Intel® Developer Workshop
Day 2
Intel MPI

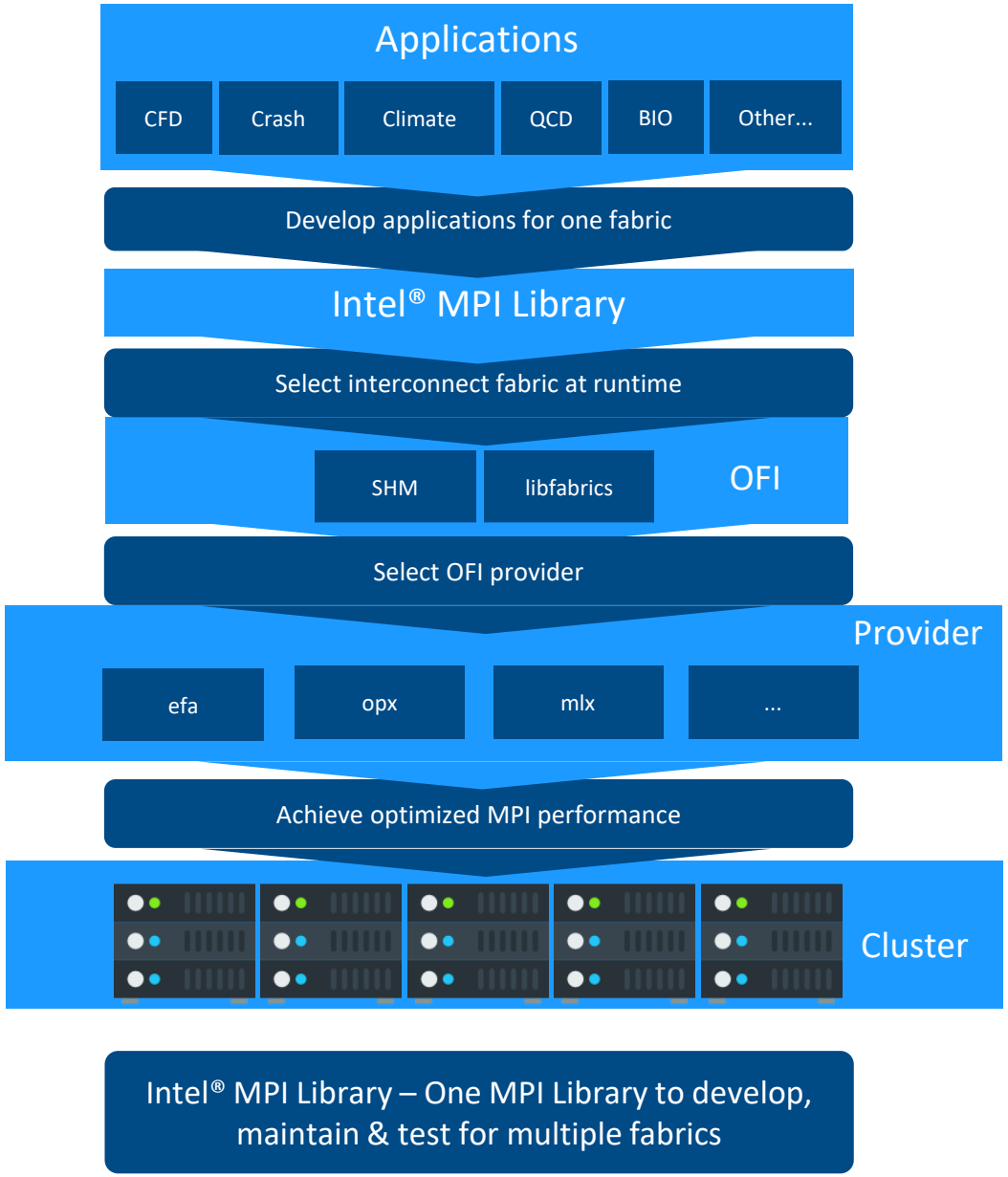
Tobias Kloeffel

20th -21st April 2026

Intel® MPI Library

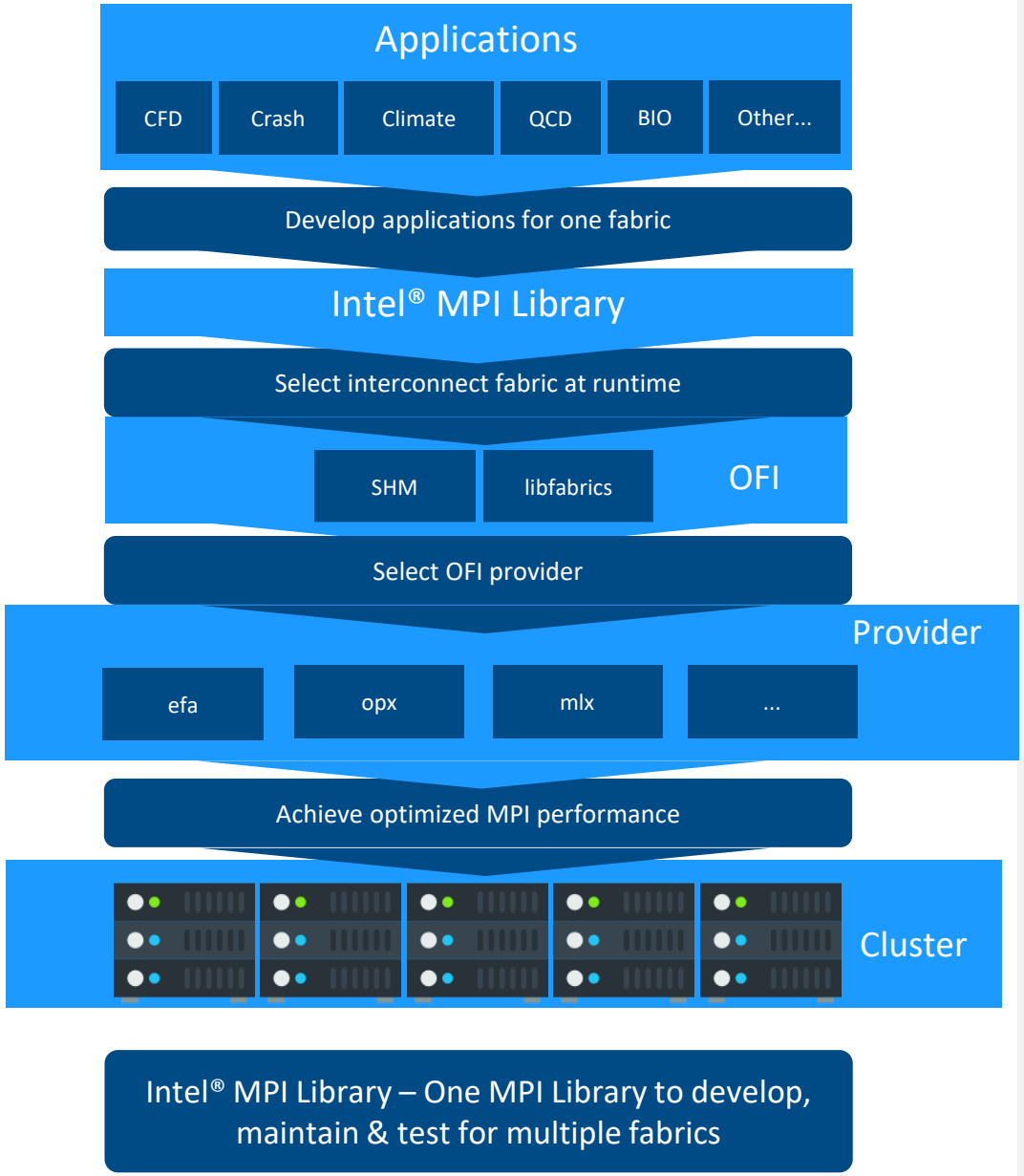
Key Features:

- MPI-1, MPI-2.2, MPI-3.1, MPI-4.0, and MPI-5.0
- Interconnect independence
- C, C++ support, Fortran language bindings
- Microsoft* Azure, Amazon* AWS/EFA, Google* GCP
- **Intel GPU** pinning, NVidia and **Intel GPU** & buffers support (GPU-aware MPI)
- PMIx support



Open Fabrics Interface Provider

- **I_MPI_FABRICS**
  - **shm**: optimized for shared-memory; can only be used if all ranks are intranode
  - **ofi**: general-purpose fabric; requires a provider
  - **shm:ofi** default and recommended
- **I_MPI_OFI_PROVIDER**
  - **mlx:** provider running over UCX
  - **efa:** elastic fabric adapter *(e.g. AWS cloud)*
  - **opx:** Cornelis Networks Omni-Path



Interconnects and Provider

Hardware Interconnect	I_MPI_OFI_PROVIDER	Details
Nvidia InfiniBand* hardware	mlx	Improve Performance and Stability with Intel MPI Library on InfiniBand.
TCP Sockets	tcp	Basic fabric present in standard computers. Is a fallback in e.g. Cloud setups (e.g. GCP, Azure*, AWS*). Can be used with IPoIB.
Omni-Path Fabric	opx	Cornelis Networks

https://www.intel.com/content/www/us/en/docs/mpi-library/developer-guide-linux/2021-16/ofi-providers-support.html

Intel® MPI release notes 2021.17

- Heterogenous core support: P-Core/E-Core detection for CPU pinning.
- Integrated changes from libfabric v2.2.0
- Linux/C only: MPI 5.0 standard support
- Tech preview: Implicit mode for Thread Split functionality
- Added Fortran modules for gcc-15.1.0. Removed legacy Fortran modules for older GCC versions (4.8-6.1) to streamline compatibility.
- Allreduce CUDA Kernel for efficient GPU data offload
- Bugs fixes
  - Including a fix for issue in libfabric's mlx provider that caused hangs under high network utilization conditions

Intel® MPI release notes 2021.18

- Expanded MPI 5.0 standard technical preview with ABI compatibility for Fortran
- Introduced technical preview of Intel® Arc™ Pro B-Series GPU support (single node)
- Added full support for Intel® Xeon 6+ processors and optimized single node performance
- Added technical preview capability for non-mpiexec based job launching to support AI frameworks and applications
- Enhanced multithreaded communication scalability with expanded thread split mode support
- Reduced job startup latency for single node workloads by defaulting to SHM fabric
- Added new tunable Alltoall algorithms accessible via I_MPI_ADJUST_ALLTOALL (values 14–18)
- Addressed a CPU affinity issue by updating the default I_MPI_HYDRA_BRANCH_COUNT value for slurm
- Integrated libfabric v2.4.0 for more reliable communication
- Updated IMB to version 2021.11, with multiple stability and compatibility fixes
- Clarified versioning with explicit patch numbers (e.g., 2021.18.0)

Heterogenous Core Support

- Automatic detection of performance cores (p-cores) and efficiency cores (e-cores) on newer client CPUs
- Gives users flexibility to choose to run MPI applications on specific core type
- CPU pinning must be enabled by setting I_MPI_PIN=1
- Controlled using I_MPI_PIN_PROCESSOR_EXCLUDE_LIST environment variable, where one of the following settings can be used:

  - “ecore” or “ecores” – Do not use e-cores for the MPI application
  - “pcore” or “pcores” – Do not use p-cores for the MPI application

Heterogenous Core Support Example

P-Cores Only

```
export I_MPI_DEBUG=6
export I_MPI_PIN=1
export I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=ecores

===== CPU pinning =====
Rank    Pid      Node name    Pin cpu
0       6886     <node name>  {0-1}
1       6887     <node name>  {2-3}
2       6888     <node name>  {4-5}
3       6889     <node name>  {6-7}
4       6890     <node name>  {8-9}
5       6891     <node name>  {10-11}
6       6892     <node name>  {12-13}
7       6893     <node name>  {14-15}
(…)
```

E-Cores Only

```
export I_MPI_DEBUG=6
export I_MPI_PIN=1
export I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=pcores

===== CPU pinning =====
Rank    Pid      Node name    Pin cpu
0       6907     <node name>  {16-17}
1       6908     <node name>  {18-19}
2       6909     <node name>  {20-21}
3       6910     <node name>  {22-23}
4       6911     <node name>  {24-25}
5       6912     <node name>  {26-27}
6       6913     <node name>  {28-29}
7       6914     <node name>  {30-31}
(…)
```

Fortran Modules Cleanup

- Added Fortran modules for gcc-15.1.0.
- Removed legacy Fortran modules for older GCC versions 4.8 – 6.1
- Use Intel MPI Bindings Kit to build the bindings for older GCC versions:

```
1. mkdir intel-mpi-binding-kit
2. cd intel-mpi-binding-kit
3. tar xf ${I_MPI_ROOT}/opt/mpi/binding/intel-mpi-binding-kit.tar.gz
4. cd f90
5. make MPI_INST=/opt/intel/oneapi/mpi/2021.17/ F90=gfortran
```

- The bindings will be generated in include/ and lib/ directories

SLURM integration

- Intel MPI’s mpirun/mpiexec uses srun as hydra bootstrap
  - Since Slurm 23.11.3 Slurm injects I_MPI_HYDRA_BOOTSTRAP_EXEC_EXTRA_ARGS=--external-launcher which basically disables Slurm pinning.
  - Intel MPI still respects SLURM_CPUS_PER_TASK and SLURM_TASKS_PER_NODE but not any hint flag
  - To make use of Slurm pinning, srun instead of mpirun/mpiexec is required. Using srun requires to set up I_MPI_PMI_LIBRARY=PATH_TO_PMI

SLURM integration 2021.18

- Intel MPI 2021.18 changes:
  - I_MPI_HYDRA_BRANCH_COUNT=0 if not set by the user explicitly
  - SLURM_OVERLAP=1 will not be set anymore => enabling job farming using mpirun

    Srun needs to create a cgroup for all ranks created on a specific node, Intel MPI will pin individual ranks to cores in that cgroup. Still more complicated than just using srun.

Implicit Mode for Thread Split Functionality

- This new mode enables threaded use of MPI collectives like MPI_Allreduce
- It is enabled by setting the environment variables:
  I_MPI_THREAD_SPLIT=1
  I_MPI_THREAD_SPLIT_MODE=implicit
- Number of threads in use by the application for MPI must be set:
  I_MPI_THREAD_MAX=N

Thread-split – implicit mode code modifications

```
1. #define N 2
2.
3. int main() {
4.     int i;
5.     int buffer[N];
6.
7.
8.     MPI_Init(NULL, NULL);
9.
10.
11.
12. #pragma omp parallel for num_threads(N)
13.     for (i = 0; i < N; i++) {
14.         // threaded partial computation
15.         // i-th thread contributes to buffer[i]
16.
17.
18.
19.     }
20.
21.     // single-threaded global communication
22.     MPI_Allreduce(buffer, buffer, N, MPI_INT,
23.         MPI_SUM, MPI_COMM_WORLD);
24.
25.     MPI_Finalize();
26.     return 0;
27.}
```

```
1. #define N 2
2.
3. int main() {
4.     int i, provided;
5.     int buffer[N];
6.
7.     MPI_Comm comms[N];
8.     MPI_Init_thread(NULL, NULL, MPI_THREAD_MULTIPLE, &provided);
9.     for (i = 0; i < N; i++)
10.         MPI_Comm_dup(MPI_COMM_WORLD, &comms[i]);
11.
12.#pragma omp parallel for num_threads(N)
13.     for (i = 0; i < N; i++) {
14.         // threaded partial computation
15.         // i-th thread contributes to buffer[i]
16.
19.
20.
21.         // threaded partial communication
22.         MPI_Allreduce(&buffer[i], &buffer[i], 1, MPI_INT,
23.             MPI_SUM, comms[i]);
24.     }
25.     MPI_Finalize();
26.     return 0;
27.}
```

Allreduce CUDA Kernel for Efficient GPU Data Offload

- This is an experimental feature to improve **MPI_Allreduce** operations in combination with CUDA GPU usage.

- It will be triggered by a variable:

  I_MPI_ADJUST_ALLREDUCE_OFFLOAD=2

Summary

- Heterogenous core support: P-Core/E-Core detection for CPU pinning.
- Integrated changes from libfabric v2.4.0
- MPI 5.0 standard support (C only)
- Fortran modules cleanup
- Implicit mode for Thread Split functionality
- Allreduce CUDA Kernel for efficient GPU data offload
- Enhanced Slurm integration
- Bug fixes

Conditional Numerical Reproducibility with Intel MPI

- I_MPI_CBWR=<*arg*>
- NOTE:
  Setting the I_MPI_CBWR in a library-wide mode using the environment variable leads to performance penalty.

<*arg*>	CBWR compatibility mode	Description
0	None	Do not use CBWR in a library-wide mode. CNR-safe communicators may be created with `MPI_Comm_dup_with_info` explicitly. This is the default value.
1	Weak mode	Disable topology aware collectives. The result of a collective operation does not depend on the rank placement. The mode guarantees results reproducibility across different runs on the same cluster (independent of the rank placement).
2	Strict mode	Disable topology aware collectives, ignore CPU architecture, and interconnect during algorithm selection. The mode guarantees results reproducibility across different runs on different clusters (independent of the rank placement, CPU architecture, and interconnection)

Conditional Numerical Reproducibility with Intel MPI

- CNR-safe communicators created using MPI_Comm_dup_with_info always work in the strict mode. For example:

```
MPI_Info hint;
MPI_Comm cbwr_safe_world, cbwr_safe_copy;
MPI_Info_create(&hint);
MPI_Info_set(hint, "I_MPI_CBWR", "yes");
MPI_Comm_dup_with_info(MPI_COMM_WORLD, hint, & cbwr_safe_world);
MPI_Comm_dup(cbwr_safe_world, & cbwr_safe_copy);
```

- Note that MPI_COMM_WORLD itself may be used for performance-critical operations without reproducibility limitations.

Tuning Intel MPI

- Intel MPI Library's out of box (OOB) tuning is designed to be widely applicable to several applications, workloads and topologies. However, further tuning is still profitable for,
  - untested number of total ranks and ranks per node combination
  - non-standard message sizes (e.g. 512 KB < msg_size < 1024 KB)
  - new network topologies
  - untested interconnects (e.g. Cray)
  - applications with high imbalance
  - non-standard/user defined datatypes
  - uncommon collectives (e.g. reduce_scatter)

- Achieving ***even small*** performance gains without code changes/rebuilding for the most time-consuming applications on a cluster over its service life represent significant savings.

Autotuner - search space

```
u31705@s001-n014:~$ impi_info -v I_MPI_ADJUST_ALLREDUCE
I_MPI_ADJUST_ALLREDUCE
  MPI Datatype:
    MPI_CHAR
  Description:
    Control selection of MPI_Allreduce algorithm presets.
    Arguments
    <algid> - Algorithm identifier
    -----------------------------------------------------
    > 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
    -----------------------------------------------------
```

range: 0-24

```
u31705@s001-n014:~$ impi_info -v I_MPI_ADJUST_BCAST
I_MPI_ADJUST_BCAST
  MPI Datatype:
    MPI_CHAR
  Description:
    Control selection of MPI_Bcast algorithm presets.
    Arguments
    <algid> - Algorithm identifier
    -----------------------------------------------
    > 1 - Binomial
    > 2 - Recursive doubling
    > 3 - Ring
    > 4 - Topology aware binomial
    > 5 - Topology aware recursive doubling
    > 6 - Topology aware ring
    > 7 - Shumilin's
    > 8 - Knomial
    > 9 - Topology aware SHM-based flat
    > 10 - Topology aware SHM-based Knomial
    > 11 - Topology aware SHM-based Knary
    -----------------------------------------------
```

range: 0-17

Environment variables – Main flow control

I_MPI_TUNING_MODE=<auto|auto:application|auto:cluster> (**disabled** by default)

I_MPI_TUNING_AUTO_ITER_NUM=<number> Tuning iterations number (**1** by default).

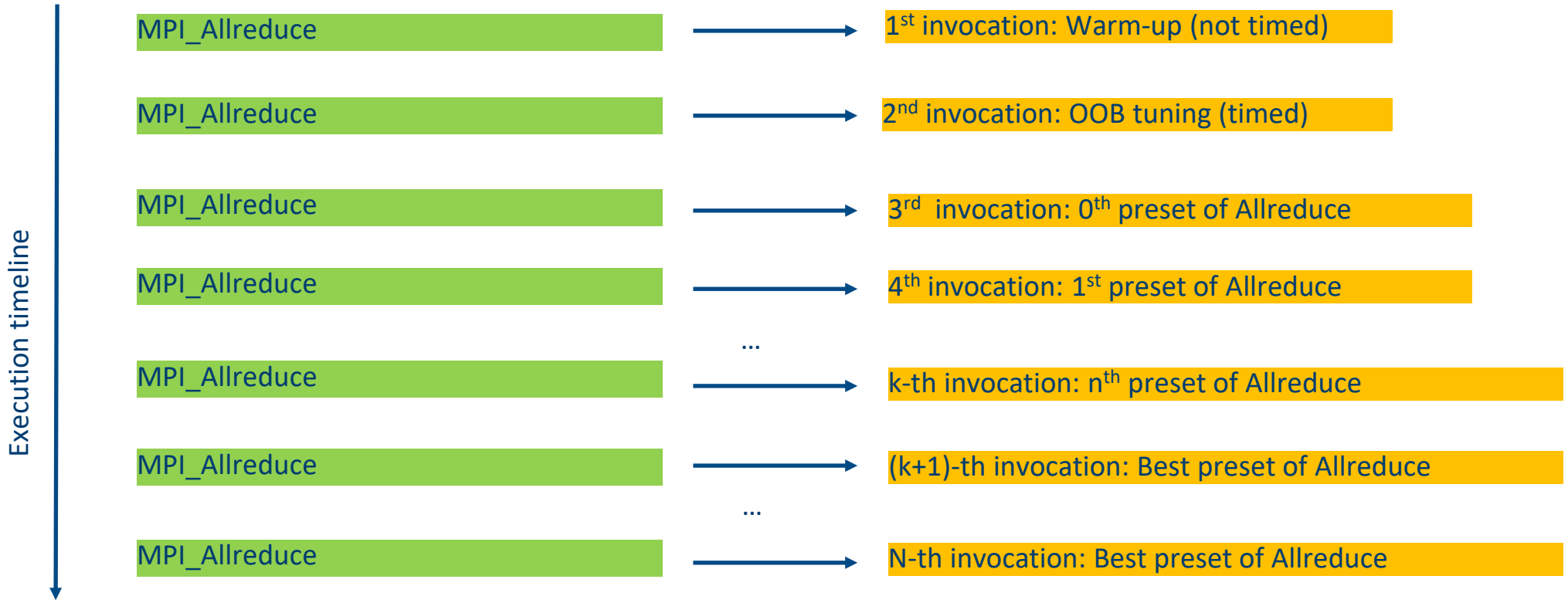
I_MPI_TUNING_AUTO_SYNC=<0|1> Call internal barrier on every tuning iteration (**disabled** by default)

Guidance on I_MPI_TUNING_AUTO_ITER_NUM

Min invocations required for a certain collective call for a certain message size in a certain communicator = I_MPI_TUNING_AUTO_WARMUP_ITER_NUM + [(range+1)* I_MPI_TUNING_AUTO_ITER_NUM]

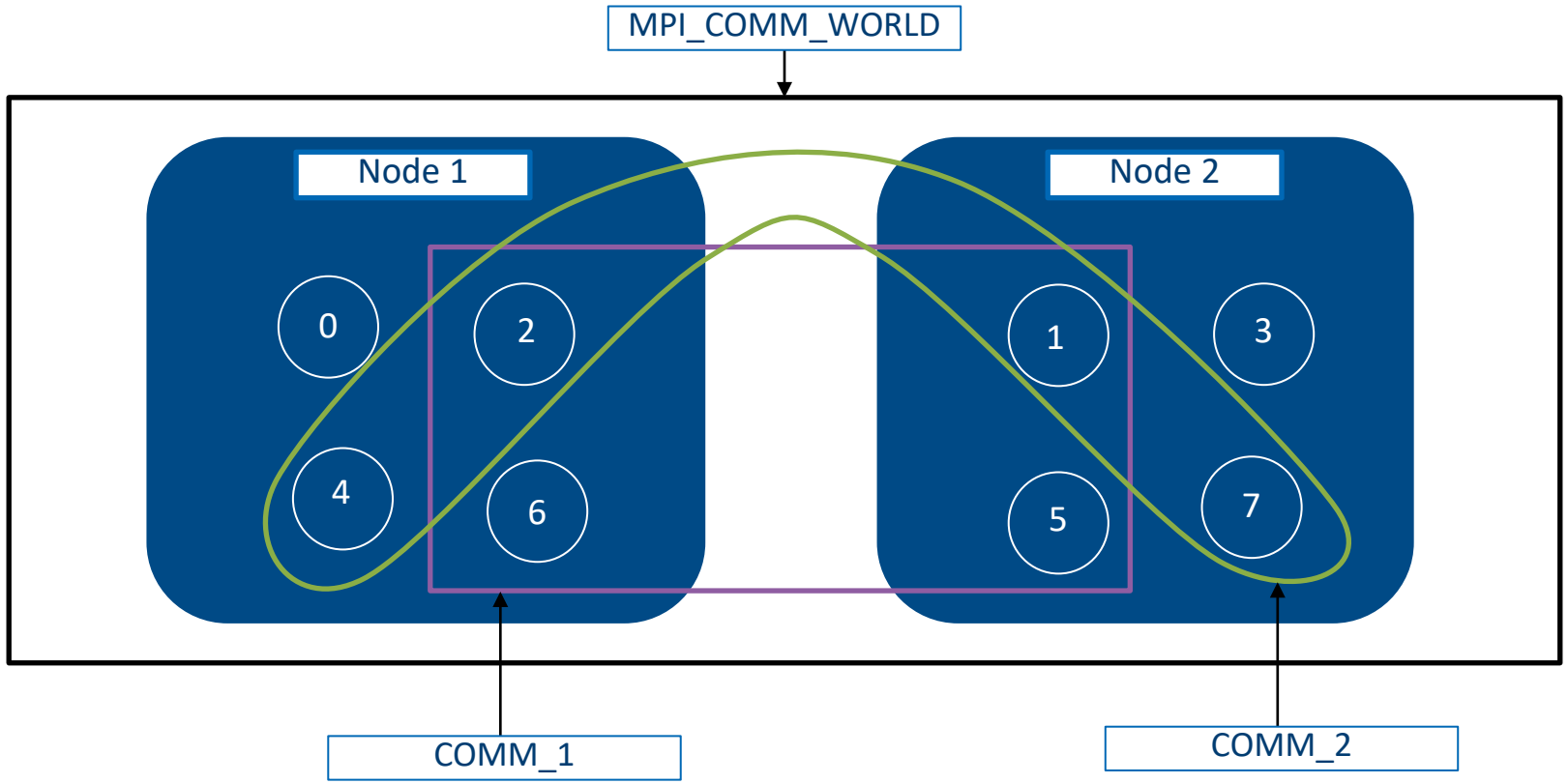
auto is an alias for auto:application

Autotuner – dynamic tuning



- No extra calls. Pure **application driven** tuning
- The procedure is performed for each message size and for each communicator

Autotuner – communicator specific tuning



Each communicator has its own tuning.

(e.g. COMM_1 and COMM_2 have independent tuning)

Autotuner – recommended usage model

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

```
$ I_MPI_TUNING_MODE=auto:cluster I_MPI_TUNING_BIN_DUMP=./tuned.dat mpirun -n 96
-ppn 48 -f hostfile IMB-MPI1 allreduce -iter 1000 -iter_policy off -npmin 96 -
time 4800
```

OR

```
$ I_MPI_TUNING_MODE=auto I_MPI_TUNING_BIN_DUMP=./tuned2.dat mpirun -n 96 -ppn 48
-f hostfile ./app
```

Step 2 – Use the results of autotuner for consecutive launches (optional):

```
$ I_MPI_TUNING_BIN=./tuned.dat mpirun -n 96 -ppn 48 -f hostfile IMB-MPI1
allreduce -iter 1000 -iter_policy off -npmin 96 -time 4800
```

Step 1 variables must be disabled in step 2!

Thank you for your attention

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. Results have been estimated or simulated.

Your costs and results may vary.

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

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

All product plans and roadmaps are subject to change without notice.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.
Notice revision #20110804