The Intel® MPI Library

Michael Steyer, Technical Consulting Engineer, Intel Architecture, Graphics & Software (IAGS)

intel®

NOTICES AND DISCLAIMERS

Refer to https://software.intel.com/en-us/articles/optimization-notice for more information regarding performance and optimization choices in Intel software products.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Agenda

1. Intel® MPI Introduction
2. Fabrics
3. Pinning
4. Tuning
    1. AutoTuner Hands-on!
5. Numerical Reproducibility

Intel® MPI Library

Deliver Flexible, efficient, and Scalable Cluster Messaging

Optimized MPI Application Performance

- Application-specific tuning
- Automatic tuning
- Support for latest Intel® Xeon® Scalable Processors

Lower Latency and Multi-vendor Interoperability

- Industry-leading latency
- Performance-optimized support for the fabric capabilities through OpenFabrics Interfaces (OFI)

Faster MPI Communication

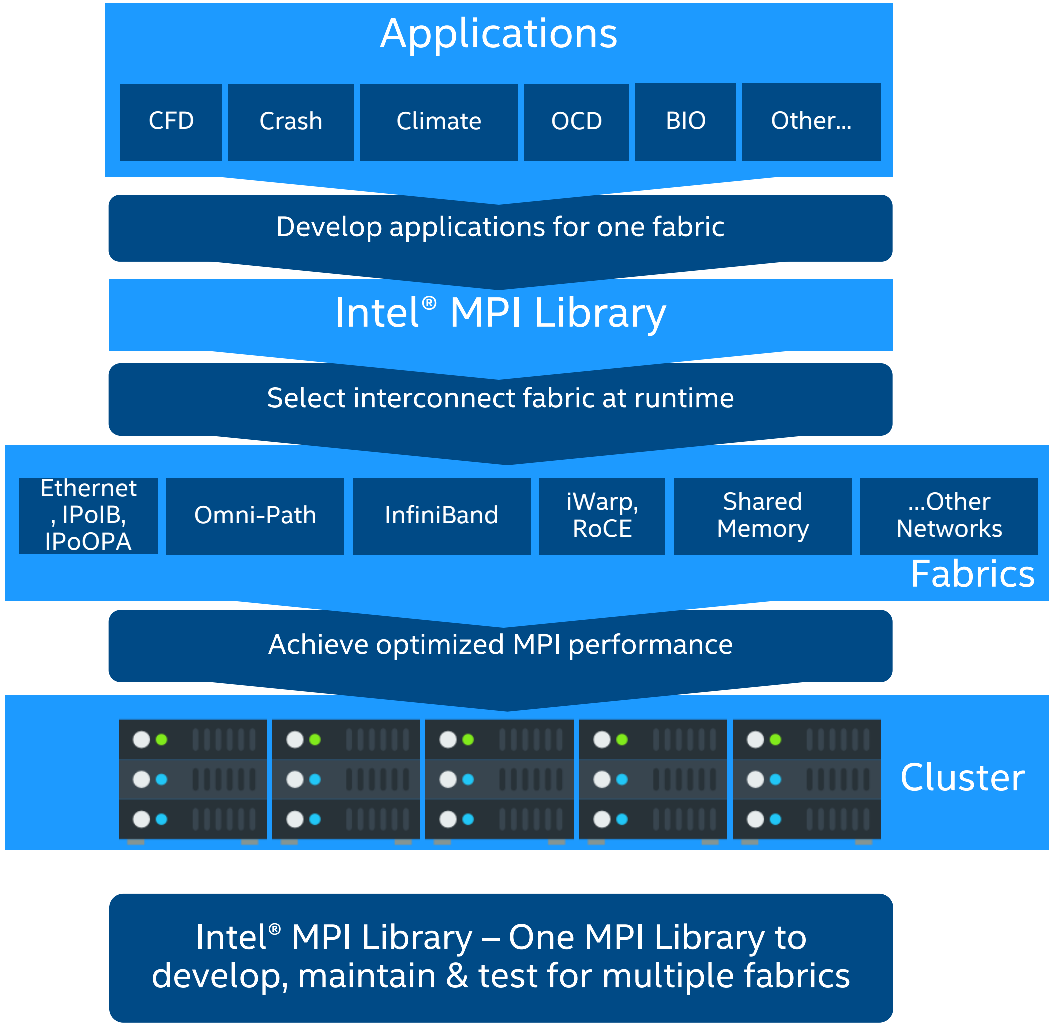
- Optimized collectives

Sustainable scalability

- Native InfiniBand interface support allows for lower latencies, higher bandwidth, and reduced memory requirements

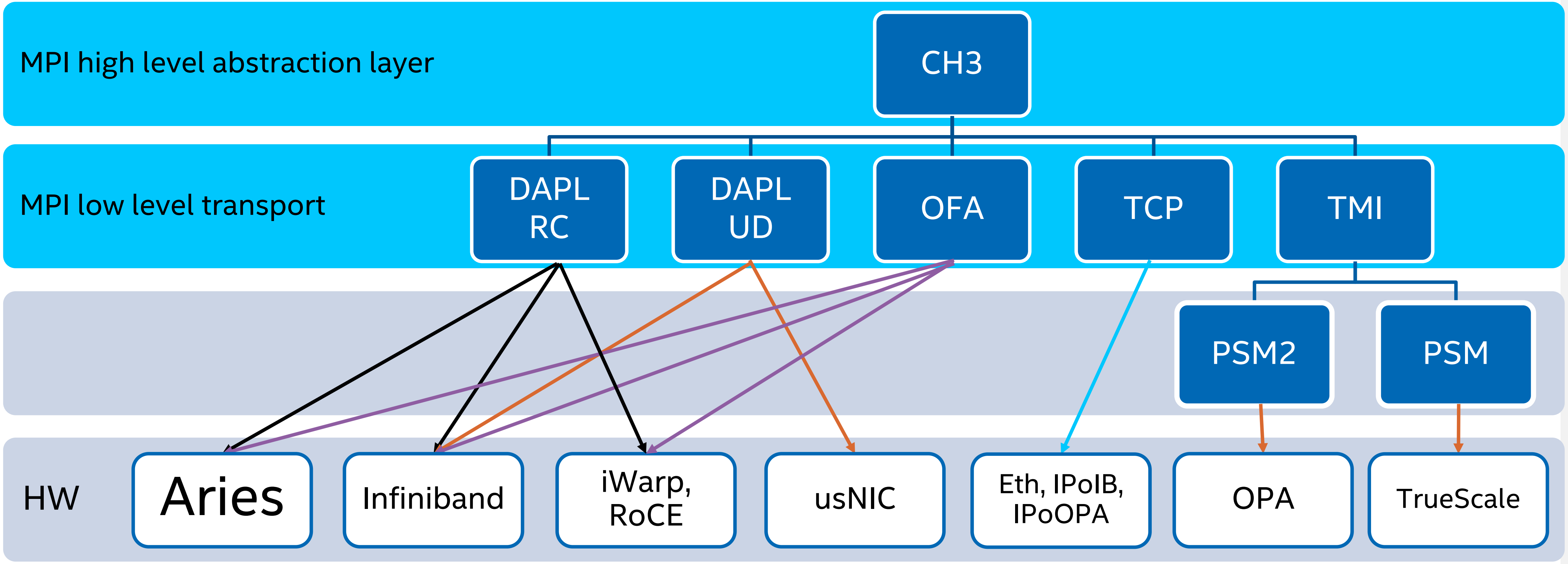
Key Updates

- Intel® GPU pinning support
- Distributed Asynchronous Object Storage (DAOS) support
- Intel® Xeon® Platinum processor 92XX optimizations
- Mellanox ConnectX: 3/4/5/6 (FDR/EDR/HDR) support enhancements



Fabrics

Intel® MPI library 2018 SW stack

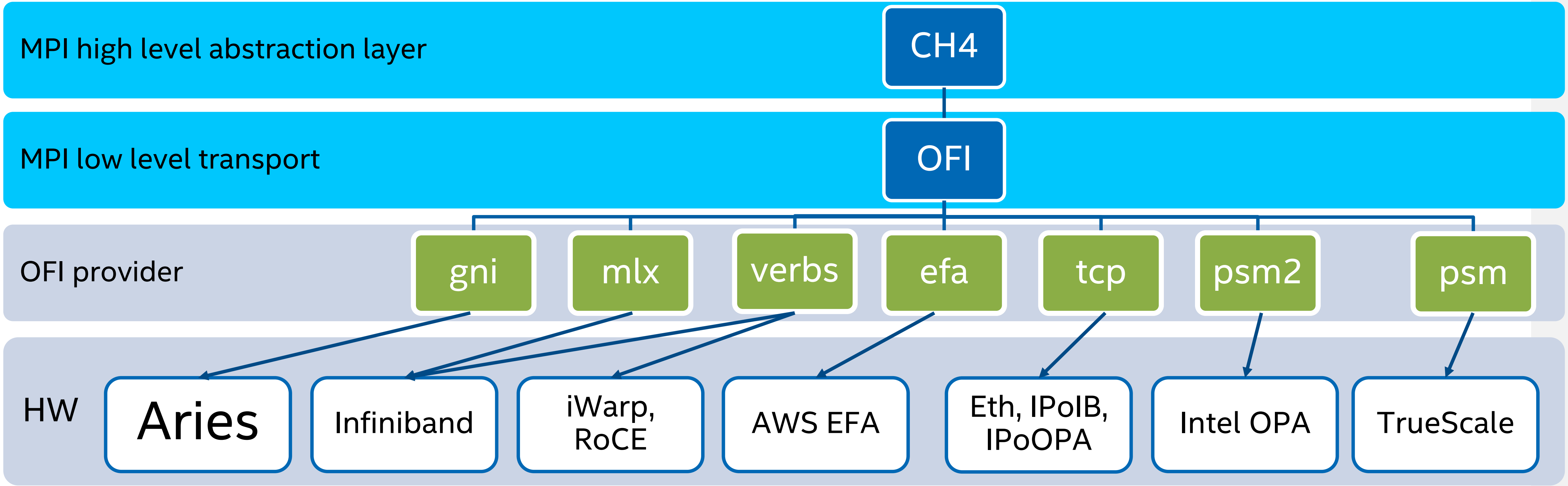


Each transport layer required independent optimization

Intel® MPI library 2019+ SW stack

OFI community

http://libfabric.org/



IMPI 2019 U8 is shipped with a prebuilt libfabric (mlx, psm2, verbs, and tcp providers)

Support for InfiniBand* Fabrics

- LibFabric verbs currently supports only the RC mode
- Stability and performance via verbs is sub-optimal
- IMPI 2019 U5 introduces custom (IMPI specific) libfabric mlx provider
- Hardware support for Dynamic Connection (DC) mode introduced with EDR* and newer

Requirements

- Intel® MPI Library 2019 Update 5 or higher
- Mellanox UCX* Framework v1.4 or higher (Mellanox* OFED)

Pinning

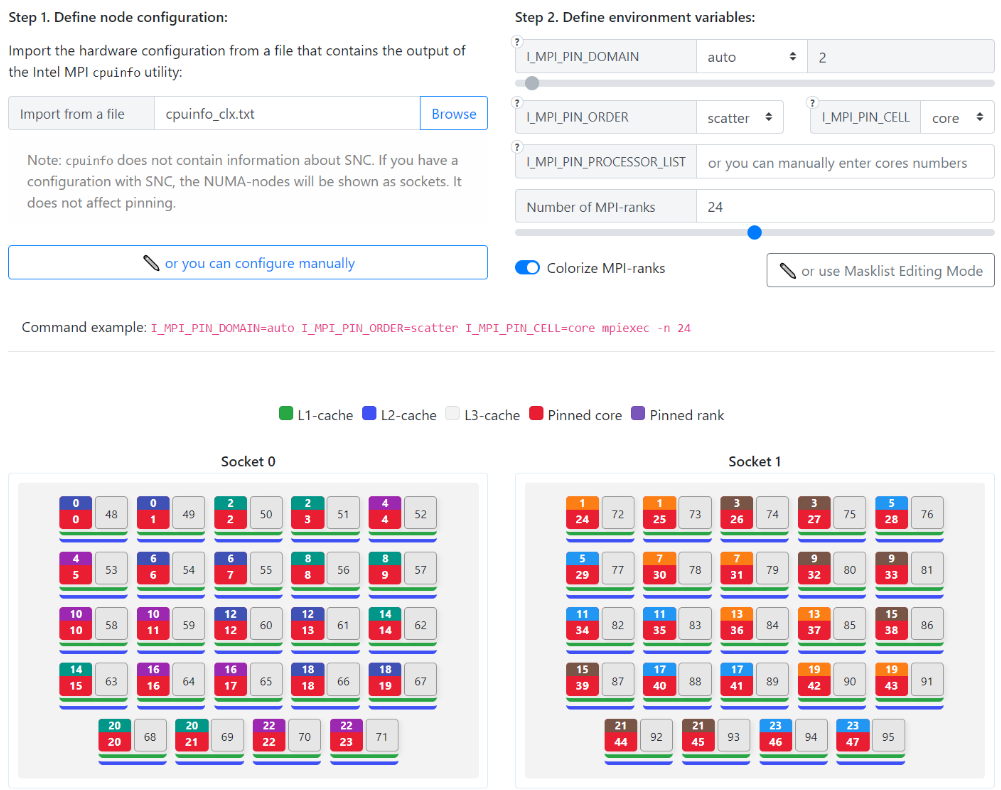
Process Pinning with Intel MPI

Default Intel Library MPI pinning	Impact
I_MPI_PIN=on	Pinning Enabled
I_MPI_PIN_MODE=pm	Use Hydra for Pinning
I_MPI_PIN_RESPECT_CPUSET=on	Respect process affinity mask
I_MPI_PIN_RESPECT_HCA=on	Pin according to HCA socket
I_MPI_PIN_CELL=unit	Pin on all logical cores
I_MPI_PIN_DOMAIN=auto:compact	Pin size #lcores/#ranks : compact
I_MPI_PIN_ORDER=compact	Order domains adjacent

The Intel MPI Pinning Simulator

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

- Starting with IMPI 2019U8
- Web- based interface -
- Platform configuration options
  - load configuration by importing cpuinfo (IMPI utility) output
  - or manually define platform configuration
- Provides IMPI environment variable settings for desired pinning



Custom Mask (left) and 4 Socket Config (right)

Step 1. Define node configuration:

Import the hardware configuration from a file that contains the output of the Intel MPI `cpuinfo` utility:

Import from a file | cpuinfo_clx.txt | Browse

Note: `cpuinfo` does not contain information about SNC. If you have a configuration with SNC, the NUMA-nodes will be shown as sockets. It does not affect pinning.

or you can configure manually

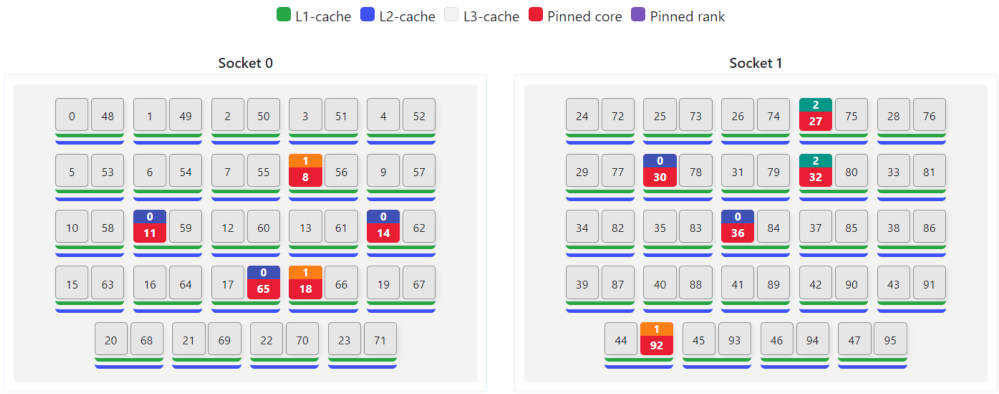
Step 2. Masklist Editing Mode:

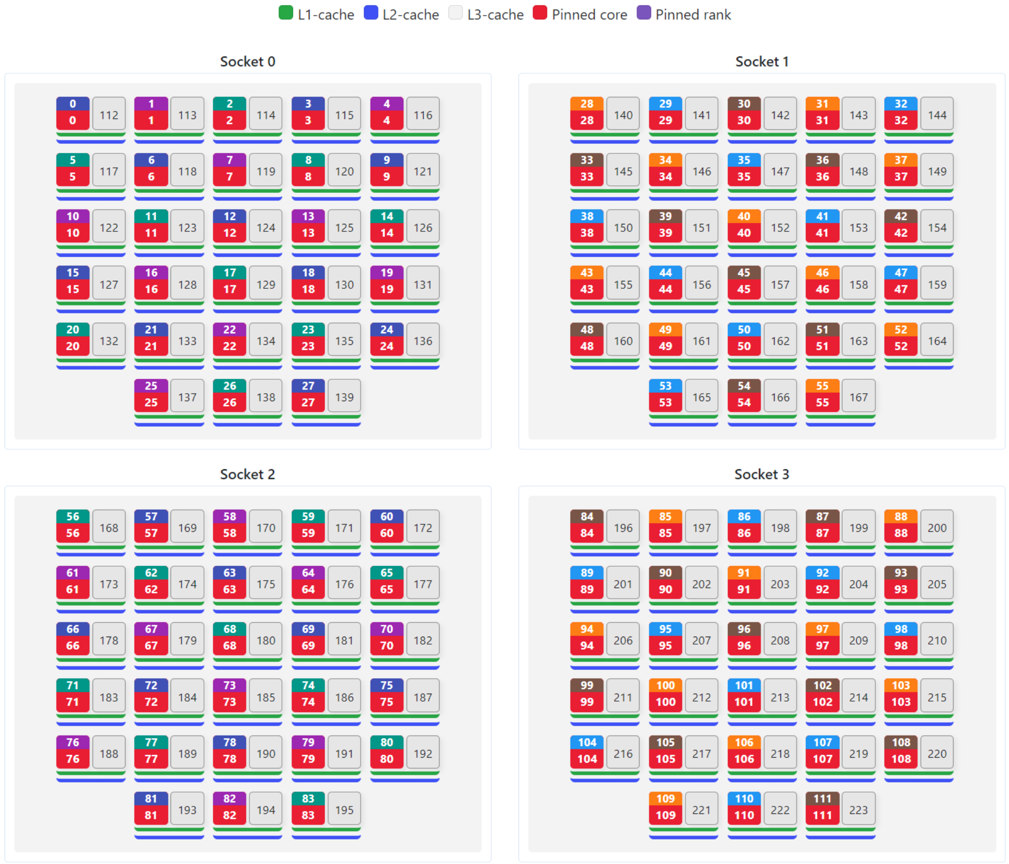
In this mode, you can manually click on the processor cores to pin MPI-ranks to them. As a result, a masklist for `I_MPI_PIN_DOMAIN` will be generated.

- After selection, click the "**Next domain**" button to proceed to pinning the next domain or the "**Clear**" button to start all again.
- You can cancel the pinning of a specific MPI-rank within the current domain **by clicking** on the processor core again.
- You can also hold down the left mouse button and move the mouse to select several cores at once.

Next domain | Clear | Exit

Command example: `I_MPI_PIN_DOMAIN=[20000001040004800,1000000000000000000040100,108000000] mpiexec -n 3`





Tuning

Intel MPI Tuning

- 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.

Intel MPI Tuning

1. mpitune
2. Fast tuner

3. ***Autotuner***
4. mpitune_fast

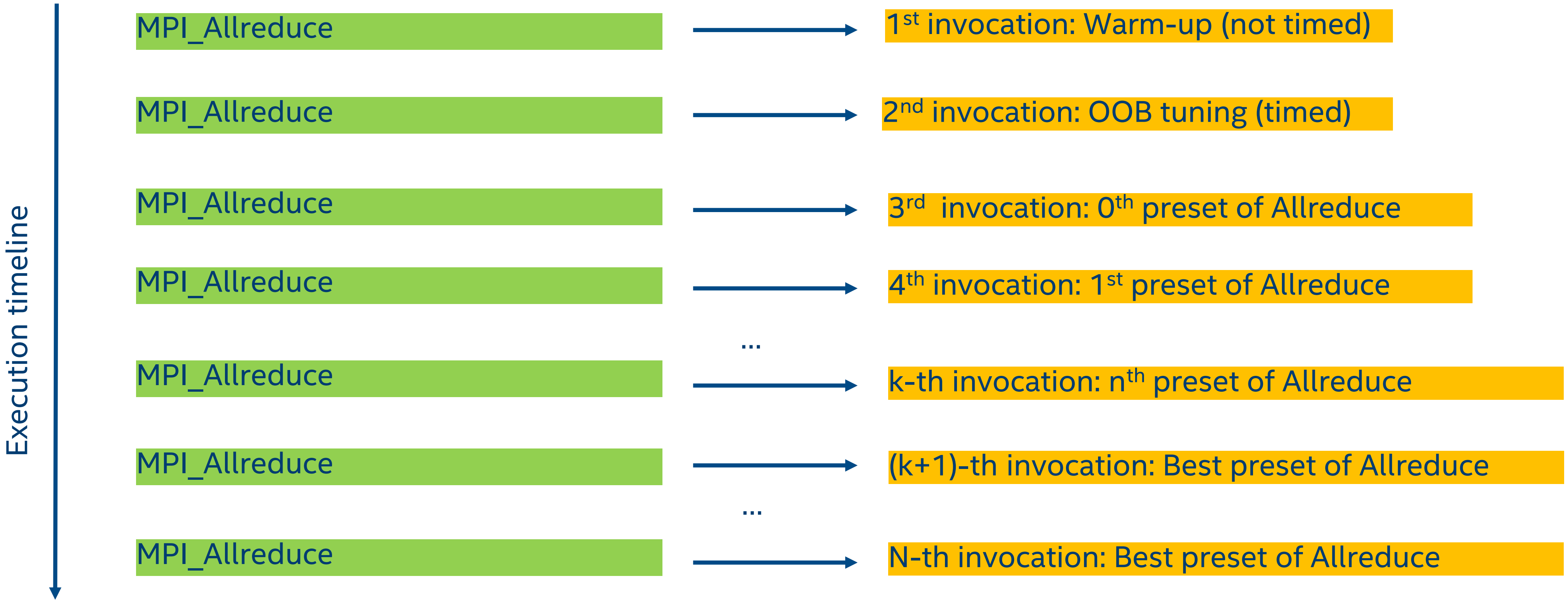
IMPI 2018 and before | IMPI 2019 and beyond

Good
Ok
Bad

Introduction

Tuning utility / Parameter	MPItune	Fast Tuner	Autotuner	mpitune_fast
Tuning overhead	Bad	Ok	Good	Good
Ease of use	Bad	Bad	Good	Good
Application tuning	Bad	Ok	Good	Bad
Microbenchmark tuning	Good	Good	Good	Good
Adoption in production environments	Bad	Bad	Good	Good

Autotuner – dynamic tuning



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

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]

Get started with the autotuner

1. Step 1 – Enable autotuner and store results (store is optional):
   - $ export I_MPI_TUNING_MODE=auto
   - $ export I_MPI_TUNING_BIN_DUMP=./tuning_results.dat
   - $ mpirun -n 96 -ppn 48 IMB-MPI1 allreduce -time 4800
2. Step 2 – Use the results of autotuner for consecutive launches (optional):
   - $ unset I_MPI_TUNING_MODE
   - $ export I_MPI_TUNING_BIN=./tuning_results.dat
   - $ mpirun -n 96 -ppn 48 IMB-MPI1 allreduce -time 4800

NOTE: You may adjust number of tuning iterations (minimal overhead/maximum precision balance) and use autotuner with every application run without results storing.

Autotuner Example

Configuration possibly slowing down tuning run in favour of results.:

- I_MPI_TUNING_MODE=auto
- I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=1
- I_MPI_TUNING_AUTO_ITER_NUM=64
- I_MPI_TUNING_AUTO_SYNC=1
- I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=4194304
- I_MPI_TUNING_AUTO_STORAGE_SIZE=4194304
- I_MPI_TUNING_BIN_DUMP=./my_tuning_file.dat

Apply tuning results via

- I_MPI_TUNING_BIN=./my_tuning_file.dat

Merging tuning files

It is possible to merge tuning files over time and generate a master tuning file if required.

```
$ I_MPI_TUNING_BIN=tuned1.dat,tuned2.dat
  I_MPI_TUNING_BIN_DUMP=./tuned_merged.dat mpirun -n 1 ./dummy_mpi_app
```

- In case of conflicts between tuning files, left most one gets higher priority.
- IMPI runtime accepts multiple tuning files through I_MPI_TUNING_BIN.

mpitune_fast

	Autotuner	mpitune_fast
Scope	**Application specific tuning**	**Cluster wide tuning**
Intended for	**Regular users**	**System administrators**

- tunes the Intel® MPI Library to the cluster configuration using autotuner functionality.
- iteratively launches the Intel® MPI Benchmarks with the proper autotuner environment and generates a tuning file.
- supports Slurm and LSF job managers. mpitune_fast automatically finds job allocated hosts and performs launches.
- **Example**

```
$ mpitune_fast -f ./hostfile -c alltoall,allreduce,barrier
```

Hands-On Intel MPI Autotuner

1) $ cp -r /lrz/sys/courses/hcow1w21/impi_labs . && cd impi_labs
2) $ ./compile.sh && sbatch impi_at.sh
3) Take some time to study impi_at.sh
4) Wait for the job to finish, study the output files
5) Feel free to change the benchmark or the tuning parameters for your own experiments

Numerical Reproducibility

Motivation: Numerical Reproducibility

```fortran
program rep
  use mpi
  implicit none
  integer :: n_ranks,rank,errc
  real*8 :: global_sum,local_value

  call MPI_Init(errc)
  call MPI_Comm_size(MPI_COMM_WORLD, n_ranks, errc)
  call MPI_Comm_rank(MPI_COMM_WORLD, rank, errc)

  local_value = 2.0 ** -60

  if(rank.eq.15) local_value= +1.0
  if(rank.eq.16) local_value= -1.0

  call
MPI_Reduce(local_value,global_sum,1,MPI_DOUBLE_PRECISION, &
         MPI_SUM,0,MPI_COMM_WORLD, errc)

  if(rank.eq.0) write(*,'(f22.20)') global_sum

  call MPI_Finalize(errc)
end program rep
```

```
$ cat ${machinefile_A}
ehk248:16
ehs146:16
ehs231:16
ehs145:16
$ cat ${machinefile_B}
ehk248:32
ehs146:32
ehs231:0
ehs145:0
$ mpiifort -fp-model strict -o ./rep.x ./rep.f90

$ export I_MPI_ADJUST_REDUCE=3
$ mpirun -n 64 -machinefile ${machinefile_A} ./rep.x
0.00000000000000000000
$ mpirun -n 64 -machinefile ${machinefile_B} ./rep.x
0.00000000000000004163

$ export I_MPI_ADJUST_REDUCE=1
$ mpirun -n 64 -machinefile ${machinefile_A} ./rep.x
0.00000000000000004163
$ mpirun -n 64 -machinefile ${machinefile_B} ./rep.x
0.00000000000000004163
```

Conditional Numerical Reproducibility with IMPI

I_MPI_CBWR – Conditional BitWise Reproducibility

Repeatable	Provides consistent results if the application is launched under exactly the same conditions – repeating the run on the same machine- and configuration.
Reproducible (conditionally)	Provides consistent results even if the distribution of ranks differs, while the number of ranks (& #threads for hybrid applications) involved has to be stable. Also, the runtime including the microarchitecture has to be consistent.

I_MPI_CBWR <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)

QUESTIONS?