in another presentation)

CONCLUSION

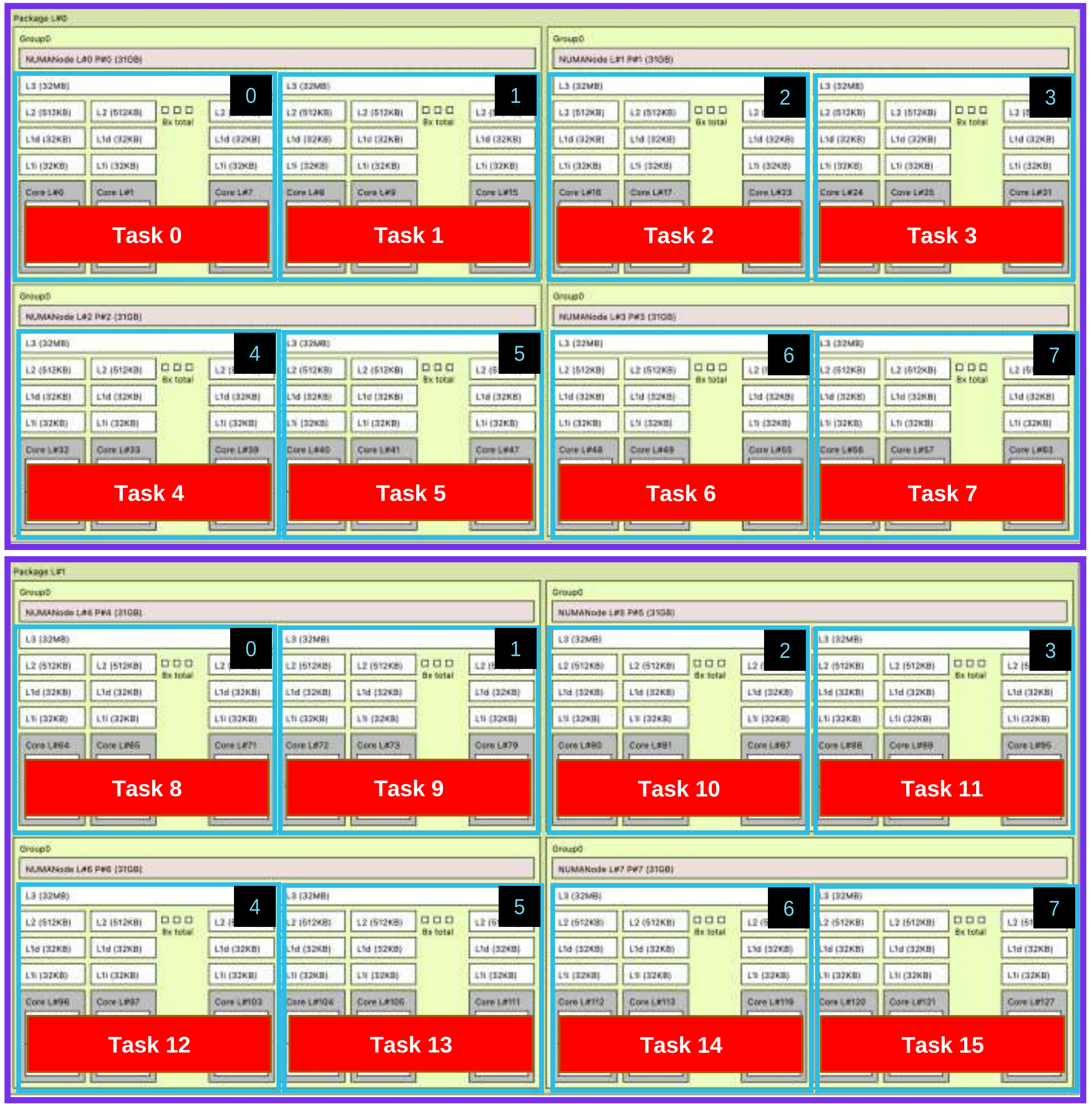
LUMI-C : OPTIMAL INITIAL PLACEMENT

2 sockets x 8 CCD = 16 MPI TASKS

each with 8 cores

```
#SBATCH --exclusive
#SBATCH --hint=nomultithread
#SBATCH --ntasks-per-node=16
#SBATCH -N X

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=close
export OMP_PLACES=cores
srun -c 8 <app>
```

(Using all cores in the node, with cores per task matching hardware)

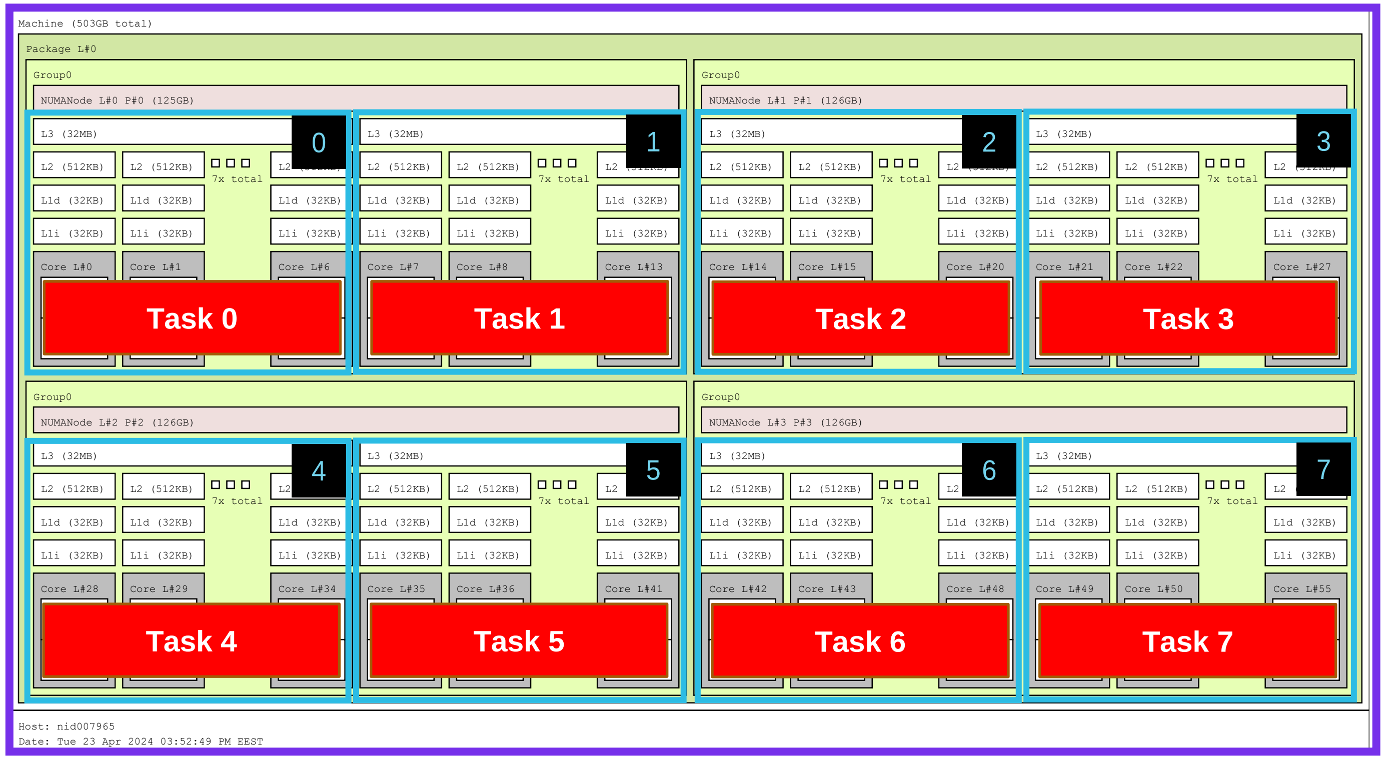


LUMI-G : OPTIMAL INITIAL PLACEMENT

1 socket x 8 CCD = 8 MPI TASKS

each with 7 cores

```
#SBATCH --exclusive
#SBATCH --ntasks-per-node=8
#SBATCH --hint=nomultithread
#SBATCH --gres=gpu:8
#SBATCH –N X

export OMP_NUM_THREADS=7
export OMP_PROC_BIND=close
export OMP_PLACES=cores
export MPICH_GPU_SUPPORT_ENABLED=1
export MPICH_OFI_NIC_POLICY=GPU

srun -c 7 ./gpu_bind_script <app>
```



(Assuming one GPU par task. Using all available cores in the node, with cores per task matching hardware)

Documentation

- Slurm documentation
- Slurm manpage

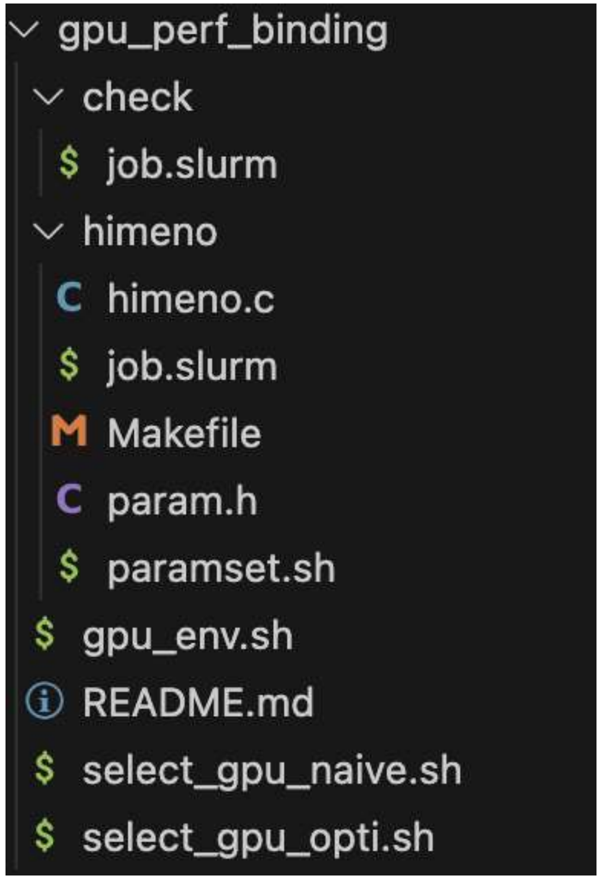
```
man srun
```

- LUMI Documentation
- ARCHER2 Documentation
- OpenMP documentation

- Just be aware that some details are site specific

Questions?

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/



```
gpu_perf_binding
  check
    job.slurm
  himeno
    himeno.c
    job.slurm
    Makefile
    param.h
    paramset.sh
  gpu_env.sh
  README.md
  select_gpu_naive.sh
  select_gpu_opti.sh
```

1) Get the right environment
- `(copy the exercices to your home directory)`
- `cd ~/<your_path>/gpu_perf_binding`
- `source ../../lumi_g.sh`
- `source gpu_env.sh`

2) Start with example in *check* directory
- `cd check`
- `sbatch job.slurm`
  - test different bindings
  - `cpu_bind, gpu_bind variables...`

3) Launch *Himeno* from the root directory
- `cd himeno && make`
- `sbatch job.slurm`
  - test different bindings

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/ (1/5)

check directory

```
Default run. Binding issues?

MPI 000 - OMP 000 - HWT 001 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 000 - OMP 001 - HWT 001 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 001 - OMP 000 - HWT 009 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 001 - OMP 001 - HWT 009 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 002 - OMP 000 - HWT 017 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 002 - OMP 001 - HWT 017 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 003 - OMP 001 - HWT 025 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 004 - OMP 001 - HWT 033 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 005 - OMP 000 - HWT 041 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 005 - OMP 001 - HWT 041 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 006 - OMP 000 - HWT 049 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 006 - OMP 001 - HWT 049 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 007 - OMP 000 - HWT 057 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
MPI 007 - OMP 001 - HWT 057 - Node nid005177 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
```

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/ (2/5)

check directory

```
Enabling CPU masks for OpenMP threads and optimal GPU bindings:

MPI 000 - OMP 000 - HWT 001 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 000 - OMP 001 - HWT 002 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 001 - OMP 001 - HWT 010 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 002 - OMP 001 - HWT 018 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 003 - OMP 001 - HWT 026 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 004 - OMP 001 - HWT 034 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
MPI 005 - OMP 001 - HWT 042 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
MPI 006 - OMP 000 - HWT 049 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 006 - OMP 001 - HWT 050 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
MPI 007 - OMP 001 - HWT 058 - Node nid005174 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/ (3/5)

himeno directory

Default run:

```
MPI 000 - OMP 000 - HWT 001 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 001 - OMP 000 - HWT 002 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 002 - OMP 000 - HWT 003 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 003 - OMP 000 - HWT 004 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 004 - OMP 000 - HWT 005 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 005 - OMP 000 - HWT 006 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 006 - OMP 000 - HWT 007 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
MPI 007 - OMP 000 - HWT 009 - Node nid005175 - RT_GPU_ID 0,1,2,3,4,5,6,7 - GPU_ID 0,1,2,3,4,5,6,7 - Bus_ID c1,c6,c9,ce,d1,d6,d9,dc
```

```
cpu : 50.143480 sec.
Loop executed for 50 times
Gosa : 8.030980e-05
MFLOPS measured : 9012.093693
Score based on Pentium III 600MHz : 108.789156
```

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/ (4/5)

himeno directory

Adding GPU binding:

```
MPI 000 - OMP 000 - HWT 001 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 002 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 003 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 004 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 005 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 006 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
MPI 006 - OMP 000 - HWT 007 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 009 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6

cpu : 5.642638 sec.
Loop executed for 50 times
Gosa : 7.531330e-04
MFLOPS measured : 80086.260270
Score based on Pentium III 600MHz : 966.758333
```

/project/project_465001098/Exercises/HPE/day2/gpu_perf_binding/ (5/5)

himeno directory

Adding CPU binding and GPU binding:

```
MPI 000 - OMP 000 - HWT 001 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 4 - Bus_ID d1
MPI 001 - OMP 000 - HWT 009 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 5 - Bus_ID d6
MPI 002 - OMP 000 - HWT 017 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 2 - Bus_ID c9
MPI 003 - OMP 000 - HWT 025 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 3 - Bus_ID ce
MPI 004 - OMP 000 - HWT 033 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 6 - Bus_ID d9
MPI 005 - OMP 000 - HWT 041 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 7 - Bus_ID dc
MPI 006 - OMP 000 - HWT 049 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 0 - Bus_ID c1
MPI 007 - OMP 000 - HWT 057 - Node nid005173 - RT_GPU_ID 0 - GPU_ID 1 - Bus_ID c6
```

```
cpu : 5.302780 sec.
Loop executed for 50 times
Gosa : 9.317707e-04
MFLOPS measured : 85219.030186
Score based on Pentium III 600MHz : 1028.718375
```