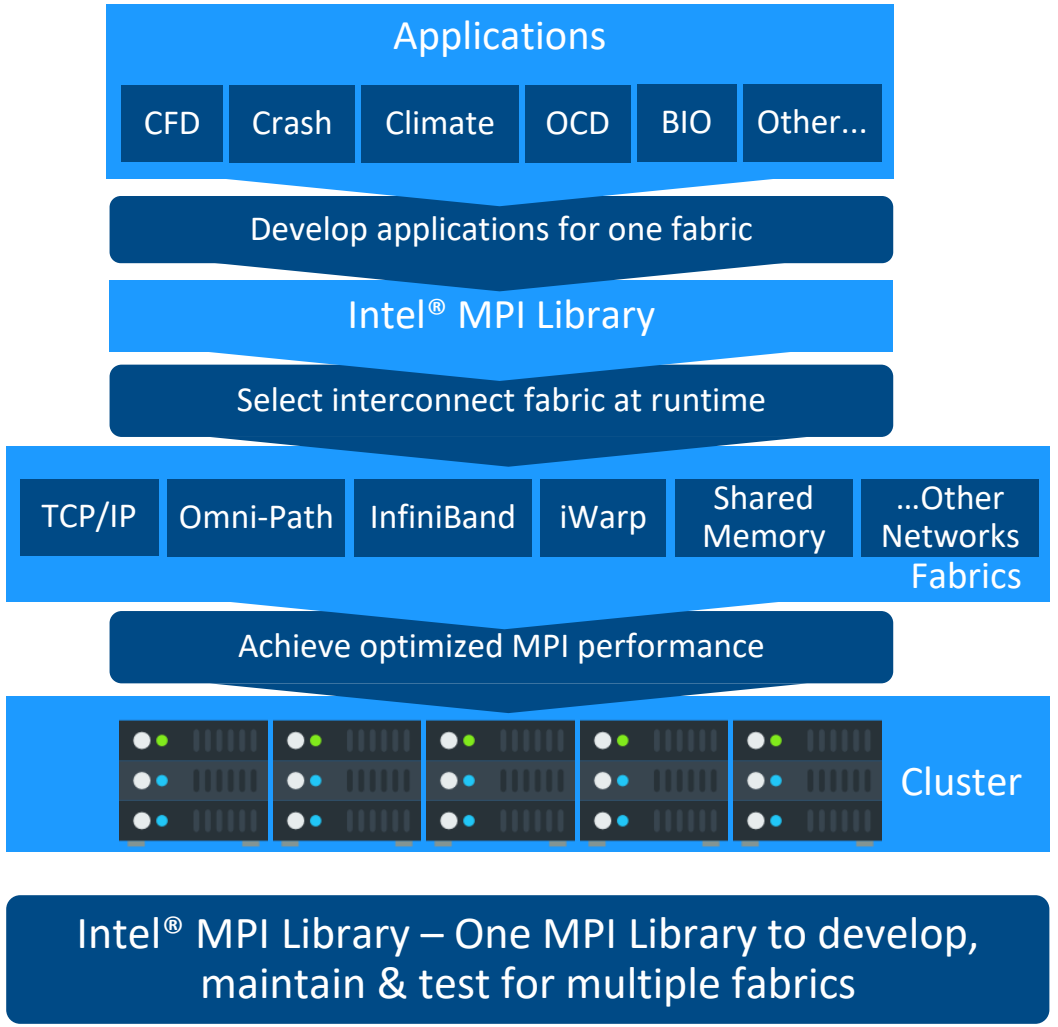
HLRN User Workshop 3-6 Nov 2020

Intel MPI Basics

Klaus-Dieter Oertel

Intel® MPI Library Overview

- Intel® MPI Library is a multifabric message-passing library that implements the open-source MPICH specification. Use the library to create, maintain, and test advanced, complex applications that perform better on HPC clusters based on Intel® processors.
- Develop applications that can run on multiple cluster interconnects chosen by the user at run time.
- Quickly deliver maximum end-user performance without having to change the software or operating environment.
- Achieve the best latency, bandwidth, and scalability through automatic tuning for the latest Intel® platforms.
- Reduce the time to market by linking to one library and deploying on the latest optimized fabrics.



Fabrics

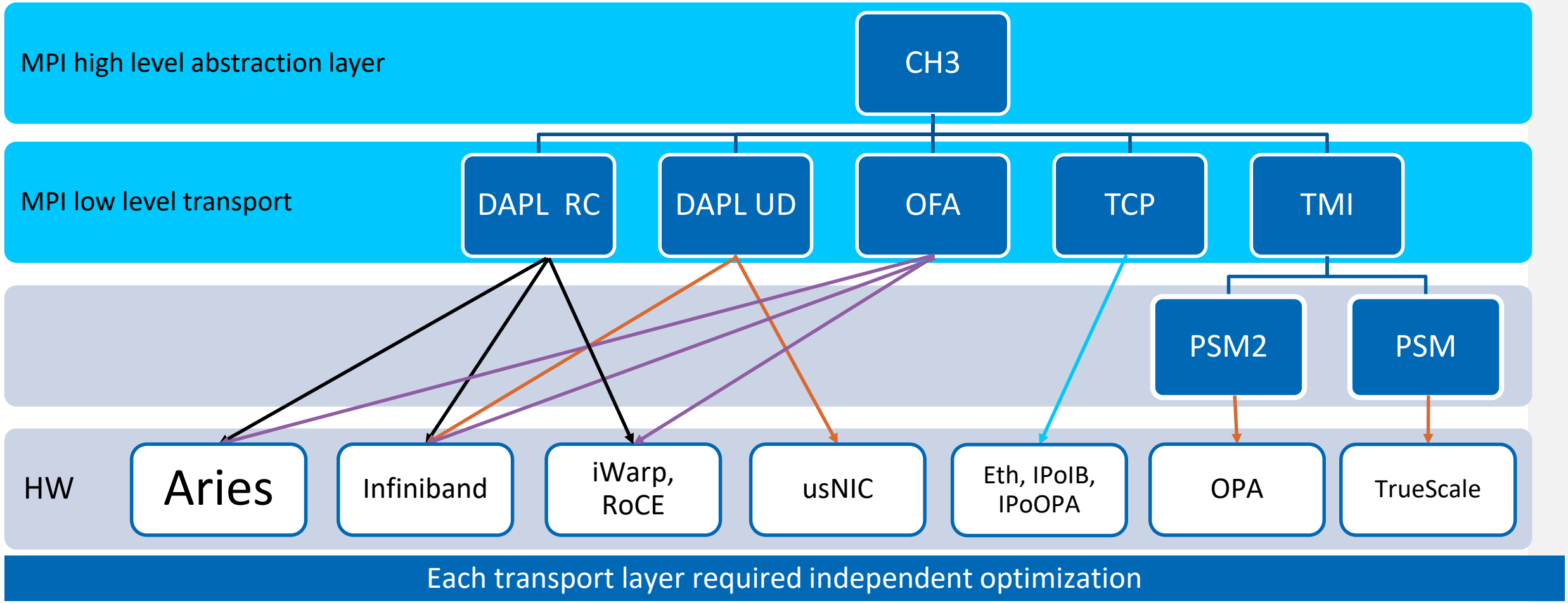
Next generation MPI - Intel® MPI Library 2019

- Next generation product goals:
  - Low instruction count on a critical path
  - Remove non scalable structures
  - Better hybrid programming models support (MPI+X)
  - Better collective operations infrastructure
- Intel® MPI Library 2019 key features:
  - Based on a **new** MPICH/CH4/OFI architecture
  - **New** Mellanox, Amazon AWS and Google GCP support
  - **New** auto tuning capabilities
  - Enhanced support for hybrid programming models
    - **New** MPI_THREAD_MULTIPLE extension
    - **New** asynchronous progress
  - **New** SHM transport
  - **New** collective operations

MPICH/CH4/OFI Architecture Improvements

- Key Features
- Expose native hardware support to the MPI layer
- Reduction in number of instructions (**1.5x** lower instruction count on MPI levels)
  - CH4 uses functions that can be inlined
  - CH3 was based on function pointers
- Removal of non-scalable data structures
  - Driven by Argonne National Laboratory
  - Optimized data structures used to map ranks in a communicator to a global rank
- Enhanced path for MPI+X (threads) models
- OFI netmod
  - Directly maps MPI constructs to OFI features as much as possible

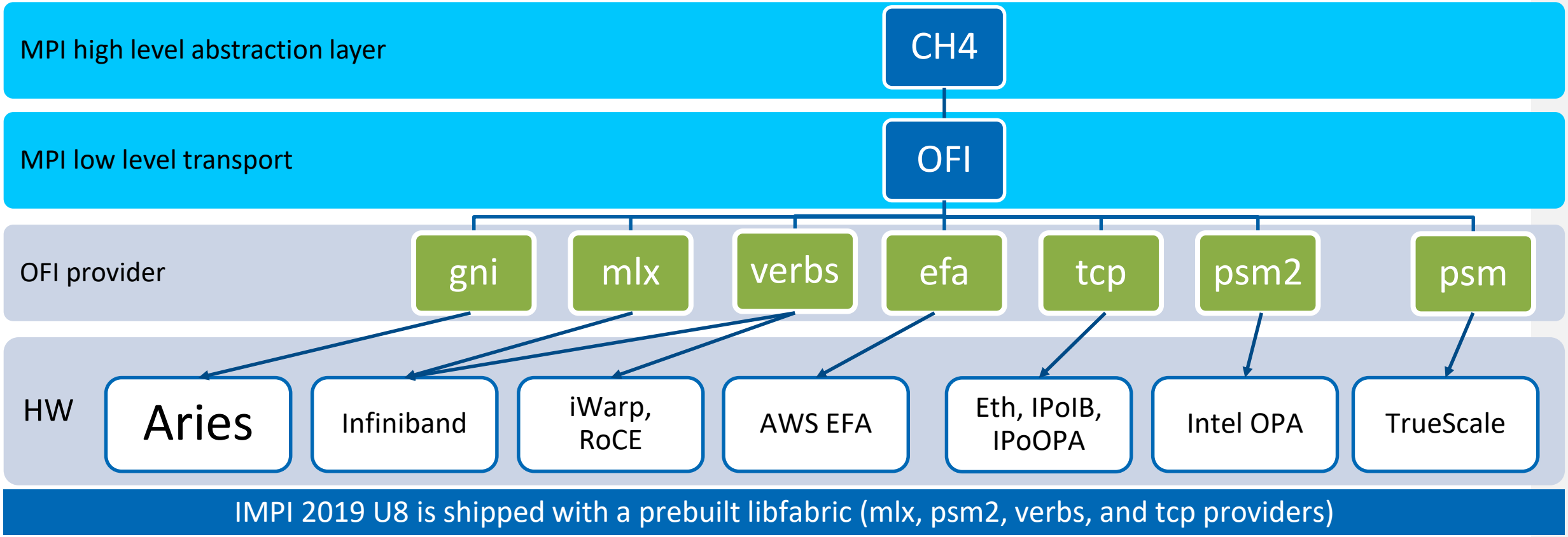
Intel® MPI library 2018 SW stack



Intel® MPI library 2019+ SW stack

OFI community

http://libfabric.org/



Intel MPI at HLRN IV - Lise

- Intel MPI 2018 Update 5
  - `module load impi[/2018.5]` default, (presumably) final 2018 version
- Intel MPI 2019 Update 9
  - `module load impi/2019.9` latest 2019 version
  - `module load impi/2019.5` outdated, deprecated

MPI Pinning

for Performance and Reproducibility

Process Pinning with Intel MPI - 1/3

- The default pinning is suitable for most scenarios
- To override the default process layout, use the -ppn option:

  $ mpirun –ppn <#processes per node> -n <#processes> …
- Intel® MPI Library respects the batch scheduler settings - to overwrite use:

  I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0
- Per-node pinning can also be achieved using a “machinefile”
- Custom processor core pinning can be achieved by two environment variables

  I_MPI_PIN_PROCESSOR_LIST - for pure MPI applications

  I_MPI_PIN_DOMAIN - for Hybrid – MPI + Threading applications

Process Pinning with Intel MPI - 2/3

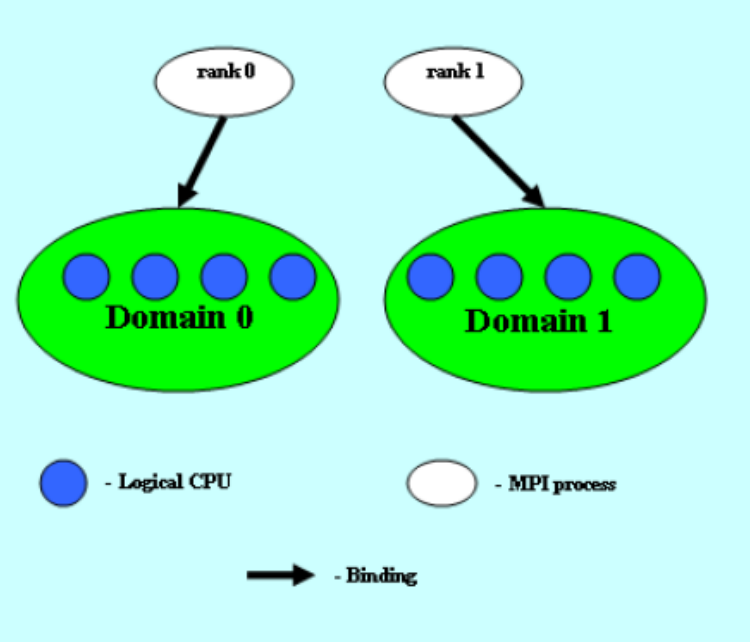
- The ‘cpuinfo’ utility from Intel MPI can be used to observe the processor topology
- Threads of Hybrid applications are not pinned by default
- Threads can migrate along the cores of a rank defined by I_MPI_PIN_DOMAIN
- Therefore, threads should be pinned as well using e.g. OMP_PLACES or KMP_AFFINITY
- Further information can be found in the “Process Pinning” section of the Intel MPI Library reference manual

Process Pinning with Intel MPI - 3/3

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

Intel® MPI Support of Hybrid Codes

- Define **`I_MPI_PIN_DOMAIN`** to split logical processors into non-overlapping subsets
- Mapping rule: **1 MPI process per 1 domain**



For OpenMP: Pin threads inside the domain with **`KMP_AFFINITY`** (or in the code)

- Threading models will “see” the mask of processors from the subset

Intel® MPI Support of Hybrid Codes

- Intel® MPI is strong in mapping and pinning support for MPI processes
- Sophisticated defaults or user controlled:
  - For pure MPI codes use **I_MPI_PIN_PROCESSOR_LIST**
  - For hybrid codes (default, takes precedence over I_MPI_PIN_PROCESSOR_LIST):

I_MPI_PIN_DOMAIN =*<size>[:<layout>]*

<size> =	**omp**	Adjust to OMP_NUM_THREADS
	auto	#CPUs/#MPIprocs (default)
	<n>	Number
<layout> =	**platform**	According to BIOS numbering
	compact	Close to each other
	scatter	Far away from each other

Defines mapping and pinning for MPI processes, leaves room for threads on remaining cores!

Intel® MPI Environment Support

- The execution command mpirun of Intel® MPI reads argument sets from the command line:
  - Sections between „**:**“ define an argument set (alternatively a line in a configfile specifies a set)
  - Host, number of nodes, but also environment can be set independently in each argument set, e.g. for running in symmetric mode on the host and the coprocessor:

```
# mpirun -env I_MPI_PIN_DOMAIN  4 –host myXEON ... \
       : -env I_MPI_PIN_DOMAIN 16 –host myMIC
```

- Adapt the important environment variables to the architecture, e.g. for OpenMP:
  - **OMP_NUM_THREADS**
  - **KMP_HW_SUBSET, KMP_AFFINITY**

OpenMP Environment

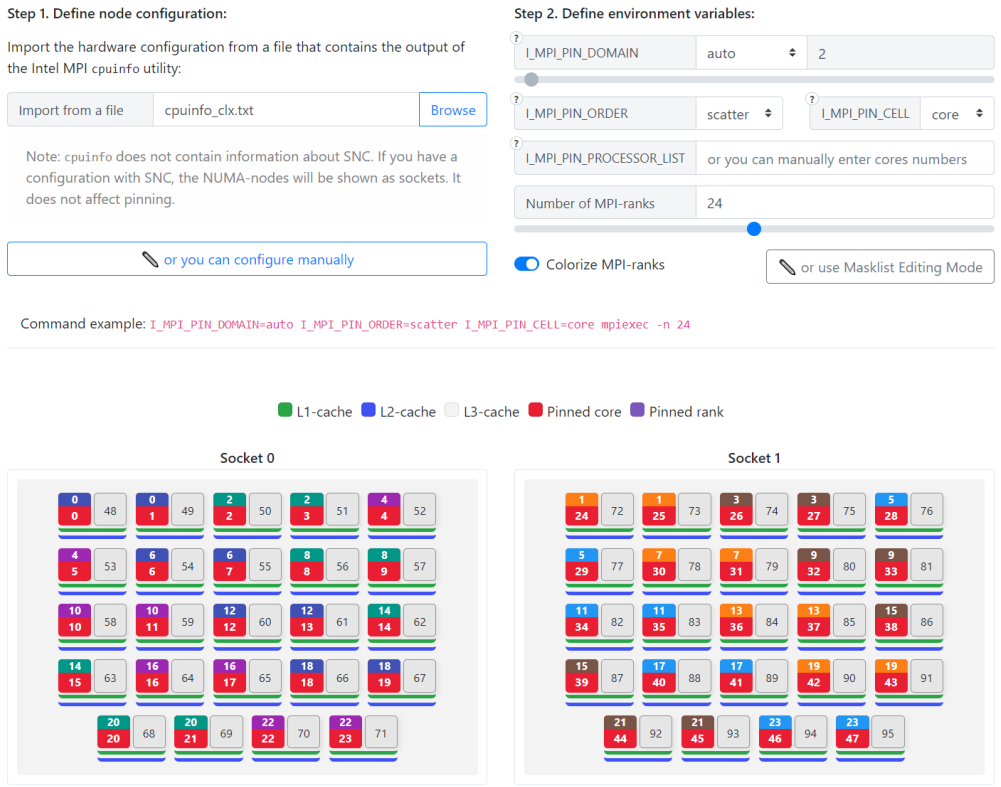
- **`KMP_AFFINITY`** for binding of OpenMP threads in given mask, e.g. Intel MPI domain per rank
- **`KMP_AFFINITY=none`** (default on Xeon): No thread binding
- **`KMP_AFFINITY=scatter`** (default on Xeon Phi): Distributes the threads as evenly as possible across the cores in round robin (using hyper/hardware threads for the 2nd sweep)
- **`KMP_AFFINITY=compact`**: Assigns thread <n>+1 to a free hyper/hardware thread as close as possible to thread <n>, filling one core after the other
- **`KMP_AFFINITY=balanced`**: Form groups of consecutive threads by dividing total #threads by #cores. Place groups in scatter manner on cores. Supported on Xeon Phi and Xeon (for the latter only for single socket systems)
- **`KMP_AFFINITY=verbose,scatter`** to list the thread binding for scatter.
- Details and examples on https://software.intel.com/en-us/cpp-compiler-18.0-developer-guide-and-reference-thread-affinity-interface-linux-and-windows

MPI Pinning Simulator

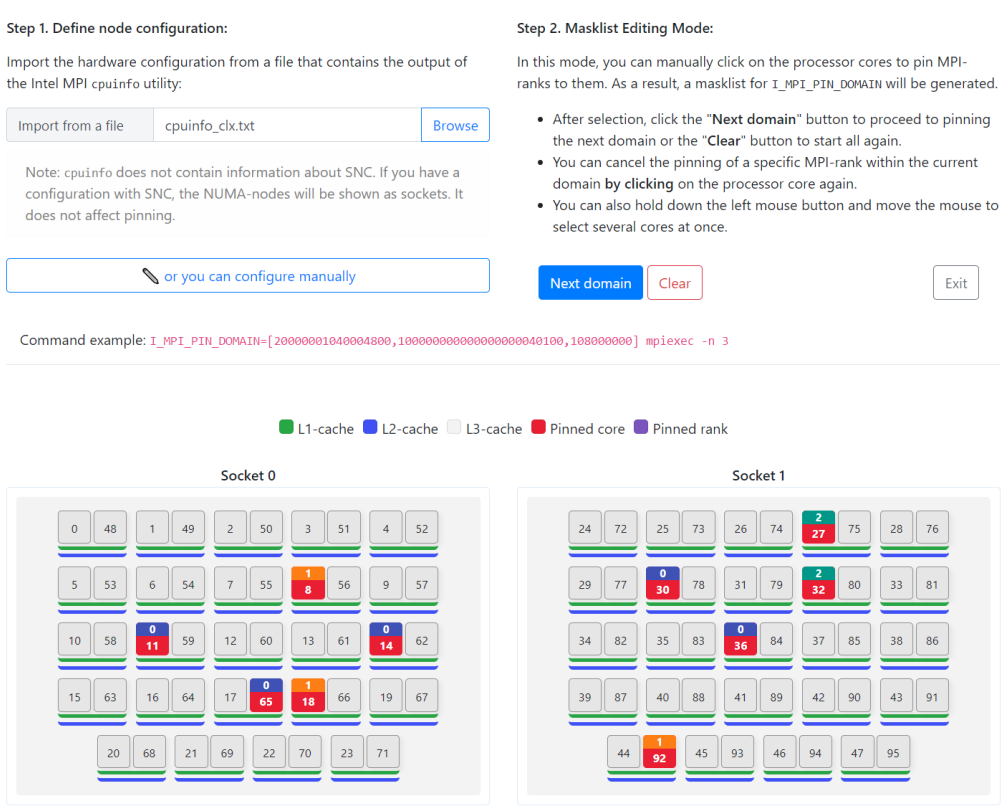
The Intel MPI Pinning Simulator

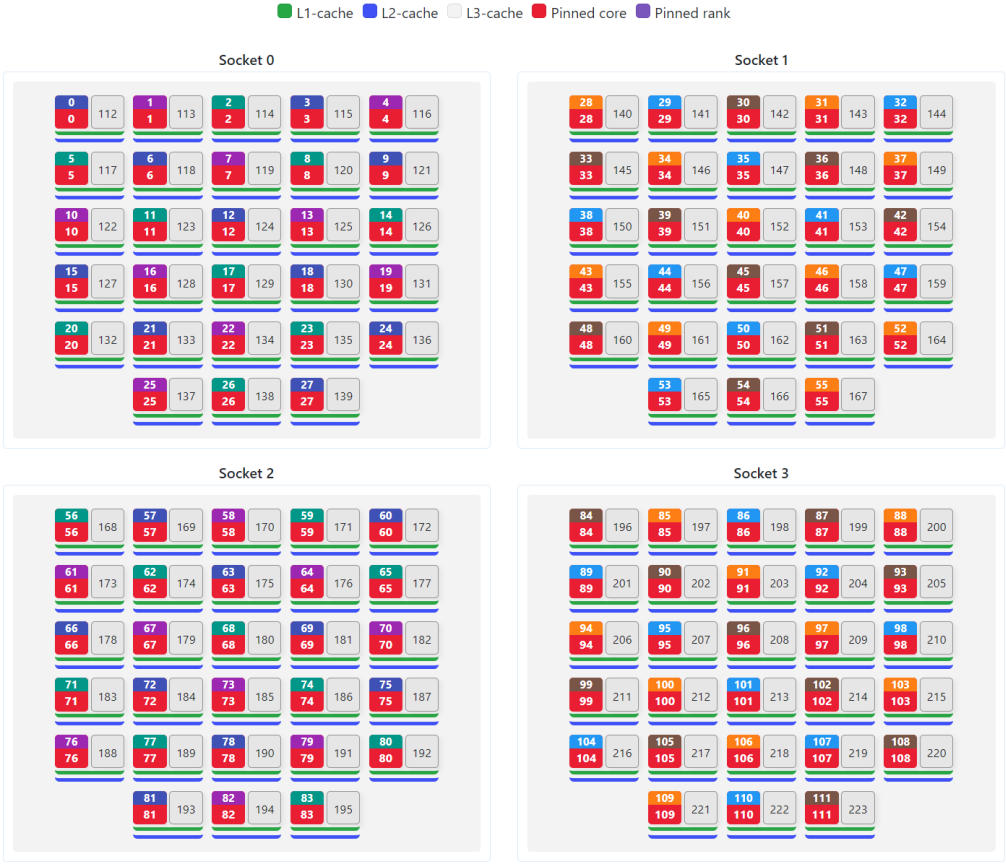
https://software.intel.com/content/www/us/en/develop/articles/pinning-simulator-for-intel-mpi-library.html

- 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)





MPI Pinning Control

at Run Time

Processor Topology Using *cpuinfo*

- Shows important features of a node: number of sockets, cores per socket including hyper-threads and caches
- Part of the Intel® MPI Library distribution
- Reads its data from /proc/cpuinfo and prints it in a more appropriate format

  /proc/cpuinfo is over 2000 lines long for a 40 core HT dual socket Skylake

  cpuinfo output for the same system is just about 100 lines and output is clearly organized
- Useful to guide process binding decisions

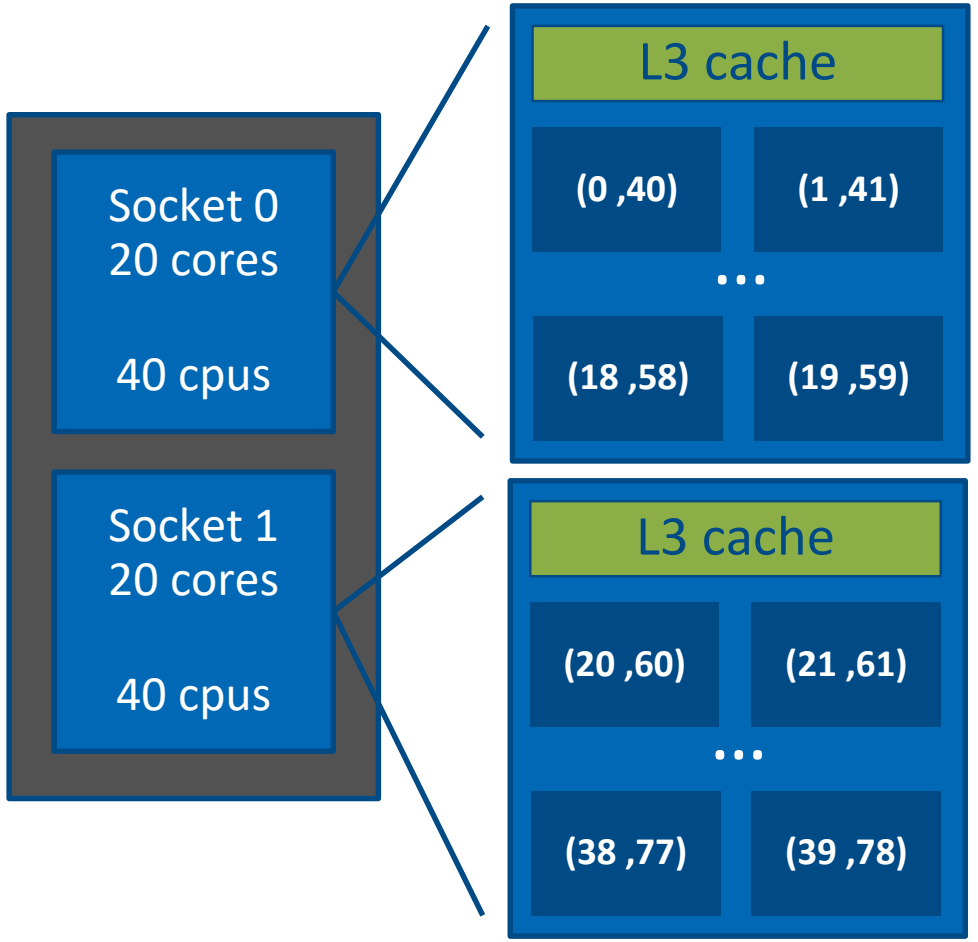
Output from *cpuinfo*

```
=====  Processor composition  =====
Processor name    : Intel(R) Xeon(R) Gold 6148
Packages(sockets) : 2
Cores             : 40
Processors(CPUs)  : 80
Cores per package : 20
Threads per core  : 2

=====  Processor identification  =====
Processor     Thread Id.    Core Id.    Package Id.
0             0             0           0
...
79            1             28          1

=====  Placement on packages  =====
Package Id.      Core Id.         Processors
0                0,1,...,28       (0,40)(1,41)...
1                0,1,...,28       (20,60)(21,61)...

=====  Cache sharing  =====
Cache    Size          Processors
L1       32  KB        (0,40)(1,41)...(20,60)(21,61)...
L2       1   MB        (0,40)(1,41)...(20,60)(21,61)...
L3       27  MB        (0,1,2,...) (20,21,22,...)
```



Runtime Information with I_MPI_DEBUG

- The I_MPI_DEBUG variable controls the information printed to stdout
- Values from 1 to 6 provide increasing levels of verbosity
- A balance setting, I_MPI_DEBUG=4, prints information about:
  - Process pinning
  - Used network interfaces
  - Intel MPI Library environment variables set by the user

Default Binding

```
$ export I_MPI_DEBUG=4
$ mpirun -machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid    Node name Pin cpu
[0] MPI startup(): 0      121023 node0     {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI startup(): 1      121024 node0     {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI startup(): 2      121025 node0     {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI startup(): 3      121026 node0     {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}
[0] MPI startup(): 4      246334 node1     {0,1,2,3,4,5,6,7,8,9,40,41,42,43,44,45,46,47,48,49}
[0] MPI startup(): 5      246335 node1     {10,11,12,13,14,15,16,17,18,19,50,51,52,53,54,55,56,57,58,59}
[0] MPI startup(): 6      246336 node1     {20,21,22,23,24,25,26,27,28,29,60,61,62,63,64,65,66,67,68,69}
[0] MPI startup(): 7      246337 node1     {30,31,32,33,34,35,36,37,38,39,70,71,72,73,74,75,76,77,78,79}

Hi from MPI task 0
...
```

Fabric provider

Active transfer modes

Equal distribution of cores among MPI ranks

Using I_MPI_PIN_PROCESSOR_LIST

```
$ export I_MPI_DEBUG=4
$ export I_MPI_PIN_PROCESSOR_LIST=0,1,20,21
$ mpirun –machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid    Node name Pin cpu
[0] MPI startup(): 0      121023 node0     {0}
[0] MPI startup(): 1      121024 node0     {1}
[0] MPI startup(): 2      121025 node0     {20}
[0] MPI startup(): 3      121026 node0     {21}
[0] MPI startup(): 4      246334 node1     {0}
[0] MPI startup(): 5      246335 node1     {1}
[0] MPI startup(): 6      246336 node1     {20}
[0] MPI startup(): 7      246337 node1     {21}

Hi from MPI task 0
...
```

Fabric provider (no changes)

Active transfer modes (no changes)

Processes bound to specified CPUs, not floating

Using I_MPI_PIN_DOMAIN

```
$ export I_MPI_DEBUG=4
$ export I_MPI_PIN_DOMAIN=omp
$ export OMP_NUM_THREADS=10
$ mpirun –machinefile ./hosts.txt -n 8 ./mpi_hello

[0] MPI startup(): Multi-threaded optimized library
[0] MPID_nem_ofi_init(): used OFI provider: psm2
...
[0] MPI startup(): shm and ofi data transfer modes
...
[0] MPI startup(): Rank   Pid    Node name Pin cpu
[0] MPI startup(): 0      121023 node0     {0,1,2,3,4,40,41,42,43,44}
[0] MPI startup(): 1      121024 node0     {5,6,7,8,9,45,46,47,48,49}
[0] MPI startup(): 2      121025 node0     {10,11,12,13,14,50,51,52,53,54}
[0] MPI startup(): 3      121026 node0     {15,16,17,18,19,55,56,57,58,59}
[0] MPI startup(): 4      246334 node1     {0,1,2,3,4,40,41,42,43,44}
[0] MPI startup(): 5      246335 node1     {5,6,7,8,9,45,46,47,48,49}
[0] MPI startup(): 6      246336 node1     {10,11,12,13,14,50,51,52,53,54}
[0] MPI startup(): 7      246337 node1     {15,16,17,18,19,55,56,57,58,59}

Hi from MPI task 0 OMP thread 0
...
```

Set binding to OMP range

Each MPI task floats on OMP_NUM_THREADS logical processors

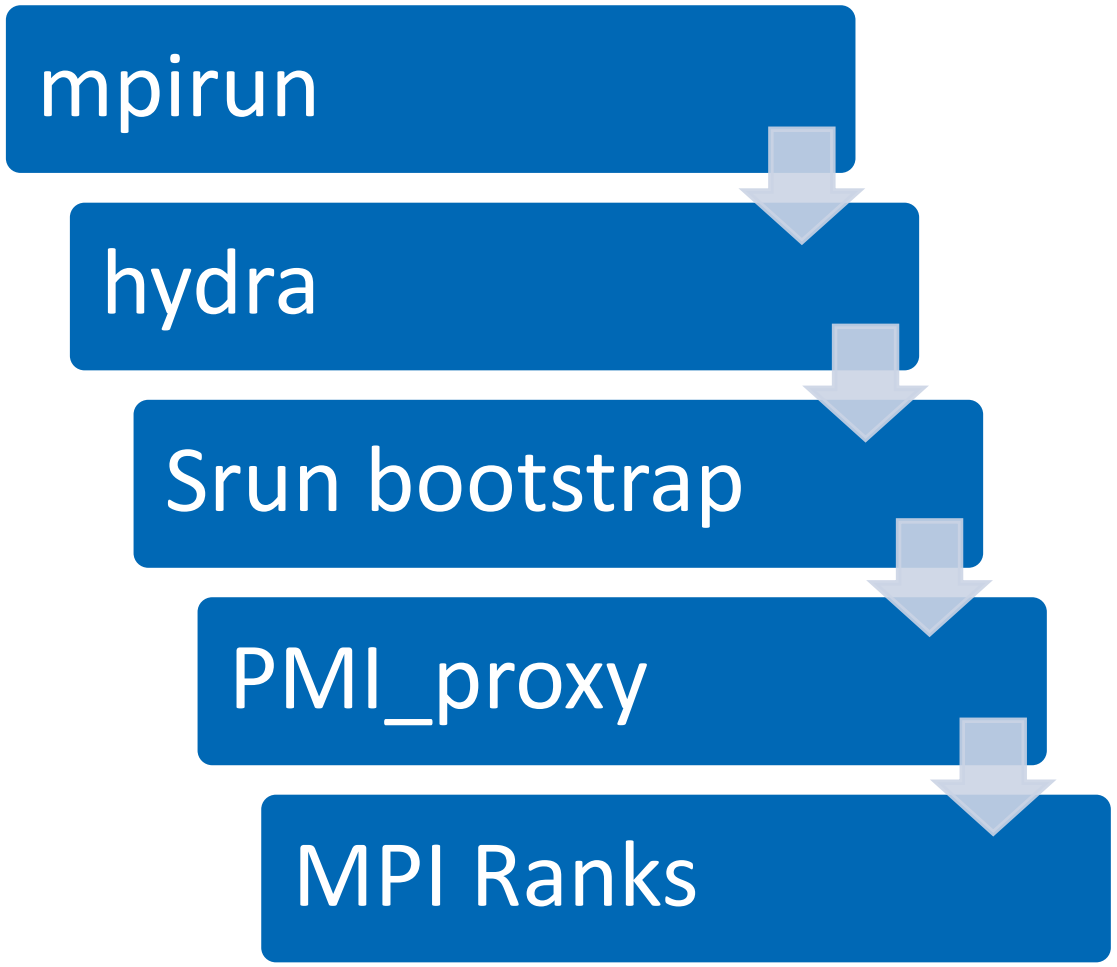
Careful in HT systems!

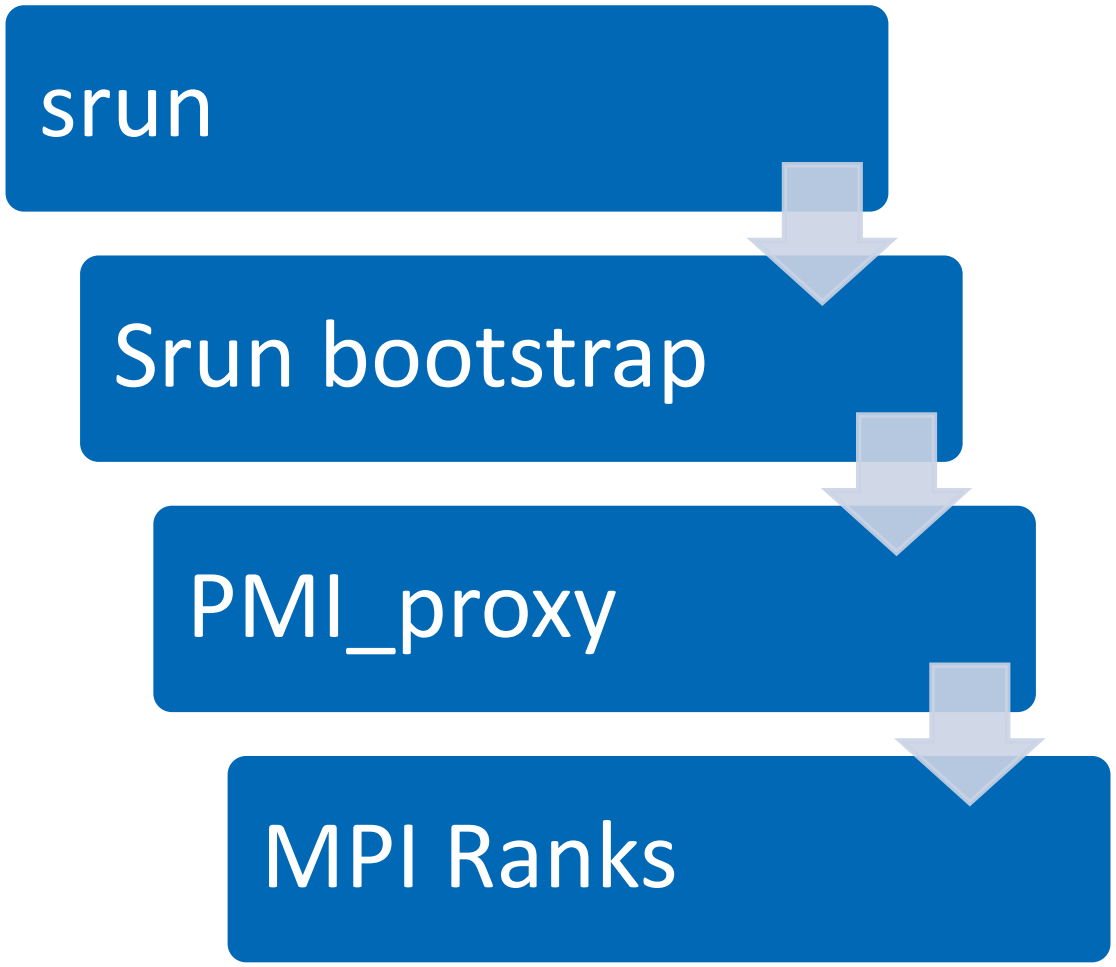
Summary

- Process pinning is the mapping of MPI ranks to hardware resources like cores, sockets, caches etc…
- Default pinning strategy of Intel MPI Library may depend on version, but it generally produces a near-optimal task to processor map
- Performance reproducibility depends strongly on process binding
- To increase performance you should control the pinning, especially for hybrid programs (pinning domains)
- Find out where to bind using cpuinfo
- Find out where things were bound using I_MPI_DEBUG

SLURM Integration

SLURM Process Manager Integration





Intranode Pinning

IMPI	SLURM
`I_MPI_HYDRA_BOOTSTRAP=slurm`	
`I_MPI_PIN_RESPECT_CPUSET=0`	I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so.0
mpirun	srun
ppn etc.	--cpus-per-task / --ntasks-per-node / etc.

Intel MPI and SLURM

- Two approaches for starting (hybrid) MPI codes:
  - `srun`: Process mapping and binding under control of SLURM
    - Always use `--cpu-bind=v[erbose]` to check mapping/binding
  - `mpirun`: Process mapping and binding under control of Intel MPI
    - Only use --nodes from SLURM (no --ntasks/--ntasks-per-node/--cpus-per-task!)

      ```
      mpirun … -n <#ranks> -ppn <#ranks_per_node> <executable>
      ```

      Use environment variables I_MPI_PIN_DOMAIN (hybrid MPI/threading) or I_MPI_PIN_PROCESSOR_LIST (pure MPI) for mapping and binding in nodes
    - Using --nodes and --ntasks (or --ntasks-per-node) from SLURM
      ➔assumption: #tasks/#nodes ranks per node are required by the job

      ```
      mpirun … <executable>
      ```
    - Always use `export I_MPI_DEBUG=4` (or higher) to check mapping/binding
- Binding of OpenMP threads has to be controlled/checked: `export KMP_AFFINITY=verbose`

SLURM Process Manager Integration

IMPI Version	Configuration
<= 2019 U5	The IMPI process launcher (rank) is checking if the I_MPI_PMI_LIBRARY was exposed or not (both srun & mpirun will work)
>= 2019U6	Users must choose either srun or mpirun
>= 2019U7	PMI1 & PMI2 are supported. The IMPI selected PMI depends on the target of I_MPI_PMI_LIBRARY and has to be aligned with the SLURM configuration or aligned with the srun user parameter e.g. –mpi=pmi2
>= 2019U8	Dynamic spawning support with PMI2 under SLURM

Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

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

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. https://software.intel.com/en-us/articles/optimization-notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

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

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

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.


intel