Creating
World
Changing
Technologies

Multi-GPU Programming - Scale-Up and Scale-Out made easy, using the Intel® MPI Library

Anatoliy Rozanov, Dmitry Durnov, Michael Steyer
Intel

intel®

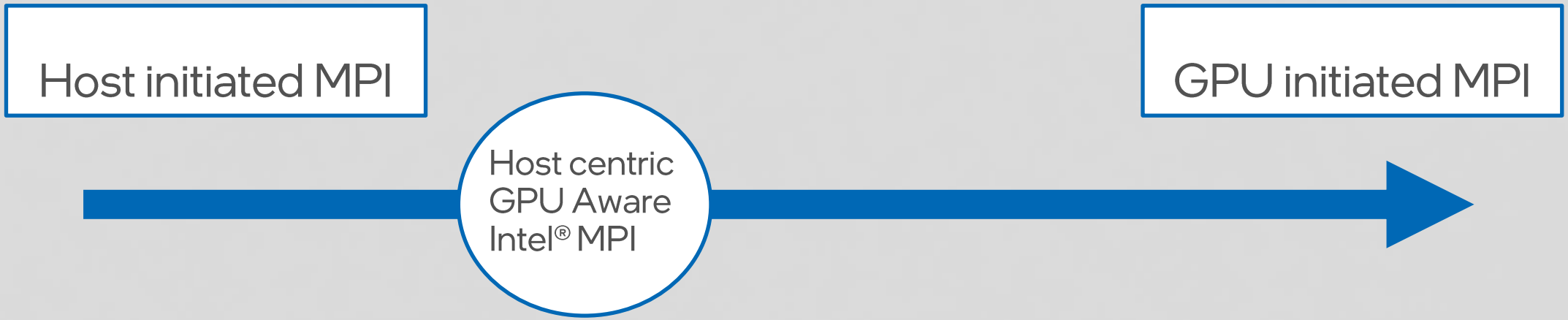
Agenda

- GPU Programming model
- Intel® MPI GPU Pinning
- Intel® MPI GPU Buffers Support
- Demo
- Questions

GPU Programming model

- MPI primitives can be used for communications between GPUs
- Host centric MPI with GPU awareness:
  - Host initiated communication:
    - Pipelining or dma_buf (direct GPU memory access)
    - Local GPU to GPU communication
    - GPU kernels for collective operations (MPI_Allreduce, MPI_Reduce)
- GPU centric MPI:
  - GPU kernel-initiated communication

GPU Programming model. Where are we now?



- GPU buffers support
- Pipelining (in progress)
- Allreduce GPU kernel (in progress)

Intel® MPI Simplifies Programming for GPUs*

- Multiple GPUs with multiple Tiles require manual domain decomposition or implicit scaling mechanism which may not be always desirable
- Intel MPI enables single GPU / Tile programming while leaving distribution to the library
- Intel MPI allows to assign GPUs / Tiles to individual MPI ranks
- Intel MPI allows users to pass GPU memory pointers to the library

*Intel GPGPUs

Intel® MPI GPU Pinning

Intel GPU pinning support

- Automatic Intel GPU resources distribution
- No user code changes required on different GPU configurations
- NUMA aware

Default settings:

I_MPI_OFFLOAD_* family:

TOPOLIB=level_zero
CELL=tile
DOMAIN_SIZE=-1 (auto)
DEVICES=all

E.g:
I_MPI_OFFLOAD_TOPOLIB=level_zero

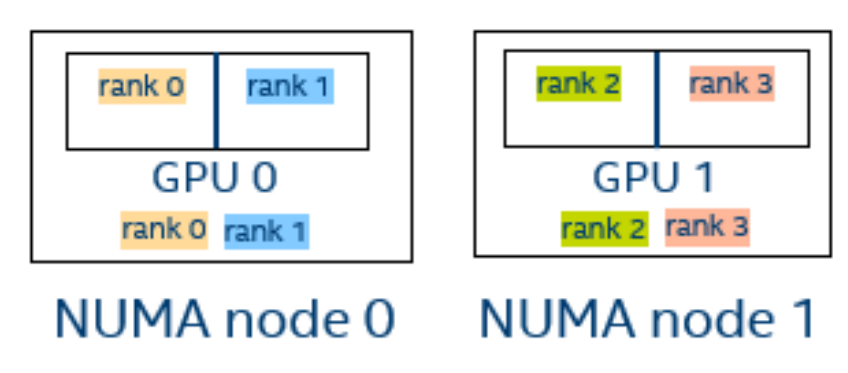
Example – Default: 4 ranks

I_MPI_DEBUG=120

```
[0] MPI startup(): ===== GPU topology on host1 =====
[0] MPI startup():  NUMA nodes : 2
[0] MPI startup():  GPUs       : 2
[0] MPI startup():  Tiles      : 4
[0] MPI startup(): =====  GPU Placement on packages  =====
[0] MPI startup(): NUMA Id    GPU Id  Tiles     Ranks
[0] MPI startup(): 0          0       (0,1)     0,1
[0] MPI startup(): 1          1       (2,3)     2,3
```

I_MPI_DEBUG=3

```
[0] MPI startup(): ===== GPU pinning on host1 =====
[0] MPI startup(): Rank Pin tile
[0] MPI startup():  0    {0}
[0] MPI startup():  1    {1}
[0] MPI startup():  2    {2}
[0] MPI startup():  3    {3}
```

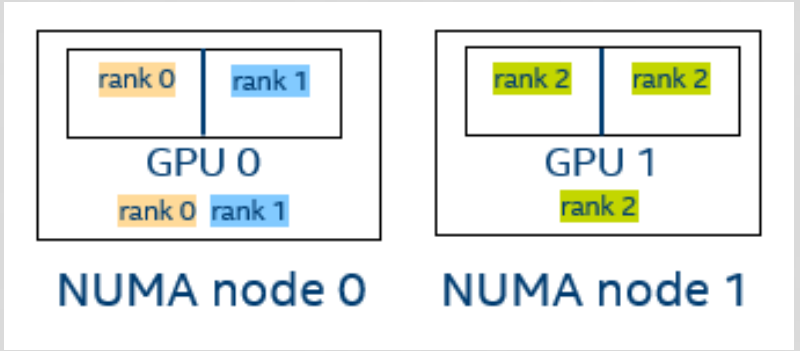


Example – Default: 3 ranks

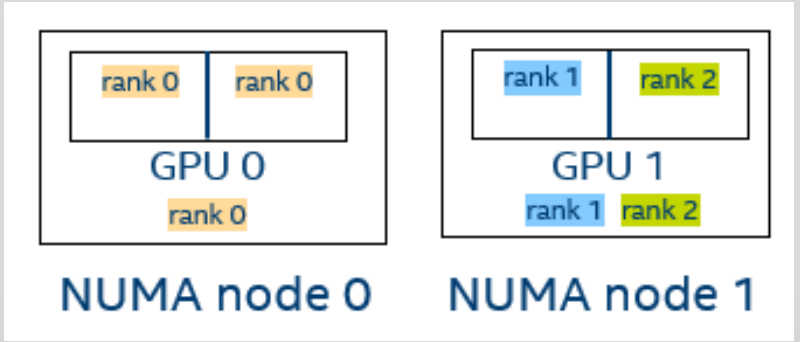
I_MPI_DEBUG=3

I_MPI_OFFLOAD_DOMAIN_SIZE=-1 (auto) [default]

```
[0] MPI startup(): ===== GPU pinning on host1 =====
[0] MPI startup(): Rank Pin tile
[0] MPI startup():  0      {0}
[0] MPI startup():  1      {1}
[0] MPI startup():  2      {2,3}
```




```
[0] MPI startup(): ===== GPU pinning on host1 =====
[0] MPI startup(): Rank Pin tile
[0] MPI startup():  0      {0,1}
[0] MPI startup():  1      {2}
[0] MPI startup():  2      {3}
```



Example – I_MPI_OFFLOAD_CELL

I_MPI_OFFLOAD_CELL=device

I_MPI_DEBUG=3

[0] MPI startup(): ===== GPU pinning on host1 =====

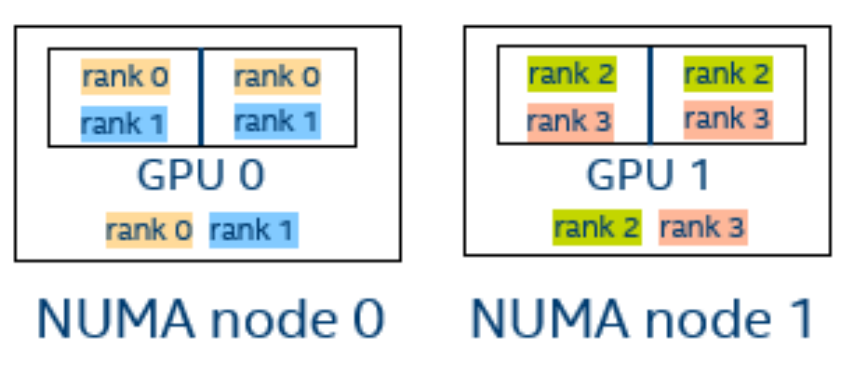
[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0,1}

[0] MPI startup(): 1 {0,1}

[0] MPI startup(): 2 {2,3}

[0] MPI startup(): 3 {2,3}



Example – I_MPI_OFFLOAD_DOMAIN_SIZE

I_MPI_OFFLOAD_DOMAIN_SIZE=1

I_MPI_DEBUG=3

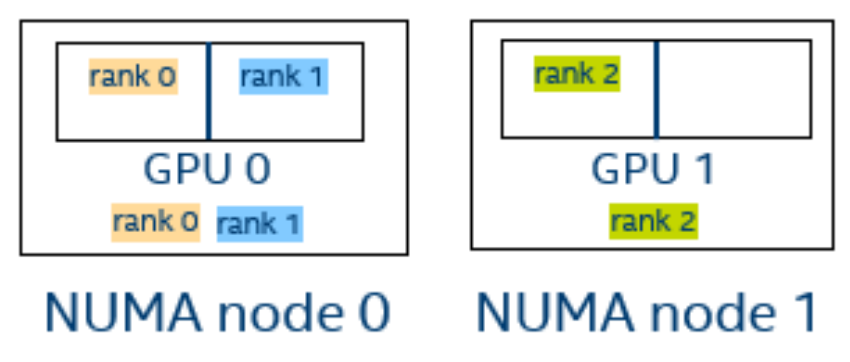
[0] MPI startup(): ===== GPU pinning on host1 =====

[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0}

[0] MPI startup(): 1 {1}

[0] MPI startup(): 2 {2}



Example – I_MPI_OFFLOAD_DEVICES

I_MPI_OFFLOAD_DEVICES=0

I_MPI_DEBUG=3

[0] MPI startup(): ===== GPU pinning on host1 =====

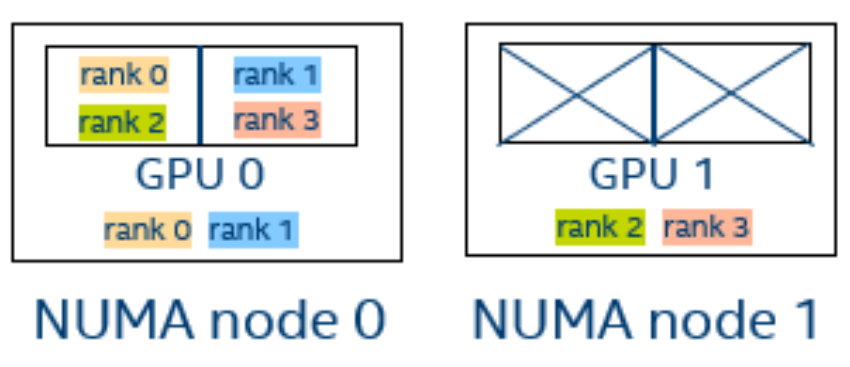
[0] MPI startup(): Rank Pin tile

[0] MPI startup(): 0 {0}

[0] MPI startup(): 1 {1}

[0] MPI startup(): 2 {0}

[0] MPI startup(): 3 {1}

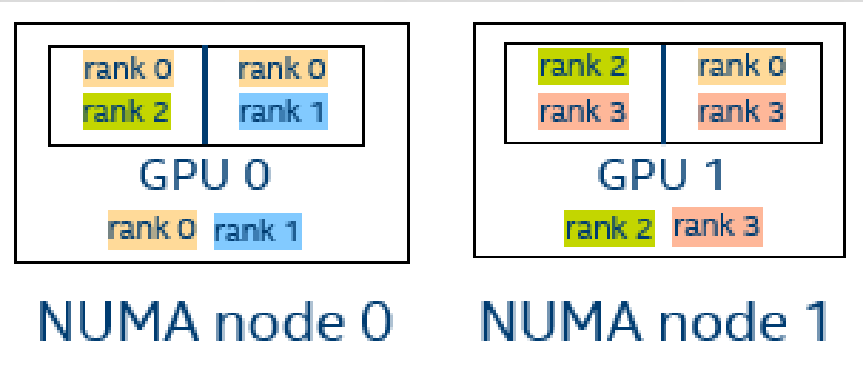


Example – I_MPI_OFFLOAD_DOMAIN

I_MPI_OFFLOAD_DOMAIN=[B,2,5,C]

reverse bit masks: [1101,0100,1010,0011]

I_MPI_DEBUG=3




```
[0] MPI startup(): ===== GPU pinning on host1 =====
[0] MPI startup(): Rank Pin tile
[0] MPI startup():  0     {0,1,3}
[0] MPI startup():  1     {1}
[0] MPI startup():  2     {0,2}
[0] MPI startup():  3     {2,3}
```

Intel® MPI GPU Buffers Support

Explicitly enable Offloading Library

Syntax

I_MPI_OFFLOAD=<value>

Arguments

0 - Disabled (default value)

1 - Auto, expects that libze_loader.so is already loaded

2 - Enabled. Intel MPI loads libze_loader.so

Execution models

1. Naïve (w/ OpenMP using map(tofrom) clause)

```
// Copy data from host to device
#pragma omp target data map(to: rank, num_values) map(tofrom: values[0:num_values])
{
    // Compute on GPU
    #pragma omp target parallel for
    for (i = 0; i < num_values; i++) {
        values[i] = values[i] + rank + 1;
    }
}

// Communicate on CPU: Send buffer to rank 0 after copy back from GPU to host
MPI_Send(values, num_values, MPI_INT, dest_rank, tag, MPI_COMM_WORLD);
```

Execution models

2. GPU buffer aware (w/ openMP using pointers)

```
// Copy data from host to device
#pragma omp target data map(to: rank, num_values, values[0:num_values]) use_device_ptr(values)
{
    // Compute on GPU
    #pragma omp target parallel for is_device_ptr(values)
    for (i = 0; i < num_values; i++) {
        values[i] = values[i] + rank + 1;
    }

    // Communicate on CPU: Send buffer to rank 0 without copy back from GPU
    MPI_Send(values, num_values, MPI_INT, dest_rank, tag, MPI_COMM_WORLD);
}
```

Execution models

- Build
  - C language:

    $ mpiicc -cc=icx -fiopenmp -fopenmp-targets=spir64 test.c -o test
  - Fortran language:

    $ mpiifort -fc=ifx -fiopenmp -fopenmp-targets=spir64 test.f90 -o test
- Execute

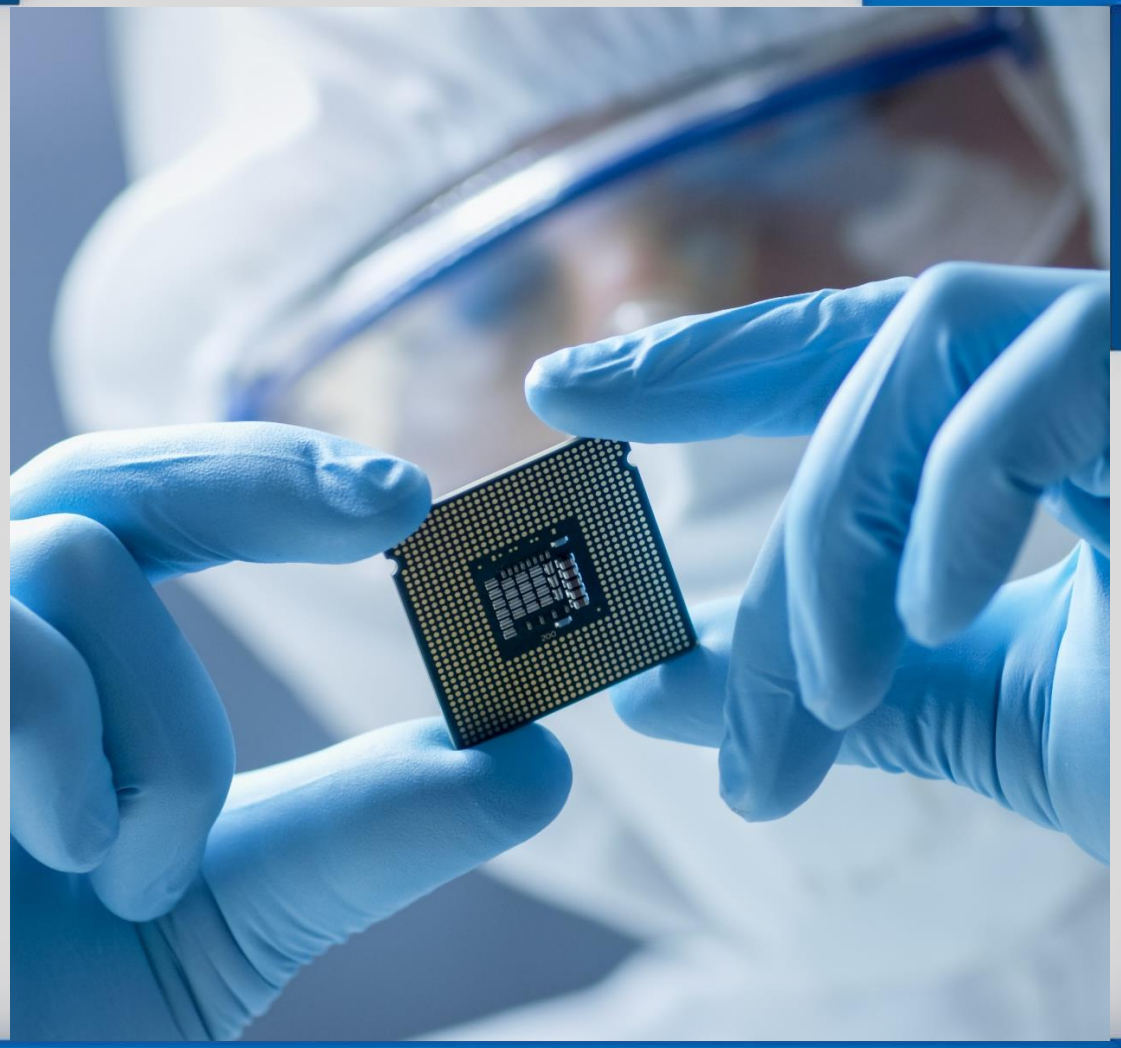
  $ I_MPI_OFFLOAD=2 mpirun -n 2 ./test

Conclusion

- «GPU Pinning» and «GPU Buffers Support» features in Intel® MPI can significantly simplify programming for GPUs
- Reference: https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/gpu-support.html

Demo

Questions?




intel