oneAPI

Multiarchitecture Programming for Accelerated Compute, Freedom of Choice for Hardware

Intel® oneAPI HPC Toolkit

MPI & Multi-GPU Programming

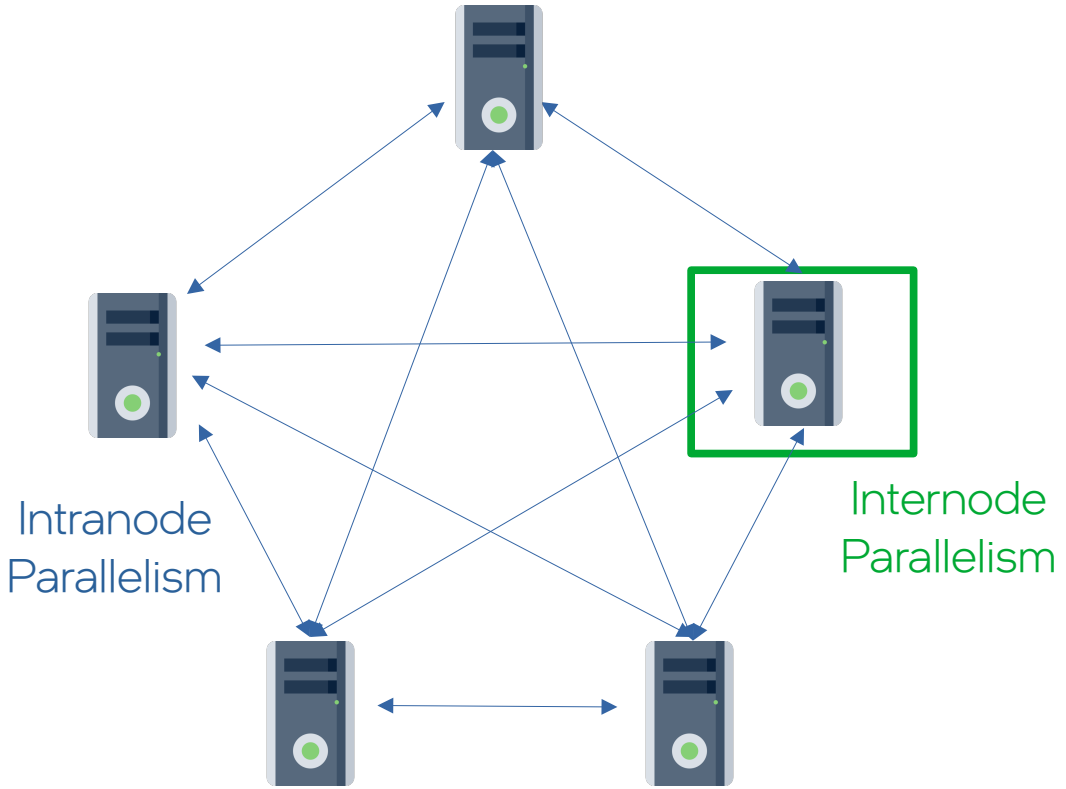
Dr. Rafael Lago

intel

Introduction to MPI

Message Passing Interface

- interface for parallel computing architectures
- defines syntax & semantics
- implementations: MPICH, OpenMPI, Intel® MPI, etc
- managed by multinational & multi-organization consortium: *https://www.mpi-forum.org/*
- Intel® Bindings for C, C++, Fortran and Java*

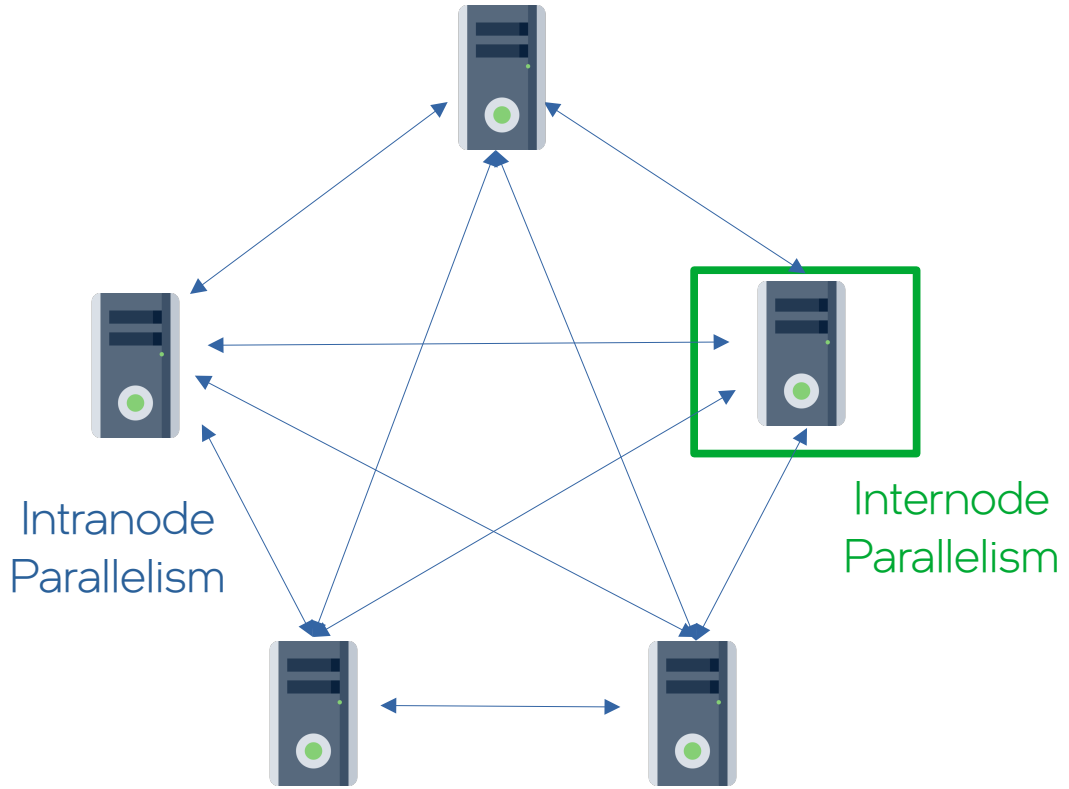


*Coding icons created by Freepik – Flaticon: *https://www.flaticon.com/free-icons/cpu-tower*

Message Passing Interface

- devised for intranode ("distributed memory")
- works with internode ("shared memory")
- works well with internode paradigma (OpenMP, SYCL, etc)
- Intel® MPI implementation:

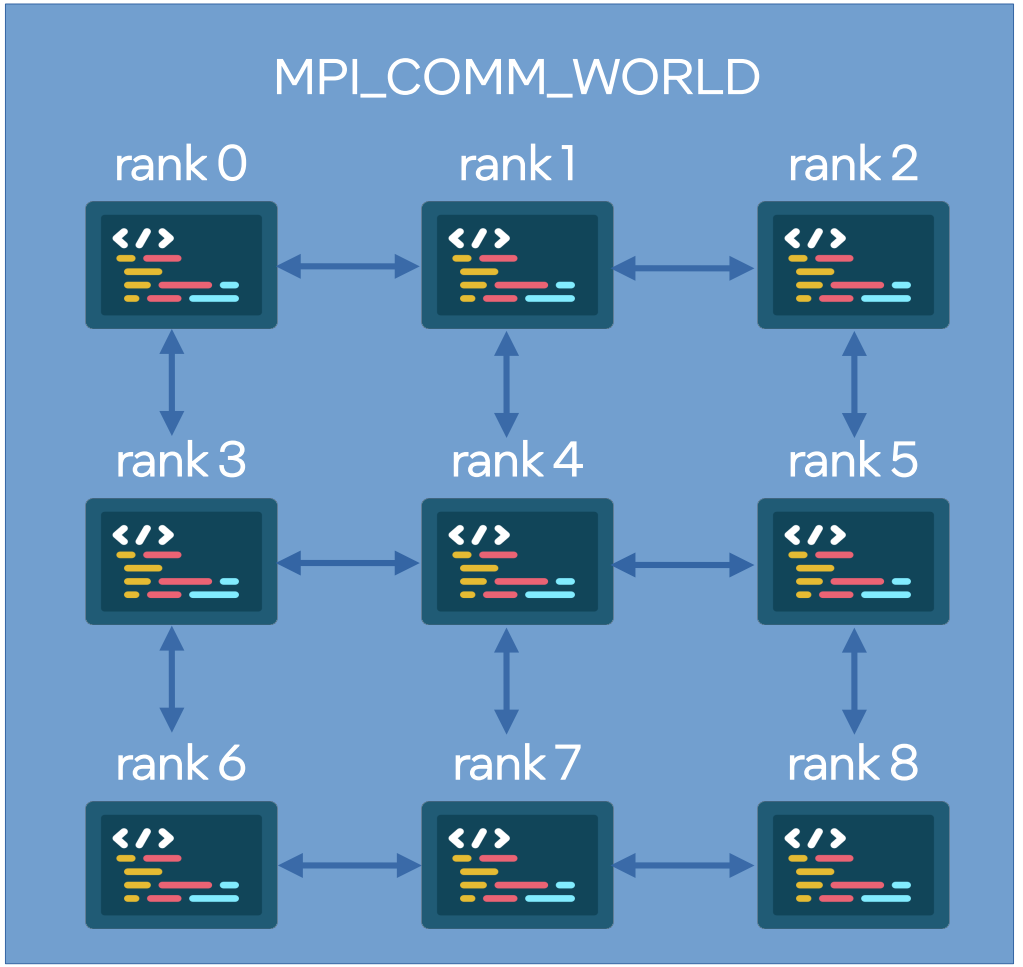
  "distinguishes host memory from accelerator memory"



*Coding icons created by Freepik – Flaticon: https://www.flaticon.com/free-icons/cpu-tower

Message Passing Interface

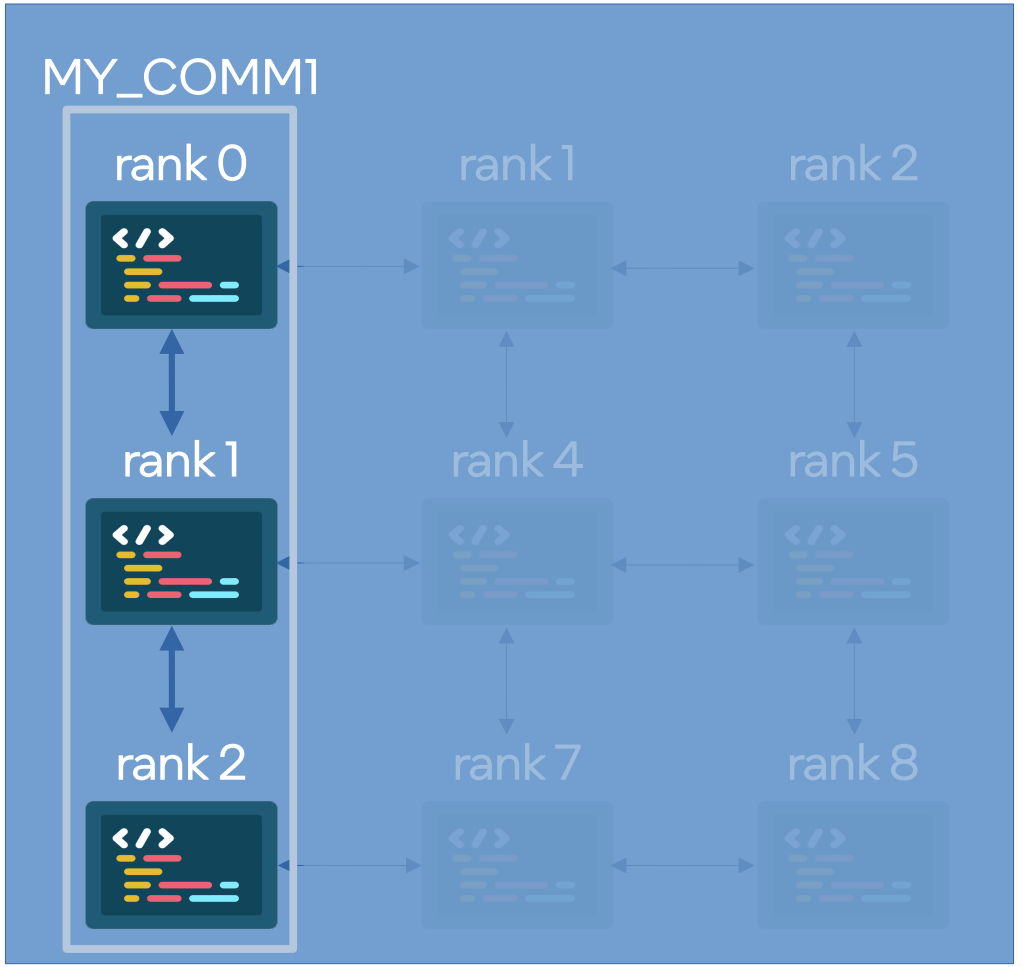
- A **communicator** contains ranks
  - Typically starts with MPI_COMM_WORLD
- each **rank** is an instance of a program
  - own memory space
  - own PID
  - own pinning (one or more threads/cores/accelerators)



*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

- A **communicator** can be split into complex topologies
- each **communicator** has its own rank numbering

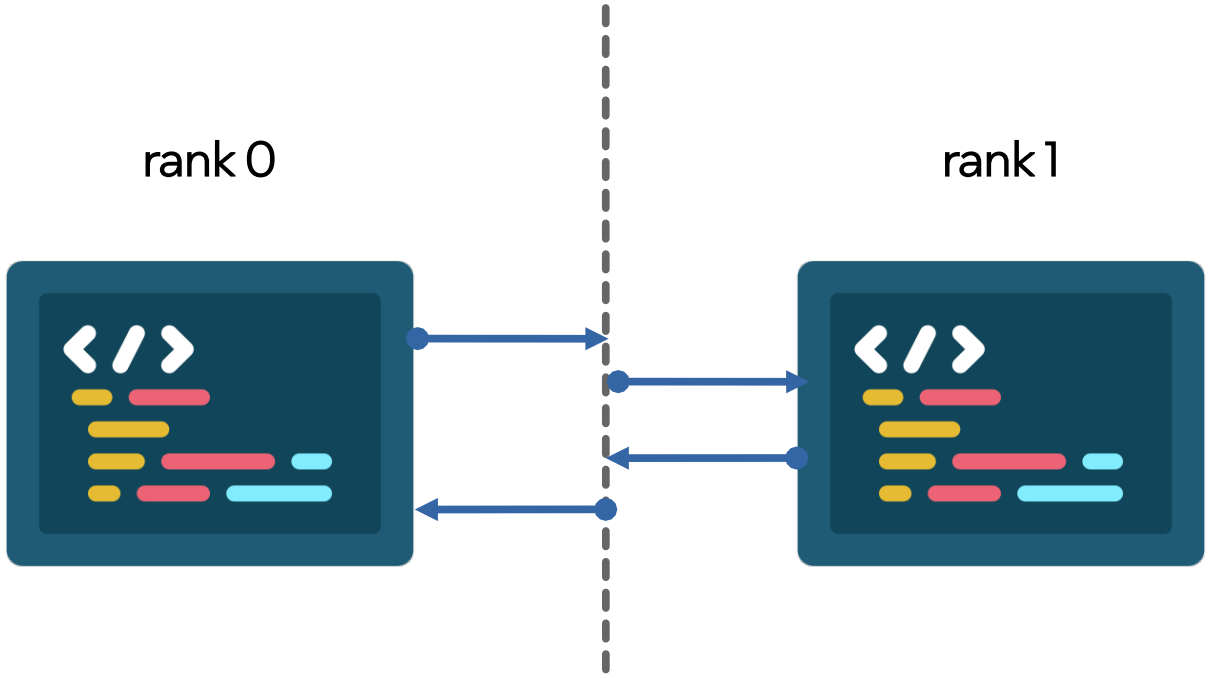


*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

Two-sided Communication

```
use mpi
integer :: myrank, ierr
real :: mydata(1024), otherdata(1024)

call MPI_Init(ierr)
call MPI_Comm_rank(myrank, MPI_COMM_WORLD, ierr)

call compute_mydata(mydata)

if (myrank == 0) then
   call MPI_Send(mydata, 1024, MPI_REAL, 1,…)
   call MPI_Recv(otherdata, 1024, MPI_REAL, 1,…)
else
   call MPI_Recv(otherdata, 1024, MPI_REAL, 0,…)
   call MPI_Send(mydata, 1024, MPI_REAL, 0,…)
end if
```



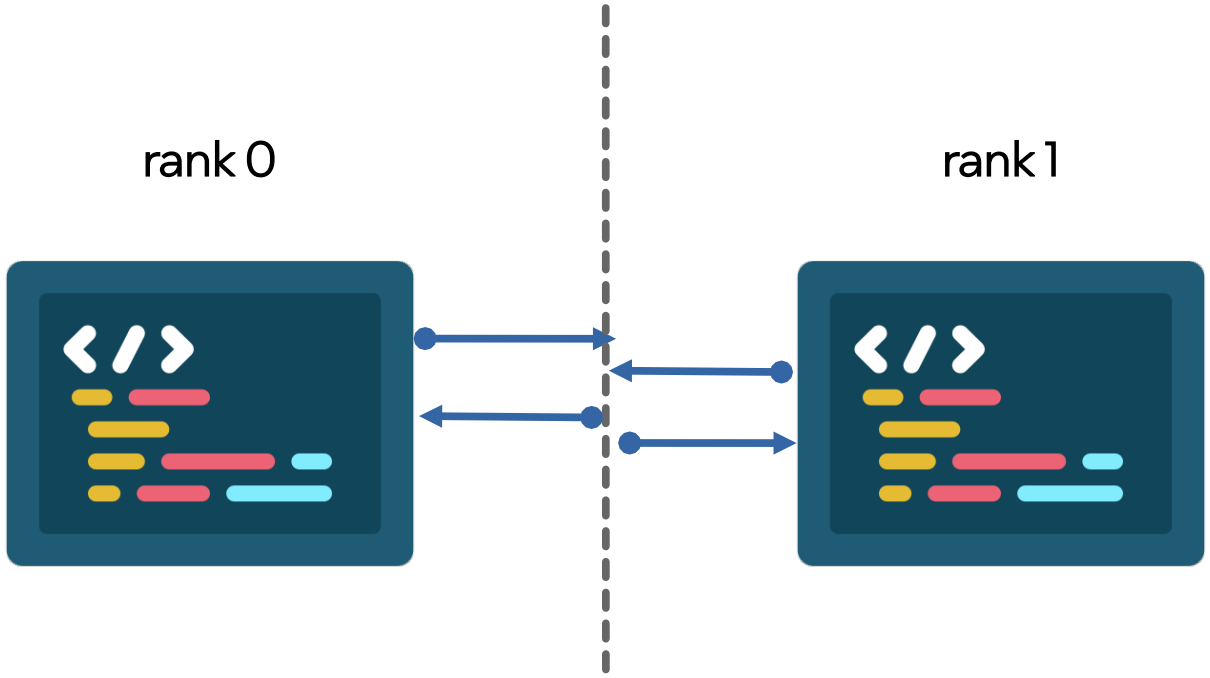
Disclaimer: this slide contains a vague representation of the process for educational purposes only!

*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

Two-sided Communication

```
use mpi
integer :: myrank, n, ierr
real :: mydata(1024), otherdata(1024)

call MPI_Init(ierr)
call MPI_Comm_rank(myrank, MPI_COMM_WORLD, ierr)
n = mod(myrank+1,2)

call compute_mydata(mydata)

call MPI_Sendrecv(mydata, 1024, MPI_REAL, n,…
                  otherdata, 1024, MPI_REAL, n,…)
```



Disclaimer: this slide contains a vague representation of the process for educational purposes only!

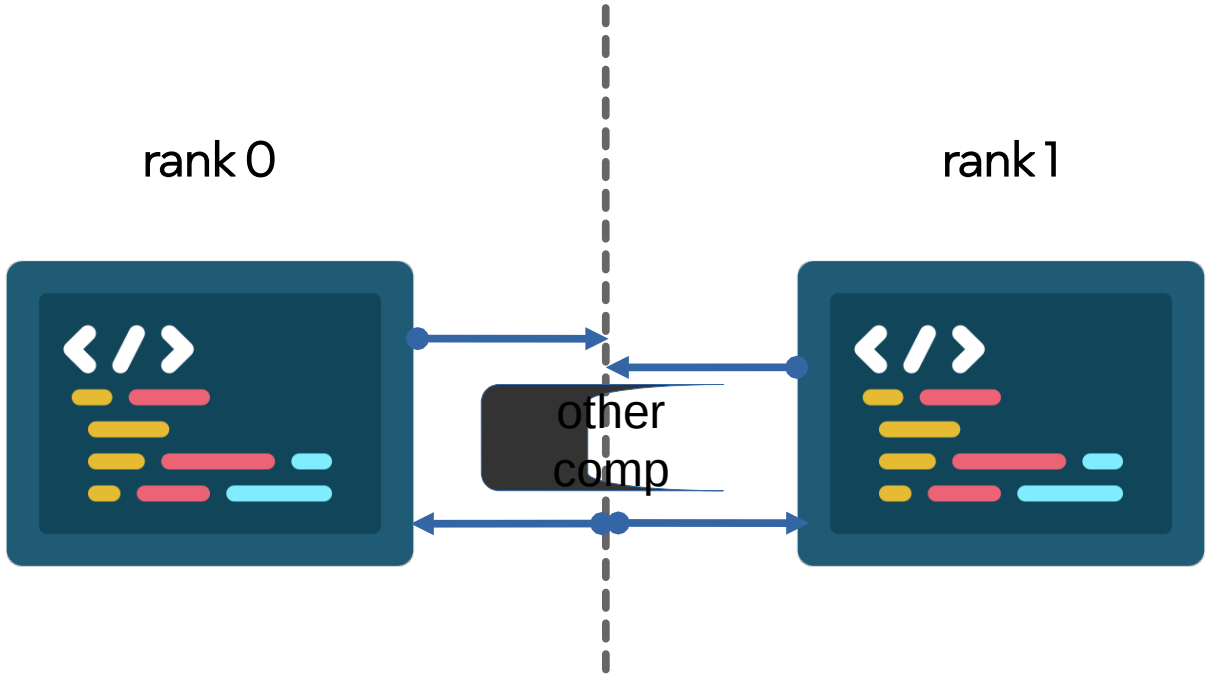
*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

Two-sided Communication - Non-Blocking

Asynchronous progress is not guaranteed by the MPI standard! Intel® MPI has special flags to enable that!

```
use mpi
integer :: myrank, n, ierr
real :: mydata(1024), otherdata(1024)

call MPI_Init(ierr)
call MPI_Comm_rank(myrank, MPI_COMM_WORLD, ierr)
n = mod(myrank+1,2)

call compute_mydata(mydata)

call MPI_Isend(mydata, …, Rq(1), ierr)
call MPI_Irecv(otherdata, …,Rq(2), ierr)

call other_computation()

call MPI_Waitall(2, Rq, …)
```



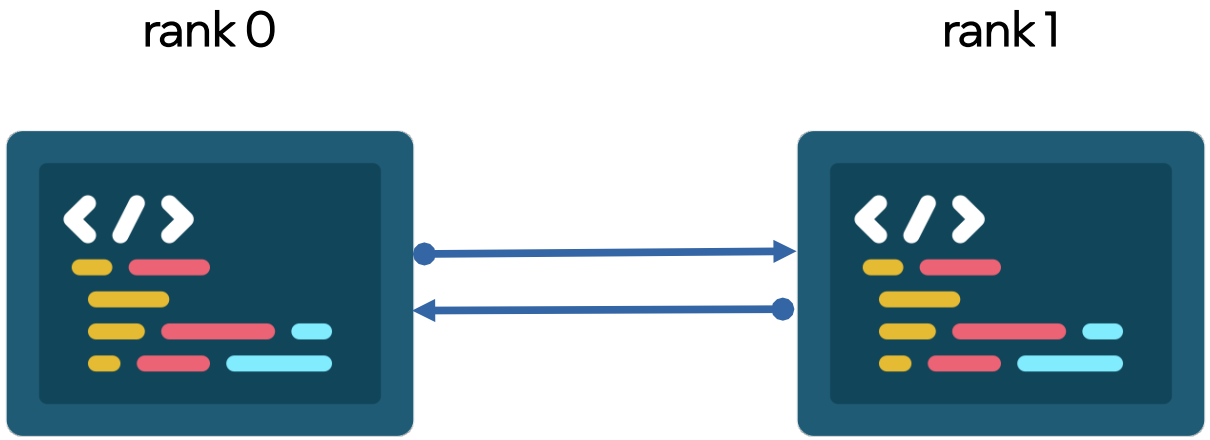
Disclaimer: this slide contains a vague representation of the process for educational purposes only!

*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

One-Sided Communication ,"Remote Memory Access", RMA

```
use mpi
integer :: myrank, n, ierr
real :: mydata(1024), otherdata(1024)

call MPI_Init(ierr)
call MPI_Comm_rank(myrank, MPI_COMM_WORLD, ierr)
n = mod(myrank+1,2)

Call MPI_Win_create(otherdata,…, otherwin, ierr)

call compute_mydata(mydata)

call MPI_Put(mydata, …, n, 1024, MPI_REAL,
             otherwin, ierr)
```



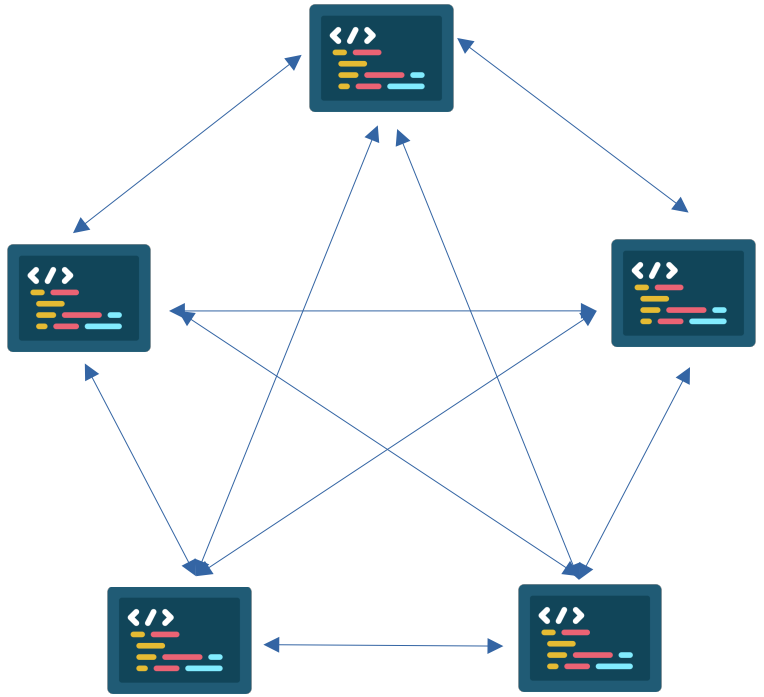
Disclaimer: this slide contains a vague representation of the process for educational purposes only!

*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Message Passing Interface

Collectives

```
use mpi
integer :: myrank, ierr
real :: mydata(1024), otherdata(1024)

call MPI_Init(ierr)
call MPI_Comm_rank(myrank, MPI_COMM_WORLD, ierr)

call compute_mydata(mydata)

call MPI_Alltoall(mydata, …, otherdata,…,
                  MPI_COMM_WORLD, ierr)

or

call MPI_IAlltoall(mydata, …, otherdata,…,
                   MPI_COMM_WORLD, Rq, ierr)
```



Disclaimer: this slide contains a vague representation of the process for educational purposes only!

*Coding icons created by juicy_fish – Flaticon: https://www.flaticon.com/free-icons/coding

Intel® MPI Library

Intel® oneAPI Tools for HPC

Intel® oneAPI HPC Toolkit

Deliver Fast Applications that Scale

What is it?

A toolkit that adds to the Intel® oneAPI Base Toolkit for building high-performance, scalable parallel code on C++, Fortran, SYCL, OpenMP & MPI from enterprise to cloud, and HPC to AI applications.

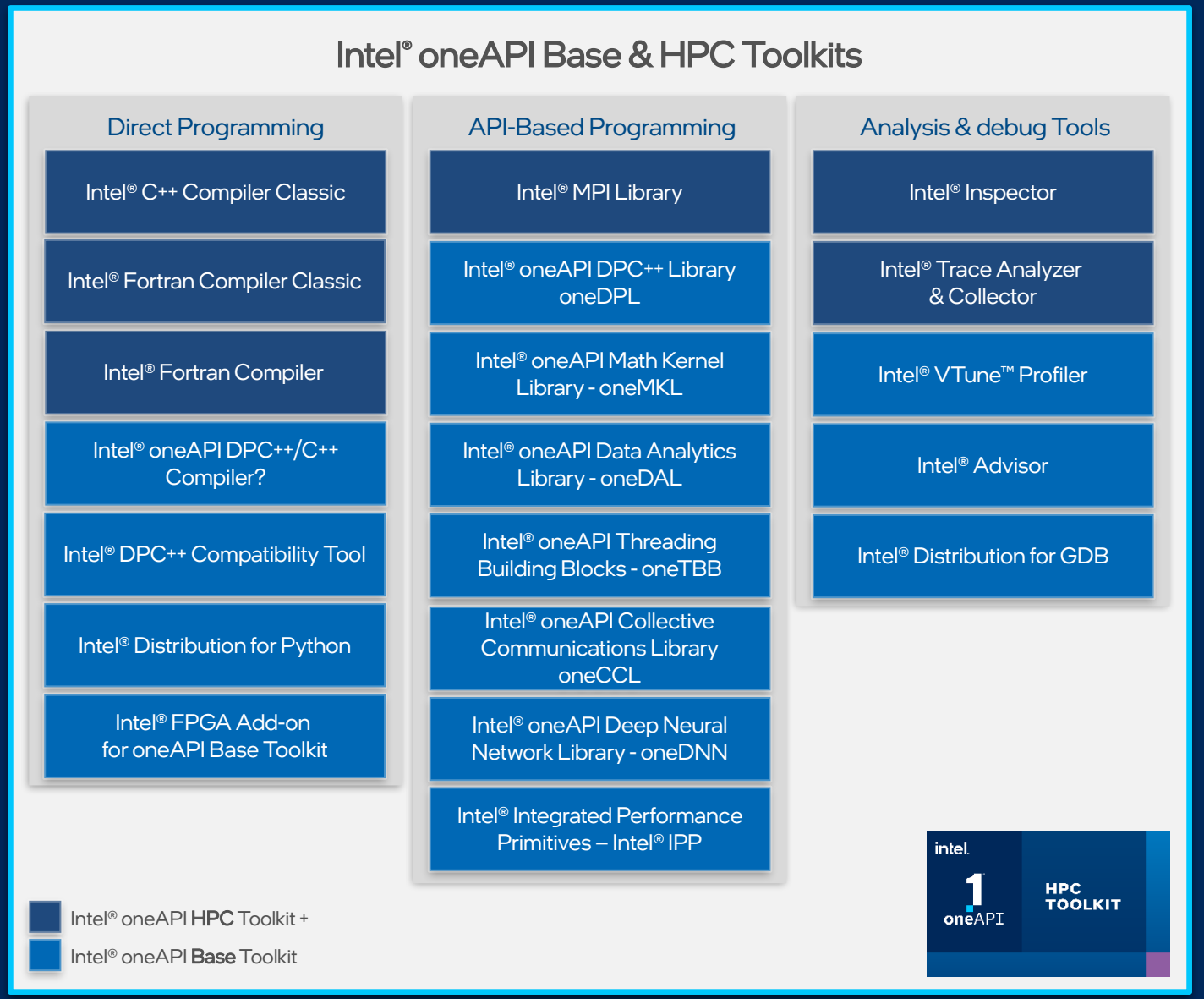
Who needs this product?

- OEMs/ISVs
- C++, Fortran, OpenMP, MPI Developers

Why is this important?

- Accelerate performance on Intel® Xeon® & Core™ processors & Intel accelerators
- Deliver fast, scalable, reliable parallel code with less effort built on industry standards

Learn More & Download



Intel® MPI Library

Key Features:

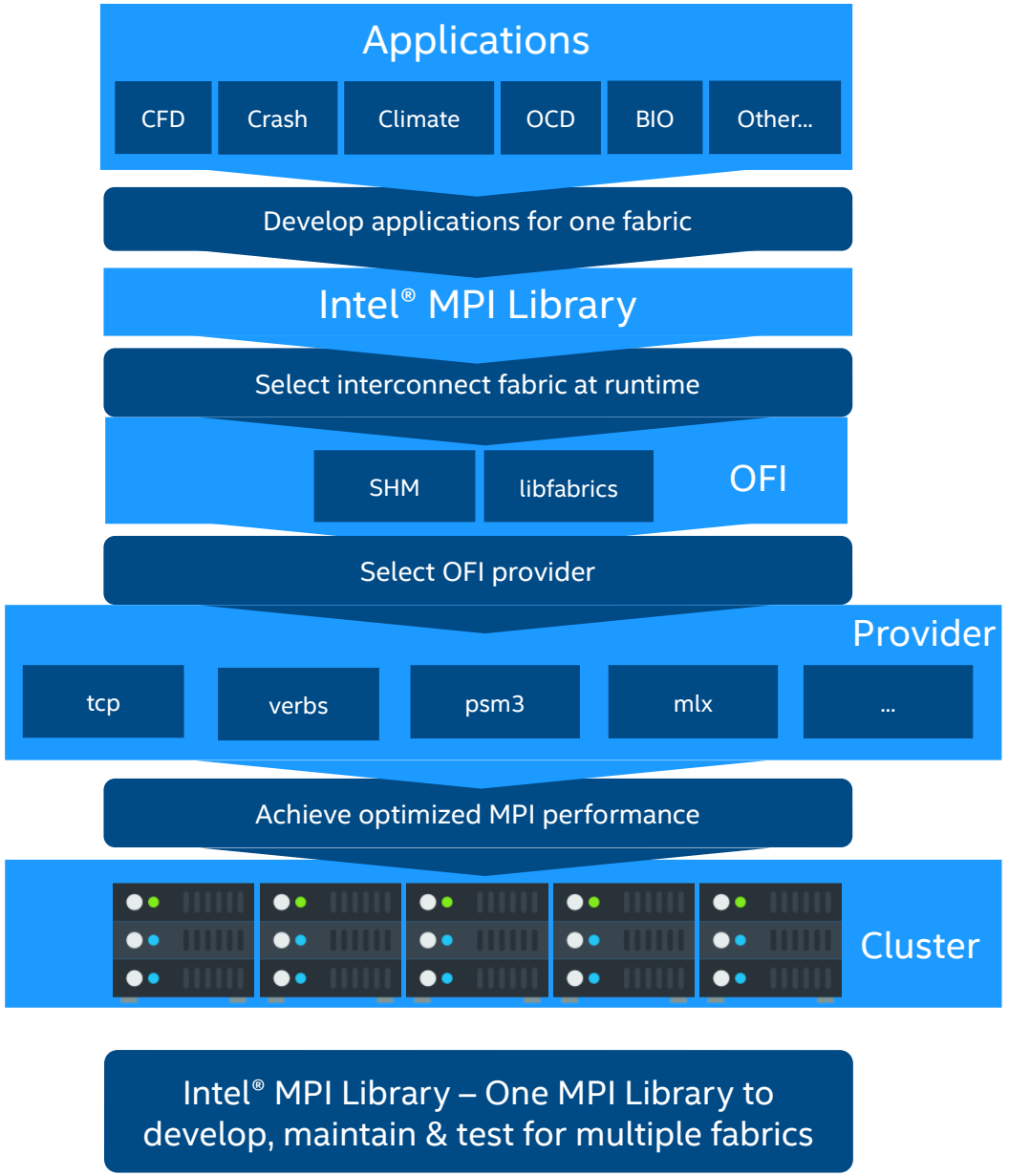
- MPI-1, MPI-2.2 and MPI-3.1 specification conformance
- Interconnect independence
- C, C++, Fortran 77, 90 & 2008 language bindings
- Amazon* AWS/EFA, Google* GCP support
- Intel GPU pinning & buffers support

2021.8 Release notes:

- initial support for the Intel® Data Center GPU MAX Series (formerly code-named Ponte Vecchio) utilizing XE Link for direct GPU to GPU communications
- enabled the new embedded Data Streaming Accelerator in 4th Generation Xeon Scalable Processors (formerly code-named Sapphire Rapids).
- performance optimizations for Intel GPUs and Intel® Xeon® CPU Max Series.

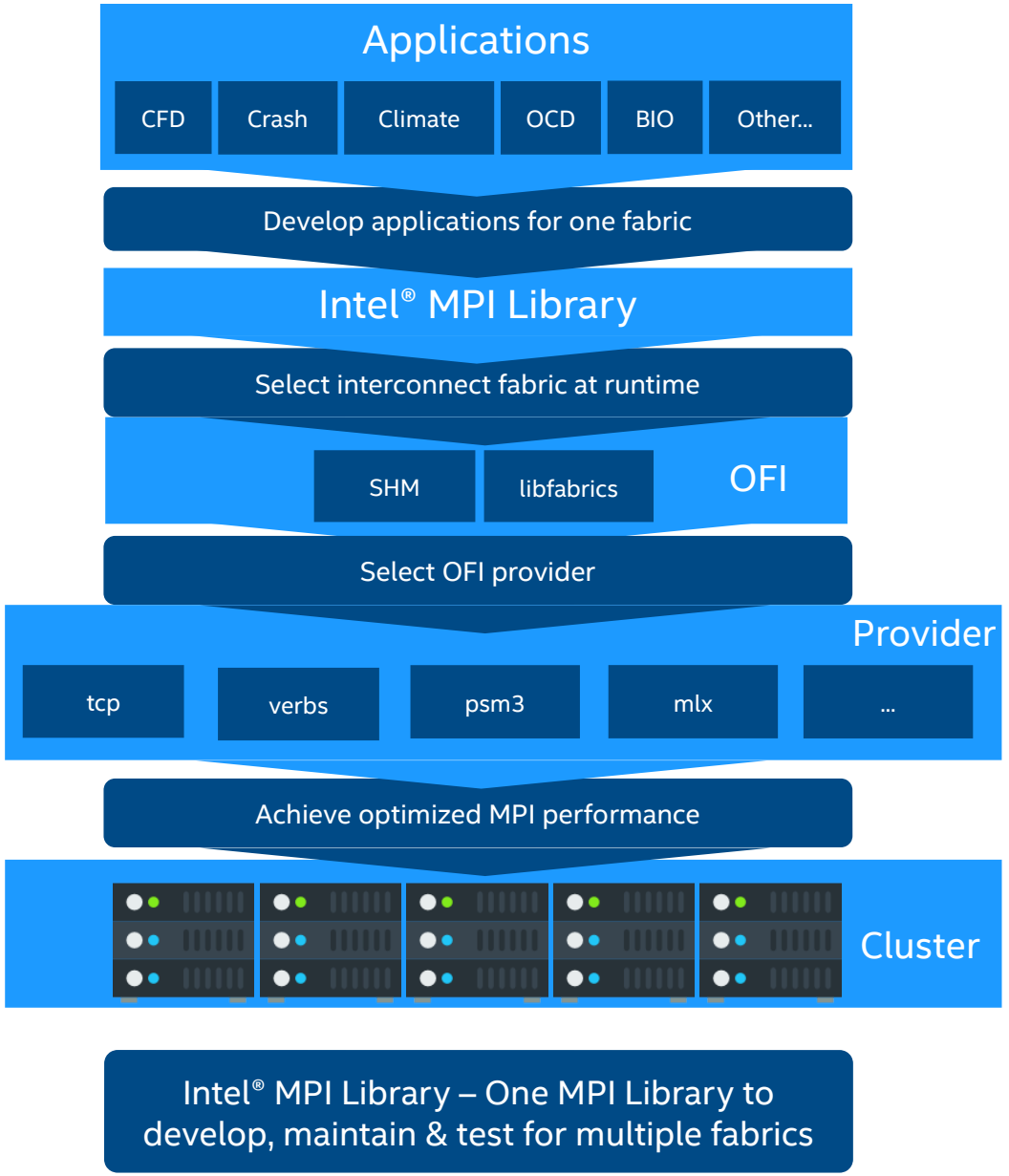
2021.9 Release notes (latest):

- Optimizations for GPU collectives with small message sizes
- Optimizations for pinning for Hybrid CPUs with P-cores and E-cores
- MPI 4.0 - big counts (Tech Preview, C interface, collectives only)



Selecting Fabrics & Providers

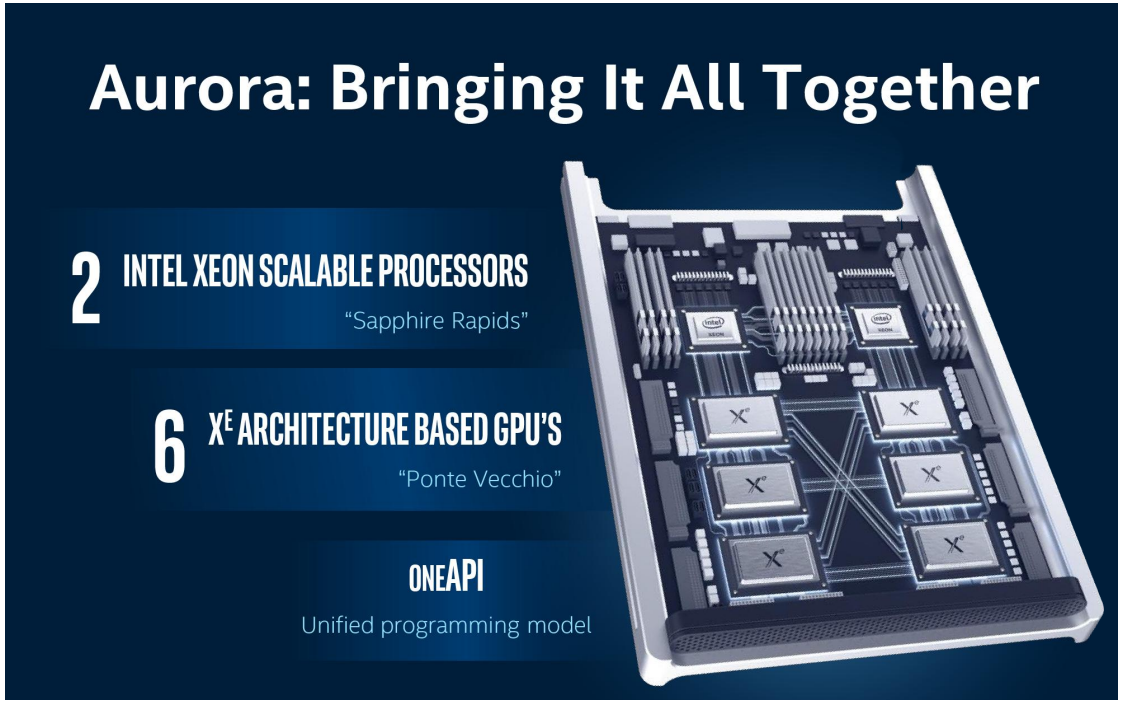
- Runtime selection via environment variables:
- I_MPI_FABRICS
  - shm: optimized for shared-memory; can only be used if all ranks are intranode
  - ofi: general-purpose fabric; requires a provider
- FI_PROVIDER
  - mlx: provider running over UCX
  - tcp: general purpose, over ethernet (e.g. for cloud applications)
  - psm3: Intel's newest provider, strongly recommended for GPU features



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-guide-linux/2021-6/ofi-providers-support.html

Intel® MPI Simplifies Programming for GPUs*

- simple distribution with one MPI rank per tile/GPU
- uncomplicated pinning via environment variables
- transparent communication with GPU buffers – no host-copy necessary!



Intel® MPI – CPU & GPU Pinning

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-pinning.html

Intel® MPI - CPU Pinning Simulator

- Web-based interface
- Platform configuration options
  - load output from `cpuinfo` (IMPI utility)
  - or manually define configuration
- Provides IMPI environment variable settings for desired pinning

See more: https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html

Intel® GPU pinning support

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

Default settings:

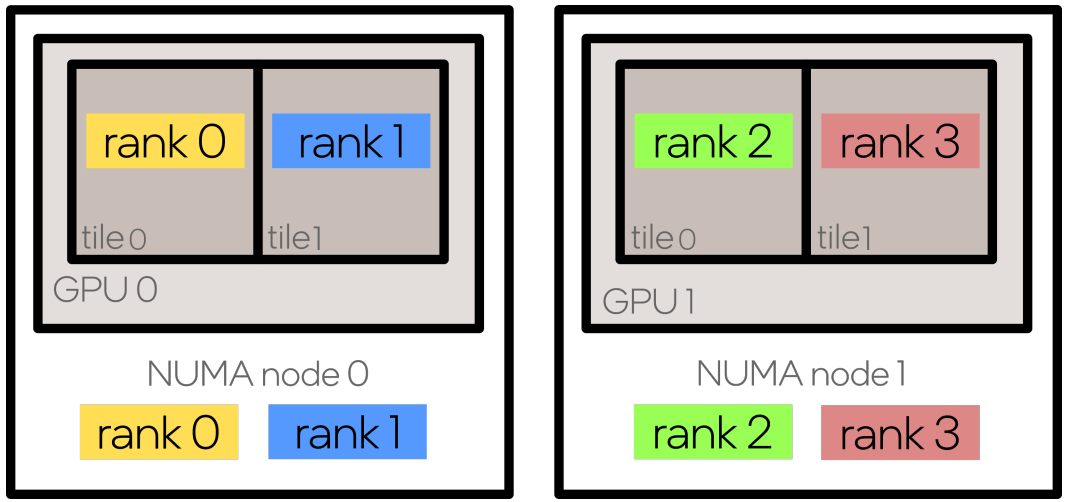
I_MPI_OFFLOAD_* family:

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

E.g: I_MPI_OFFLOAD_TOPOLIB=level_zero

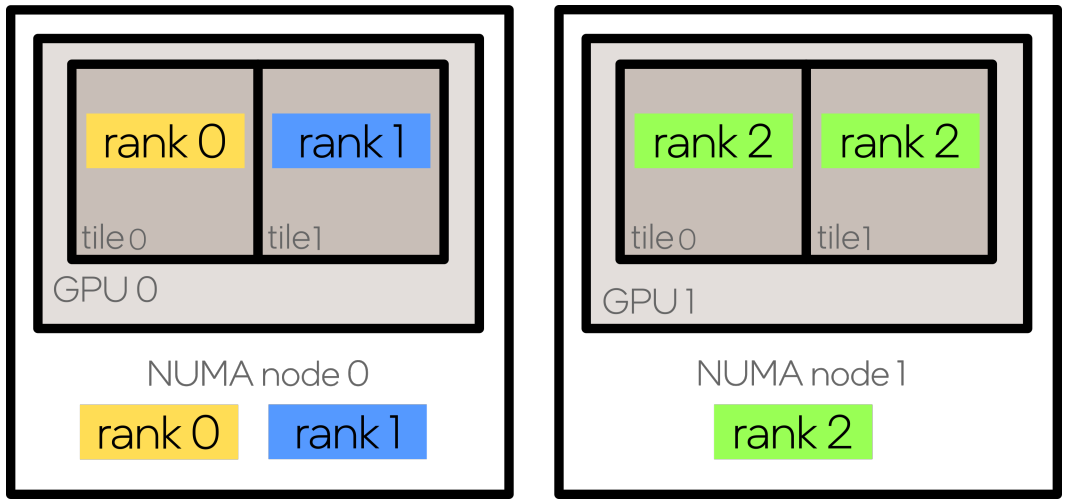
Example – Default: 4 ranks

```
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
```



Example – Default: 3 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
```



Example – I_MPI_OFFLOAD_DOMAIN_SIZE

I_MPI_OFFLOAD_DOMAIN_SIZE=1

I_MPI_DEBUG=3

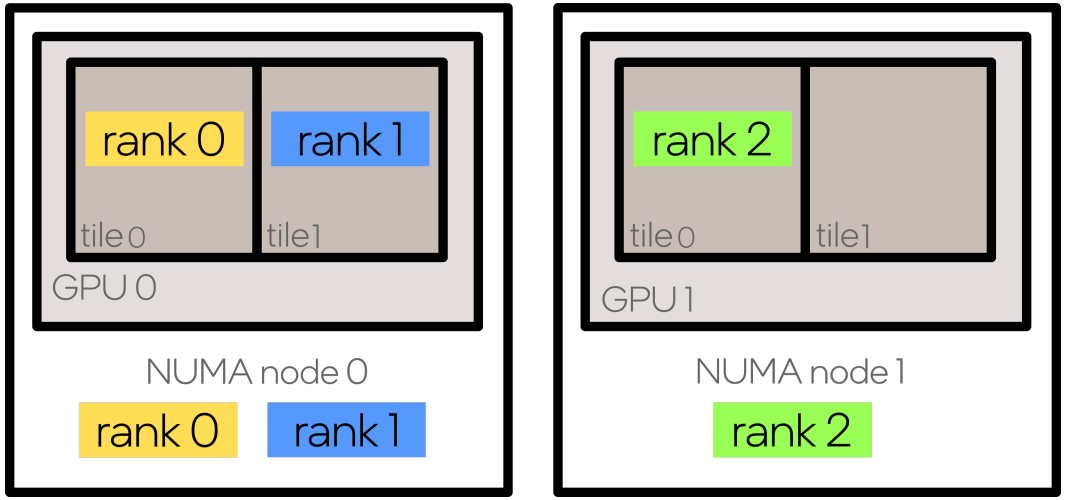
[0] MPI startup(): ===== GPU pinning on host1 =====

[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_CELL

I_MPI_OFFLOAD_CELL=device

I_MPI_DEBUG=3

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

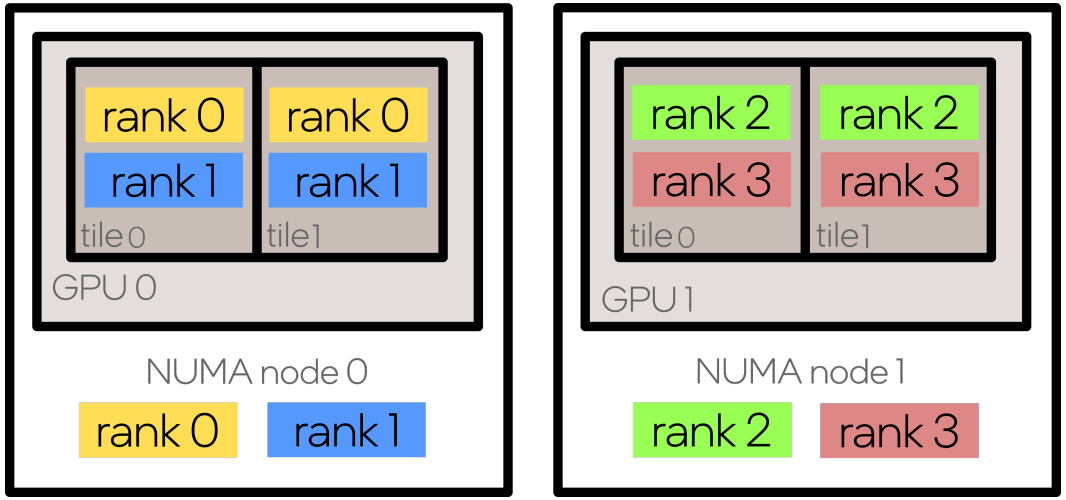
[0] MPI startup(): Rank Pin tile

[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_DEVICES

I_MPI_OFFLOAD_DEVICES=0

I_MPI_DEBUG=3

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

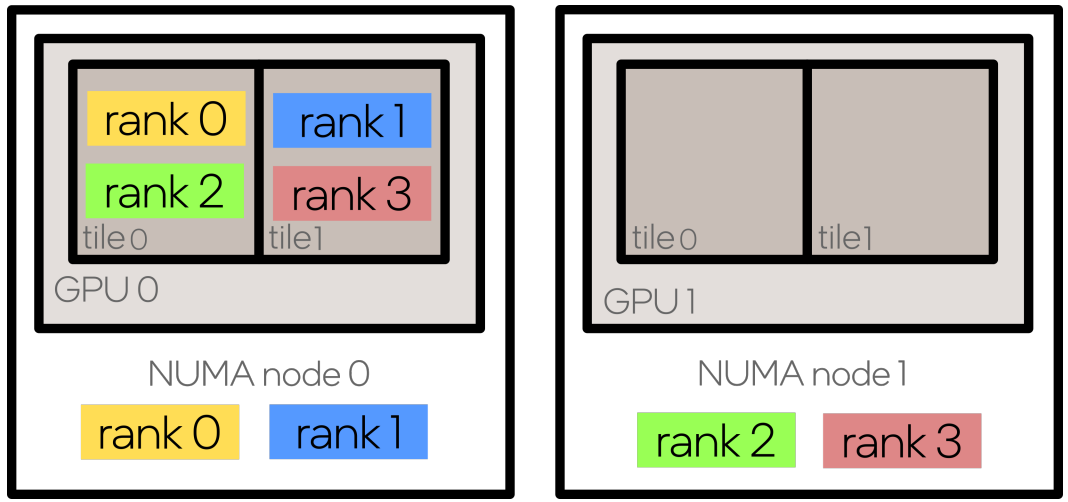
[0] MPI startup(): Rank Pin tile

[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 =====

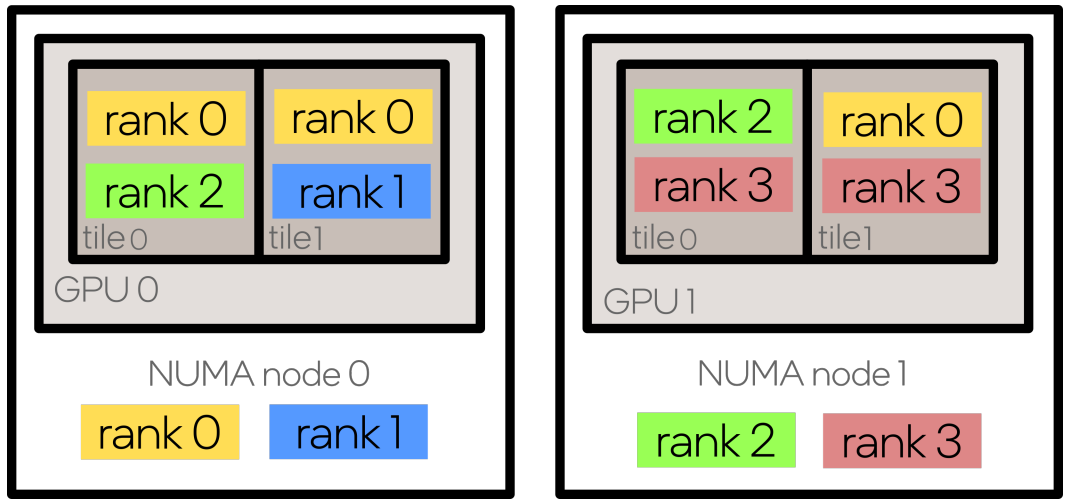
[0] MPI startup(): Rank Pin tile

[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

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

Execution models – OpenMP Offloading

1 - Old copy-back technique

```
#pragma omp target data map(to: ..., b[0:len]) map(tofrom: c[0:len])
{
  double *a = (double*) omp_target_alloc(len * sizeof(double), 0);

  for (int j=0; j < nrep; j++) {
    // Compute on GPU
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < len; i++) {
      a[i] = sin(b[i] + alpha * c[i]);
    }

}
// Copy data back to host and send from there
 MPI_Sendrecv(c, len, MPI_DOUBLE, destRank,   1,
              b, len, MPI_DOUBLE, sourceRank, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

Execution models – OpenMP Offloading

2 – GPU-buffer aware

```
#pragma omp target data map(to: ..., b[0:len]) map(tofrom: c[0:len])
                use_device_ptr(b,c)
{
  double *a = (double*) omp_target_alloc(len * sizeof(double), 0);

  for (int j=0; j < nrep; j++) {
    // Compute on GPU
    #pragma omp target teams distribute parallel for is_device_ptr(b,c)
    for (int i = 0; i < len; i++) {
      a[i] = sin(b[i] + alpha * c[i]);1
    }

    // GPU-aware MPI sendrecv
    MPI_Sendrecv(b, len, MPI_DOUBLE, destRank,   1,
                 c, len, MPI_DOUBLE, sourceRank, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

For SYCL with USM: automatically GPU-aware!

Execution models – OpenMP Offloading

3 – Compiling & Running

- C language:

```
$ mpiicc -cc=icx -fiopenmp -fopenmp-targets=spir64 test.c -o test
```

- Fortran language:

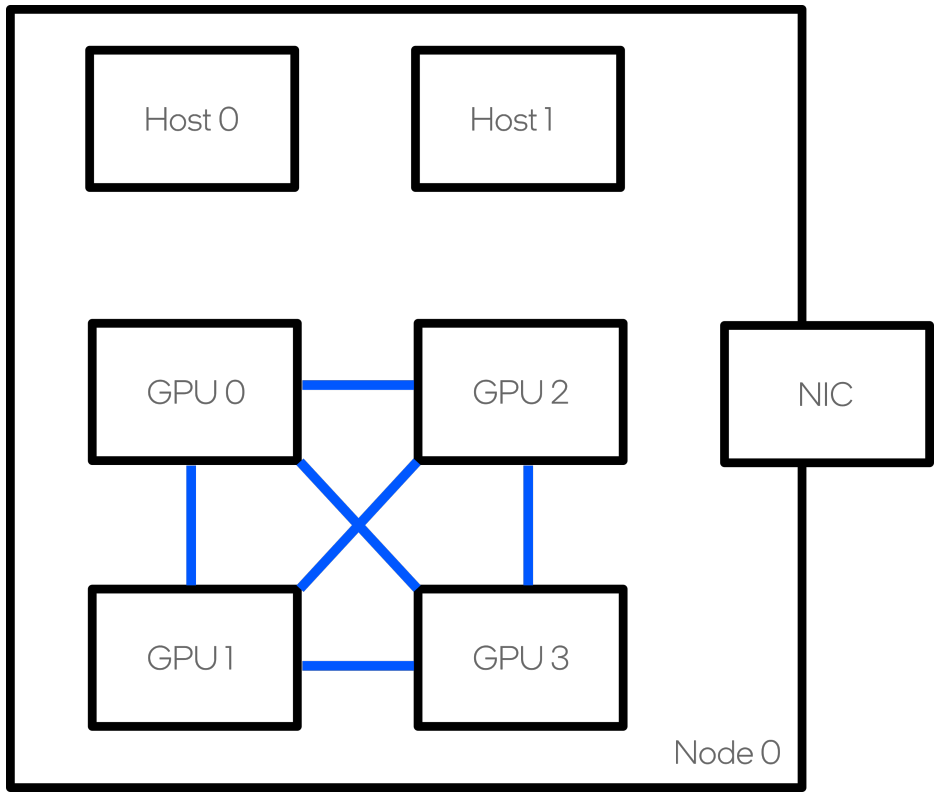
```
$ mpiifort -fc=ifx -fiopenmp -fopenmp-targets=spir64 test.f90 -o test
```

- Launch application as usual:

```
$ I_MPI_OFFLOAD=1 mpirun –n 8 test
```

GPU-Buffer Control Variables

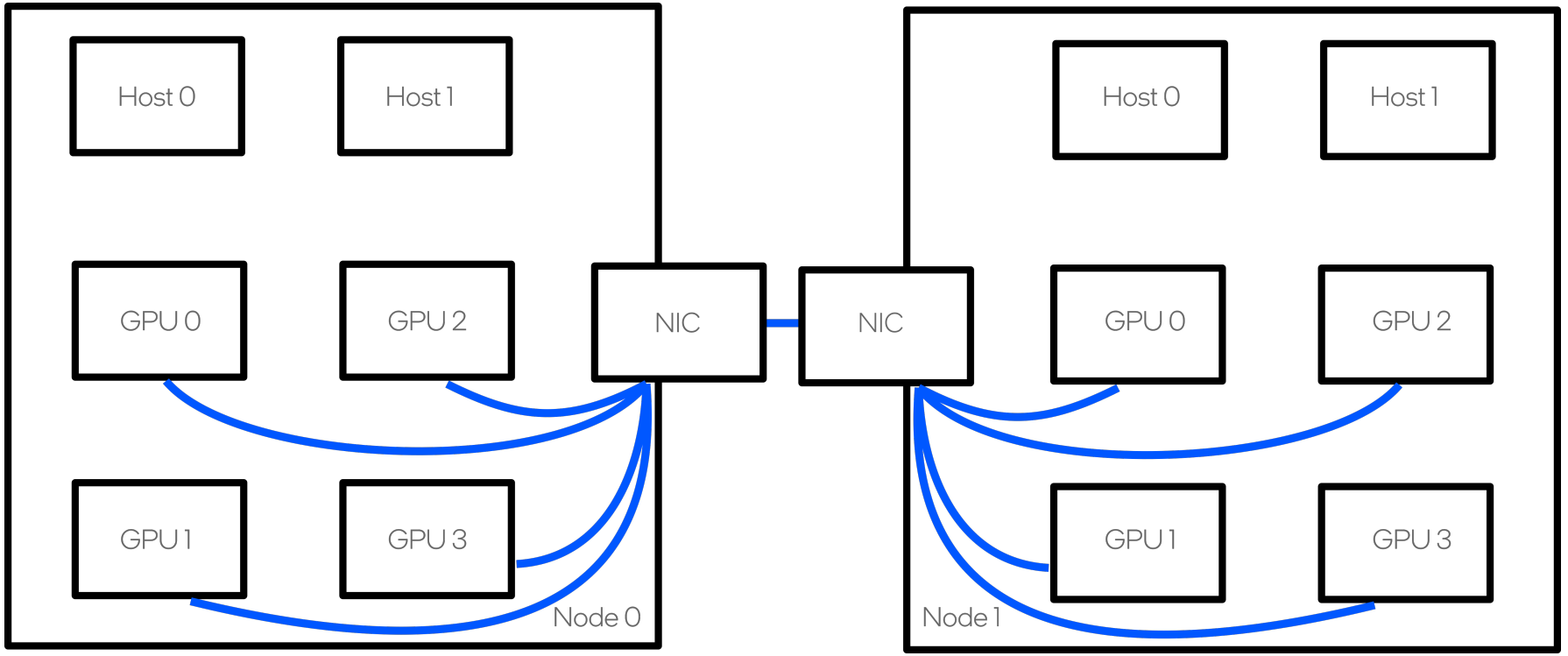
- I_MPI_OFFLOAD_IPC: toggles GPU IPC (i.e. XeLink; single-node)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

GPU-Buffer Control Variables

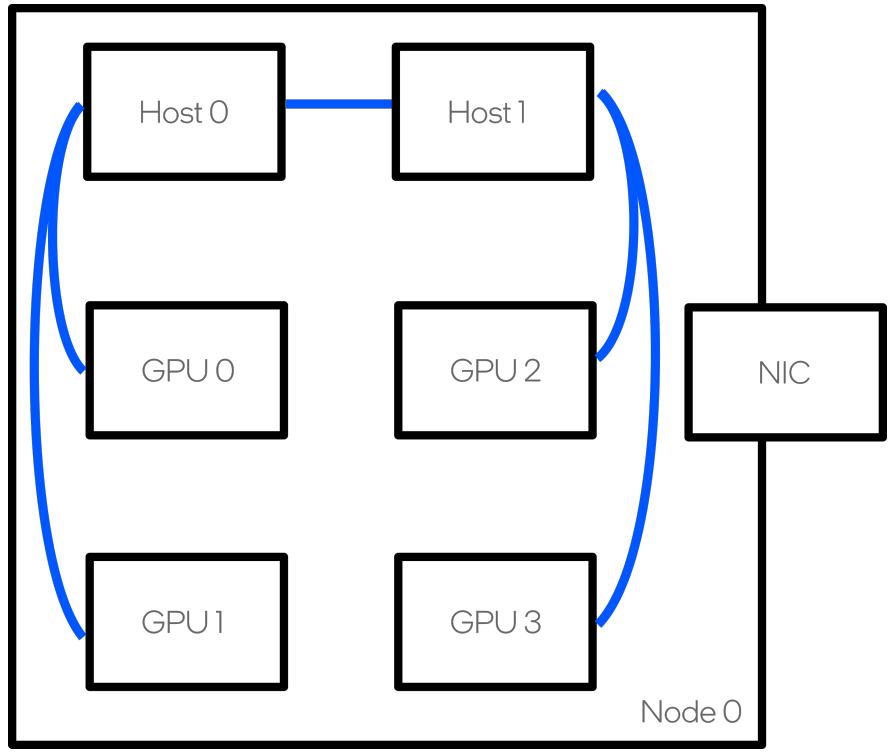
- I_MPI_OFFLOAD_RDMA: toggles D2D copy through NIC (multi-node)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

GPU-Buffer Control Variables

- I_MPI_OFFLOAD_PIPELINE: toggles use of pipeline algorithms (for large message sizes)
- I_MPI_OFFLOAD_PIPELINE_THRESHOLD: uses pipeline for messages larger than this threshold (default is 65536)



See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/gpu-buffers-support.html

Tech Preview - IMB and IPC

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html

Notices & Disclaimers

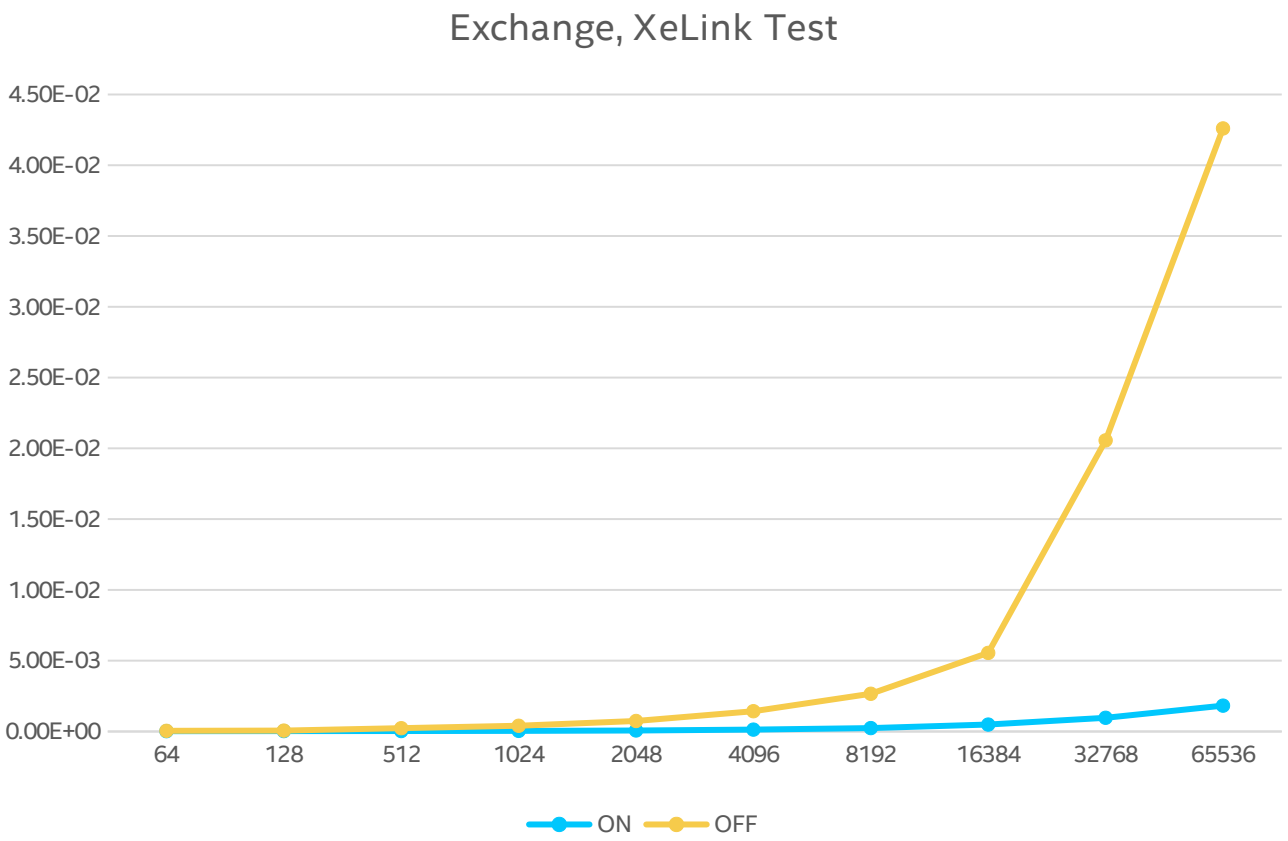
Performance varies by use, configuration and other factors. Learn more on the Performance Index site.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

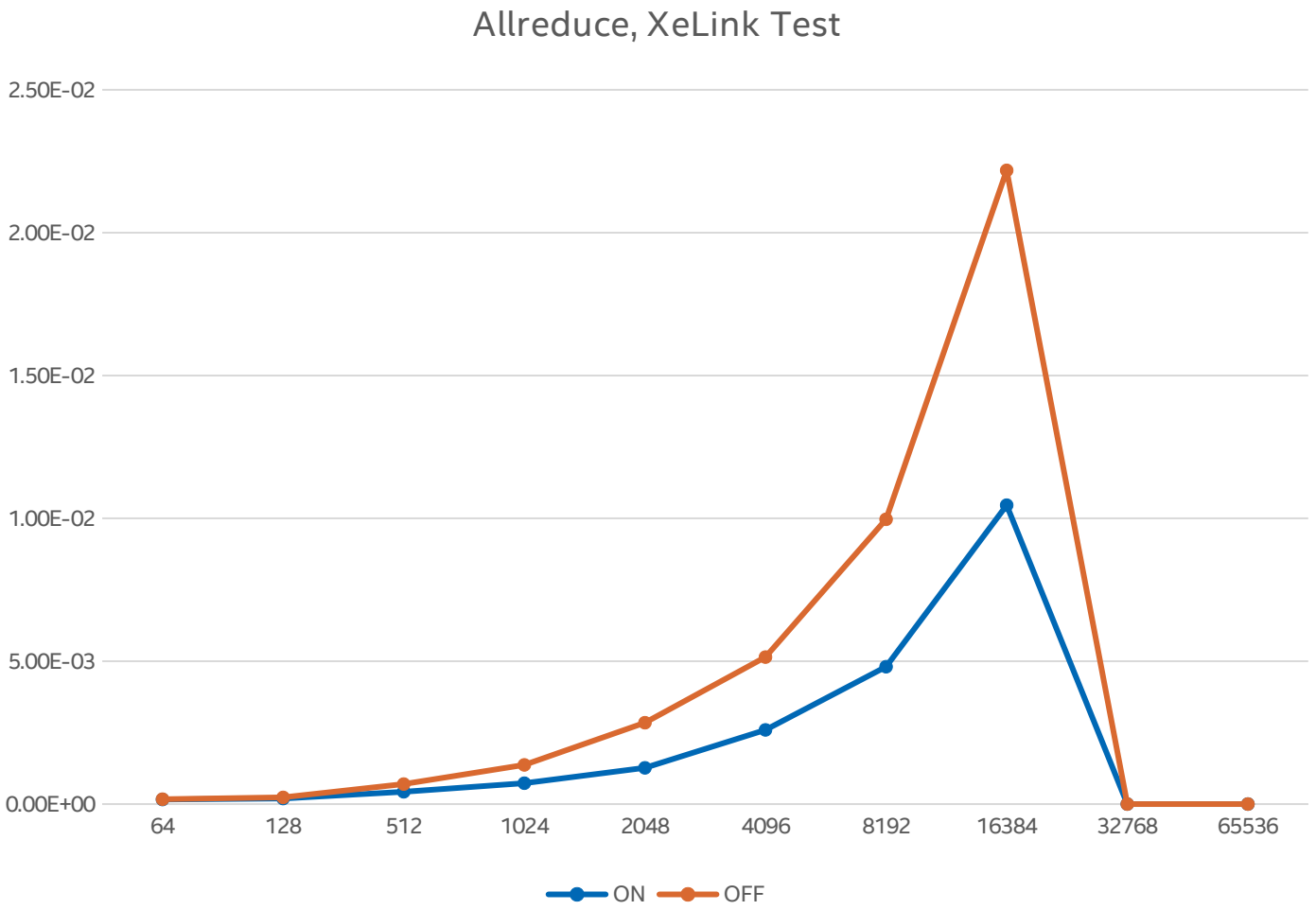
IMB Benchmark
MPI_Sendrecv Comm. Time Comparison



- 8 MPI Ranks (1 per tile)
- Y-axis: lower is better
- X-axis: message size in KiB
- Double-precision

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html

IMB Benchmark
MPI_Allreduce Comm. Time Comparison



- 8 MPI Ranks (1 per tile)
- Y-axis: lower is better
- X-axis: message size in KiB
- Double-precision

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html

I_MPI_ADJUST_* Family

- Environment variables for selecting the algorithm
  - No recompilation!
- **Performance** depends on
  - Hardware
  - Message Size
  - Number of MPI ranks
  - Topology
- Intel® MPI provides a default setting which should be **performant for most cases**

I_MPI_ADJUST_ALLREDUCE	MPI_Allreduce	1. Recursive doubling 2. Rabenseifner's 3. Reduce + Bcast 4. Topology aware Reduce + Bcast 5. Binomial gather + scatter 6. Topology aware binominal gather + scatter 7. Shumilin's ring 8. Ring 9. Knomial 10. Topology aware SHM-based flat 11. Topology aware SHM-based Knomial 12. Topology aware SHM-based Knary
I_MPI_ADJUST_ALLTOALL	MPI_Alltoall	1. Bruck's 2. Isend/Irecv + waitall 3. Pair wise exchange 4. Plum's
I_MPI_ADJUST_BARRIER	MPI_Barrier	1. Dissemination 2. Recursive doubling 3. Topology aware dissemination 4. Topology aware recursive doubling 5. Binominal gather + scatter 6. Topology aware binominal gather + scatter 7. Topology aware SHM-based flat 8. Topology aware SHM-based Knomial 9. Topology aware SHM-based Knary

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/i-mpi-adjust-family-environment-variables.html

I_MPI_ADJUST_* Family

Custom tuning may be profitable for:

- untested number of ranks configurations
- non-standard message sizes (e.g. 512 KB < msg_size < 1024 KB)
- new network topologies
- untested interconnects
- applications with high imbalance
- non-standard/user defined datatypes
- uncommon collectives (e.g. reduce_scatter)

I_MPI_ADJUST_ALLREDUCE	MPI_Allreduce	1. Recursive doubling 2. Rabenseifner's 3. Reduce + Bcast 4. Topology aware Reduce + Bcast 5. Binomial gather + scatter 6. Topology aware binominal gather + scatter 7. Shumilin's ring 8. Ring 9. Knomial 10. Topology aware SHM-based flat 11. Topology aware SHM-based Knomial 12. Topology aware SHM-based Knary
I_MPI_ADJUST_ALLTOALL	MPI_Alltoall	1. Bruck's 2. Isend/Irecv + waitall 3. Pair wise exchange 4. Plum's
I_MPI_ADJUST_BARRIER	MPI_Barrier	1. Dissemination 2. Recursive doubling 3. Topology aware dissemination 4. Topology aware recursive doubling 5. Binominal gather + scatter 6. Topology aware binominal gather + scatter 7. Topology aware SHM-based flat 8. Topology aware SHM-based Knomial 9. Topology aware SHM-based Knary

See more: https://www.intel.com/content/www/us/en/docs/mpi-library/developer-reference-linux/2021-8/i-mpi-adjust-family-environment-variables.html

Intel® MPI Tuner – Simple Usage

1) Enable autotuner and store results (store is optional):

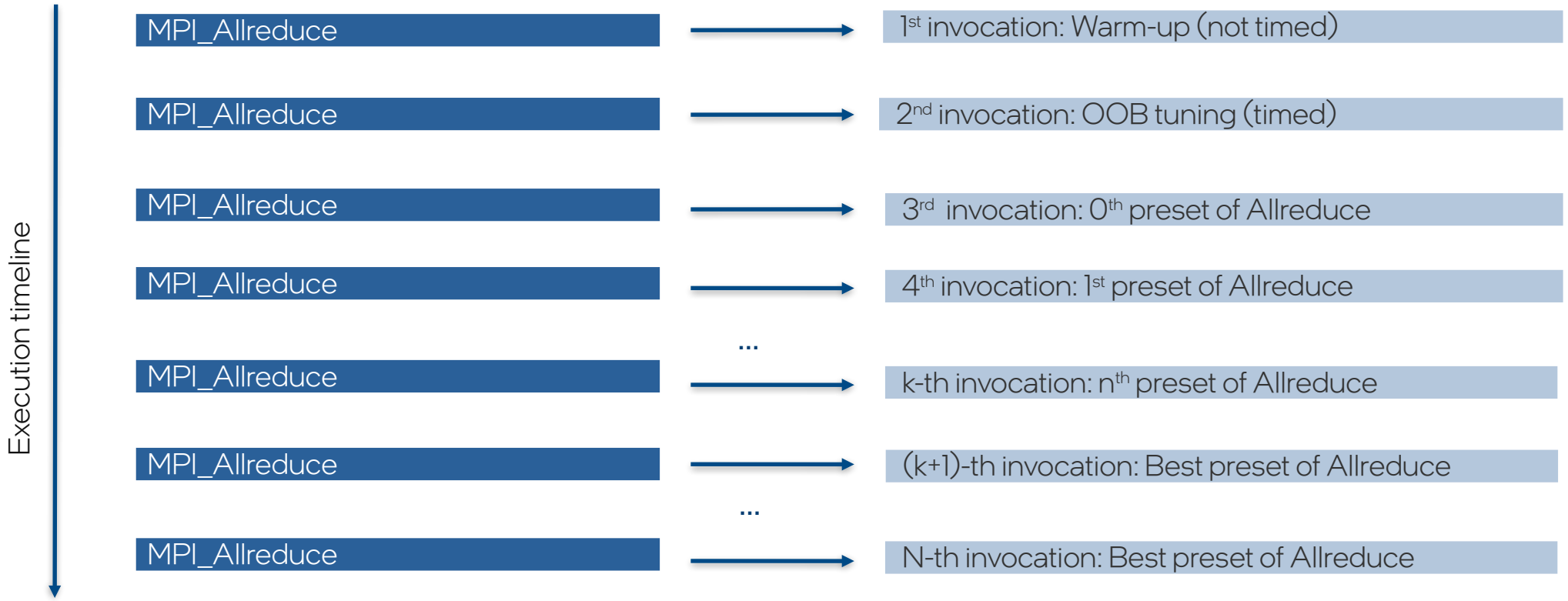
export **I_MPI_TUNING_MODE=auto**
export **I_MPI_TUNING_BIN_DUMP=./tuning_results.dat**
export **I_MPI_TUNNING_AUTO_ITER_NUM=1**
mpirun <mpi_args> <app with args>

(this run may be slower, due to the tunning)

2) Use the results of autotuner for subsequent launches (optional):

unset **I_MPI_TUNING_MODE**
export **I_MPI_TUNING_BIN=./tuning_results.dat**
mpirun <mpi_args> <app with args>

Intel® MPI Tuner – Simple Usage



(performed for each message size/communicator)

Intel® MPI Tuner – More Options

- I_MPI_TUNING_MODE=<auto|auto:application|auto:cluster> (disabled by default)
- I_MPI_TUNING_AUTO_POLICY=<min|max|avg> Which metric to use to select best algorithm (max by default)
- I_MPI_TUNING_AUTO_SYNC=<0|1> Call internal barrier on every tuning iteration (0 by default)
- I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=<num> (1 by default)
- I_MPI_TUNING_AUTO_ITER_NUM=<num> (1 by default)

  Suggestion: min #iter per collective/message size/communicator
  *I_MPI_TUNING_AUTO_WARMUP_ITER_NUM + [(range+1)*I_MPI_TUNING_AUTO_ITER_NUM]*
- I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=<max_mem> Controls message size limit (64Kb default)
- I_MPI_TUNING_AUTO_STORAGE_SIZE=<max_size> Communicator storage size (512 Kb default)
- Merging tuning files:

  export **I_MPI_TUNING_BIN=tuned1.dat,tuned2.dat** *# more files allowed*
  export **I_MPI_TUNING_BIN_DUMP=tuning_merged.dat**
  mpiexec –n 1 ./dummy_mpi_app

Troubleshooting MPI Applications

Using System's GDB

- Interactive debugging using system's gdb:

  $ mpirun -n 4 **-gdb** IMB-MPI1 allreduce

  or

  $ mpirun -n 4 **–gdba** *<MPI_PID>*

- Starts one gdb-server and one gdb-client per rank. User interacts with gdb-server only.

```
[ rafael@icx1 ] $ mpirun -n 4 -gdb ./a.out
mpigdb: attaching to 50265 ./a.out icx1
mpigdb: attaching to 50266 ./a.out icx1
mpigdb: attaching to 50267 ./a.out icx1
mpigdb: attaching to 50268 ./a.out icx1
[0-3] (mpigdb) b test.c:37
[0-3]   Breakpoint 1 at 0x400912: file ./test.c, line 37.
[0-3] (mpigdb) r
[0-3]   Continuing.
[1-3]
[0]
[1]     Breakpoint 1, printHello (rank=1, size=4) at ./test.c:37
[2]     Breakpoint 1, printHello (rank=2, size=4) at ./test.c:37
[3]     Breakpoint 1, printHello (rank=3, size=4) at ./test.c:37
[0]     Breakpoint 1, printHello (rank=0, size=4) at ./test.c:37
[1-3]   37          MPI_Get_processor_name(name, &namelen);
[0]     37          MPI_Get_processor_name(name, &namelen);
[0-3] (mpigdb) r
[0-3]   Continuing.
Hello world: rank 0 of 4 running on icx1
Hello world: rank 1 of 4 running on icx1
Hello world: rank 2 of 4 running on icx1
Hello world: rank 3 of 4 running on icx1
[0]     [Inferior 1 (process 50265) exited normally]
[1]     [Inferior 1 (process 50266) exited normally]
[2]     [Inferior 1 (process 50267) exited normally]
[3]     [Inferior 1 (process 50268) exited normally]
```

Troubleshooting MPI Applications

SLURM's multi-prog

- **srun –multi-prog ./multiprog.conf**
- multiprog.conf example:

```
# <rank-range> <app-with-args>
0-1 ./my_app
2   vtune -c hotspots -- ./my_app
3   vtune -c hpc-performance -- ./my_app
4   vtune -c memory-access -- ./my_app
5   ./my_app
```

Troubleshooting MPI Applications

Checking correctness

```
[ rafael@icx1 ] $ mpirun -n 4 -check_mpi ./a.out

[0] INFO: CHECK LOCAL:EXIT:SIGNAL ON
[0] INFO: CHECK LOCAL:EXIT:BEFORE_MPI_FINALIZE ON
[0] INFO: CHECK LOCAL:MPI:CALL_FAILED ON
[0] INFO: CHECK LOCAL:MEMORY:OVERLAP ON
(...)
[0] INFO: CHECK GLOBAL:COLLECTIVE:INVALID_PARAMETER ON
[0] INFO: CHECK GLOBAL:COLLECTIVE:COMM_FREE_MISMATCH ON
[0] INFO: maximum number of errors before aborting: CHECK-MAX-ERRORS 1
[0] INFO: maximum number of reports before aborting: CHECK-MAX-REPORTS 0 (= unli
[0] INFO: maximum number of times each error is reported: CHECK-SUPPRESSION-LIMI
[0] INFO: timeout for deadlock detection: DEADLOCK-TIMEOUT 60s
[0] INFO: timeout for deadlock warning: DEADLOCK-WARNING 300s
[0] INFO: maximum number of reported pending messages: CHECK-MAX-PENDING 20

Hello world: rank 0 of 4 running on icx1

[1] ERROR: LOCAL:MPI:CALL_FAILED: error
[1] ERROR:    Invalid rank has value 100 but must be nonnegative and less than 4
[1] ERROR:    Error occurred at:
[1] ERROR:       MPI_Send(*buf=0x7ffdcc877c38, count=1, datatype=MPI_INT, dest=1
[1] ERROR:       printHello (/home/rafael/area51/support/ICC_PLAYGROUND/./test.c
[1] ERROR:       main (/home/rafael/area51/support/ICC_PLAYGROUND/./test.c:26)
[1] ERROR:       __libc_start_main (/lib64/libc-2.31.so)
[1] ERROR:       _start (/home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x
[1] INFO: 1 error, limit CHECK-MAX-ERRORS reached => aborting
```

(Attention: the output can be quite verbose!)

- Intel Trace Analyser and Collector Correctness Check:

  mpirun -n 4 **-check_mpi** <app>
  or
  export LD_PRELOAD=${VT_SLIB_DIR}/**libVTmc.so**:${I_MPI_ROOT}/lib/release/libmpi.so
  srun <app>