Intel® MPI Tuning

Dr. Heinrich Bockhorst, 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.
Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries.
Other names and brands may be claimed as the property of others.

Agenda

1. Automatic Tuning
2. Multiple Endpoints / Asynchronous progress

Tuning

Motivation

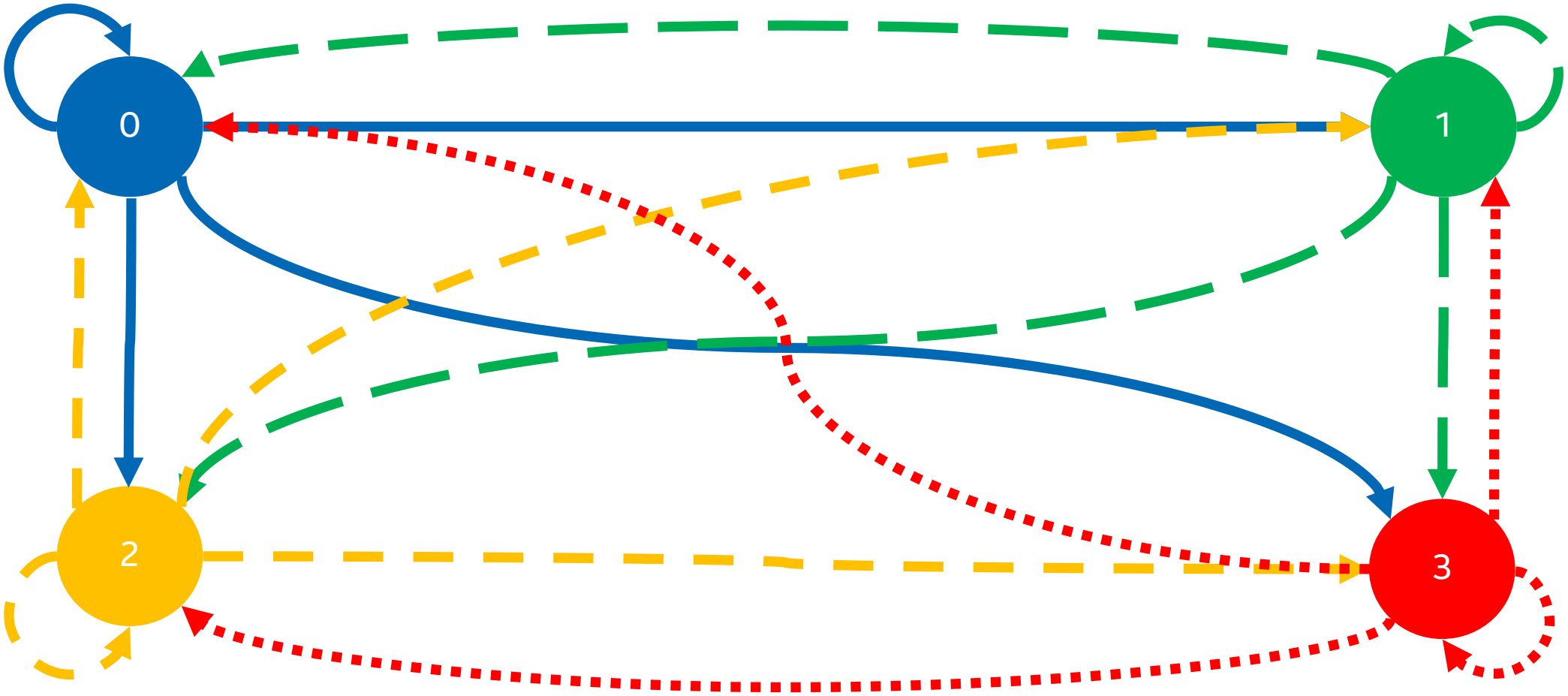
"The **optimal algorithm** and the **optimal buffer** size for a given **message size** depends on a given configuration of the system including the gap values of the networks, memory models, the underlying communication layer etc. The **optimal parameters** for a system **can be best determined by conducting experiments on the system**."

[Automatically Tuned Collective Communications –

Sathish S. Vadhiyar, Graham E. Fagg, Jack Dongarra –

Computer Science Department - University of Tennessee, Knoxville]

Example: MPI_Allgather()



Intel® MPI Allgather Implementation

Environment Variable	Collective Operation	Algorithms
I_MPI_ADJUST_ALLGATHER	MPI_Allgather	1. Recursive doubling algorithm 2. Bruck's algorithm 3. Ring algorithm 4. Topology aware Gatherv + Bcast algorithm 5. Knomial algorithm

I_MPI_ADJUST_<opname>=<algid>[:<conditions>][;<algid>:<conditions>[...]]

Don’t use this Tuning Approach!!!

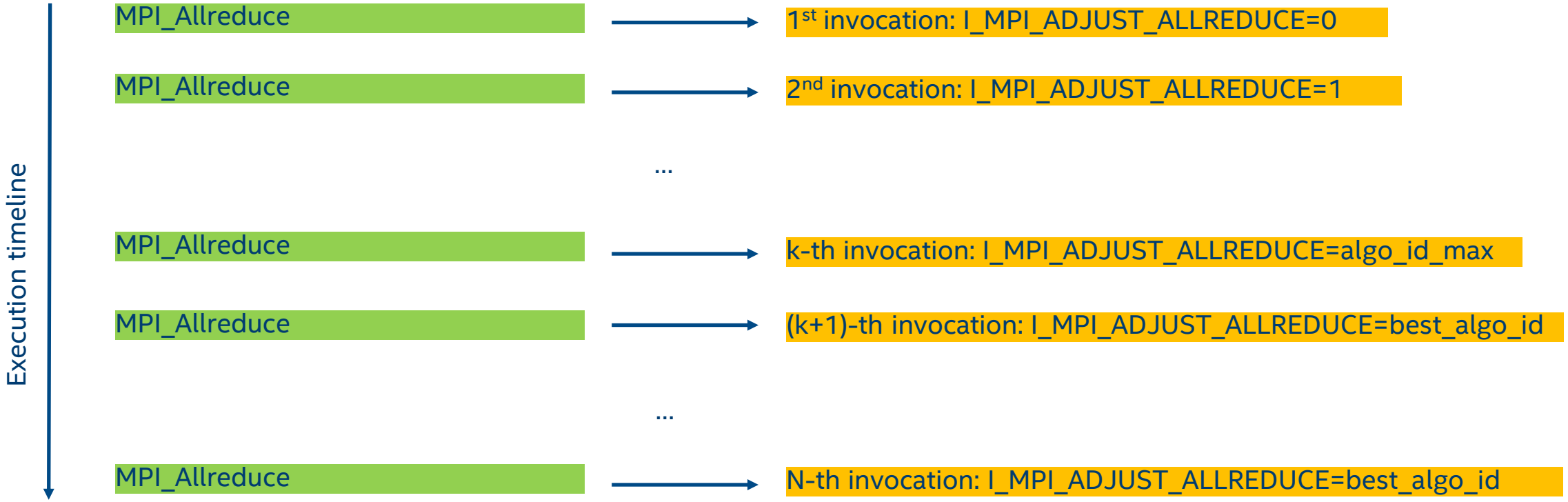
Intel MPI library tuning approaches

	mpitune	mpitune_fast tuner	autotuner
Micro benchmark tuning			
Application tuning			
Easy of use			
Cluster time			
Adoption to environment			

Intel MPI library tuning approaches

	mpitune	mpitune_fast tuner	autotuner
Micro benchmark tuning			
Application tuning			
Easy of use			
Cluster time			
Adoption to environment			

Intel MPI Library 2019 autotuner tuning flow



No extra calls. Pure application driven tuning

The procedure is performed for each message size and for each communicator

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 -iter 1000,800 -time 4800
2. Step 2 – Use the results of autotuner for consecutive launches (optional):
   - $ export I_MPI_TUNING_BIN=./tuning_results.dat
   - $ mpirun -n 96 -ppn 48 IMB-MPI1 allreduce -iter 1000,800 -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.

Environment Variables - Main flow control

- I_MPI_TUNING_MODE=<auto> Enable autotuner (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)
- I_MPI_TUNING_AUTO_WARMUP_ITER_NUM=<number> Warmup iterations number (1 by default).

NOTE: Assume that there are around 30 algorithms to be iterated. E.g. Application has 10000 invocations of MPI_Allreduce 8KB. For full tuning cycle I_MPI_TUNING_AUTO_ITER_NUM may be in 30 to 300 (if there is no warmup part) range. High value is recommended for the best precision. Iteration number for large messages may depend on I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD .

I_MPI_TUNING_AUTO_SYNC is highly recommended for tuning file store scenario.

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=10
- 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

Intel MPI library tuning approaches

	mpitune	mpitune_fast tuner	autotuner
Micro benchmark tuning			
Application tuning			
Easy of use			
Cluster time			
Adoption to environment			

mpitune_fast

- Starting with IMPI 2019 U7
- Target: system wide tuning
- Shell script that runs IMB under the hood
- Similar to autotuner with IMB target but automatically adepts #IMB iterations to the required minimum

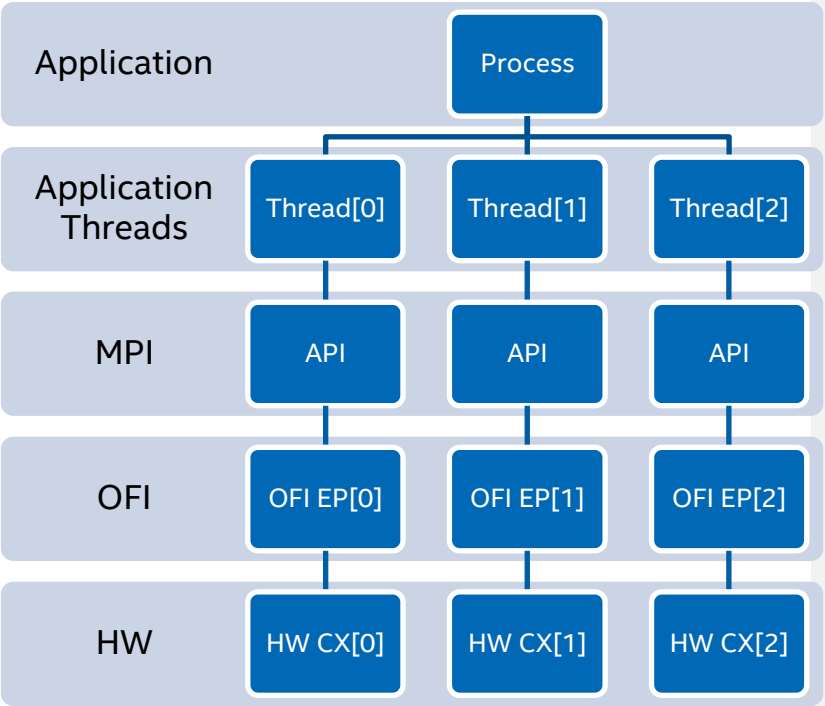
```
$ mpitune_fast -h

POSIX compatible Intel MPI autotunung script
Usage:
  mpitune_fast [OPTIONS]
Options:
  -p|--impi-path
      Set directory with installed IMPI package.
      You can source IMPI environment scripts instead or set I_MPI_ROOT variable.
  -d|--results_dir
      Set custom directory for tuning results, host files and logs.
      Default: current working directory.
  -f|--hostfile
      Set host file path.
      One host name per line. You can set I_MPI_HYDRA_HOST_FILE variable instead.
      If you are using SLURM/LSF cluster manager script should detect allocated hosts by itself.
  -o|--origin
      Set origin tuning file to merge it with new tuning results.
      Origin tuning file will not be changed.
  -ppn|--ppn
      Set custom process per node count list to tune delimited by commas.
      Example: 1,8,16
      Default: all powers of two up to the physical core count inluding physical core count.
  -c|--colls
      Set custom collective operations to tune delimited by commas.
      Example: allreduce,reduce,allgather
      Default: allreduce,reduce,gather,scatter,bcast,barrier
```

Multiple Endpoints/Asynchronous Progress

Enhanced support for Hybrid Programming Models

- **New** MPI_THREAD_MULTIPLE model extension
  - Available with release_mt library version: I_MPI_THREAD_SPLIT=1
- **New** asynchronous progress engine design

- **Note:** not available for OFI/mlx provider (WIP. Only OFI/verbs is available for Mellanox)



OFI EP - OFI endpoint

HW CX - Independent HW context

Multiple Endpoints based features in intel® MPI library 2019

- **Thread-split**
  - **Decrease threading computation imbalance** - communicate as soon as data is ready, don't wait for the slowest thread
  - **Improve interconnect saturation** from single MPI rank (Intel® Omni Path Fabric, InfiniBand and Ethernet are supported)
  - **Avoid implied bulk synchronization** threading barriers and overhead on parallel sections start/stop
- Asynchronous progress threads
  - **Offload communication** from application threads to MPI progress threads
  - **Improve computation/communication** overlap
  - **Parallelize communication** by multiple MPI progress threads

Both features are available only for:

- Linux
- release_mt (non default version)

THREAD-SPLIT – STRONG SCALING CODE MODIFICATIONS

```c
#define N 2


int main() {


    int i;
    int buffer[N];


    MPI_Init(NULL, NULL);


 #pragma omp parallel for num_threads(N)
  for (i = 0; i < N; i++)
  {

     // threaded partial computation
     // i-th thread contributes to buffer[i]


  }

  // single-threaded global communication
  MPI_Allreduce(buffer, buffer, N, MPI_INT,
                              MPI_SUM, MPI_COMM_WORLD);

  MPI_Finalize();

  return 0;


}
```

```c
#define N 2


int main() {


    int i, provided;
    int buffer[N];


    MPI_Comm comms[N];

    MPI_Init_thread(NULL, NULL, MPI_THREAD_MULTIPLE, &provided);


    for (i = 0; i < N; i++)
        MPI_Comm_dup(MPI_COMM_WORLD, &comms[i]);


 #pragma omp parallel for num_threads(N)
  for (i = 0; i < N; i++)
  {
     // threaded partial computation
     // i-th thread contributes to buffer[i]


     // threaded partial communication inside parallel region

     MPI_Allreduce(&buffer[i], &buffer[i], 1, MPI_INT,
                                      MPI_SUM, comms[i]);

   }


   MPI_Finalize();

   return 0;
}
```

How to achieve an overlap Compute & Comm

- The MPI Standard does not guarantee asynchronous communication for non-blocking operations
- A helper thread is needed to asynchronously progress a message while the (main) application thread is doing computations
- Some fabrics support the offloading of specific operations
- Async. progress support in IMPI exists for
  - P2P operations & blocking collectives
  - partial support for non-blocking collectives (MPI_Ibcast, MPI_Ireduce, and MPI_Iallreduce).

How to achieve an overlap Compute & Comm

- A helper- thread might imbalance HPC workloads and is therefore switched off by default (I_MPI_ASYNC_PROGRESS)
- Helper threads currently requires the use of the Multiple Endpoint implementation library (release_mt) – currently (2019U8) no SHM transport
  - source ../mpivars release_mt/debug_mt
  - note that release_mt is not the default library
  - default release is also thread safe – supporting all 4 MPI thread levels

How to achieve an overlap Compute & Comm

- Progress threads can be pinned via I_MPI_ASYNC_PROGRESS_PIN=<list>
- Exclude resourced used by progress threads for regular IMPI ranks via I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=<list>
- The #of threads per rank can be controlled via I_MPI_ASYNC_PROGRESS_THREADS
- If multiple helper threads per rank are used, the multi ep feature on a communicator basis must be used –> requires code changes

How to achieve an overlap Compute & Comm

- For hybrid MPI jobs (e.g. + OpenMP)
  - number of ranks -> helper threads -> (logical) cores
  - helper thread resources (cores) would be relatively low

- For pure (homogeneous) MPI jobs
  - helper thread resources would be relatively large
  - workaround of pinning multiple helper threads from different ranks on the same (shared) resources

How to achieve an overlap Compute & Comm

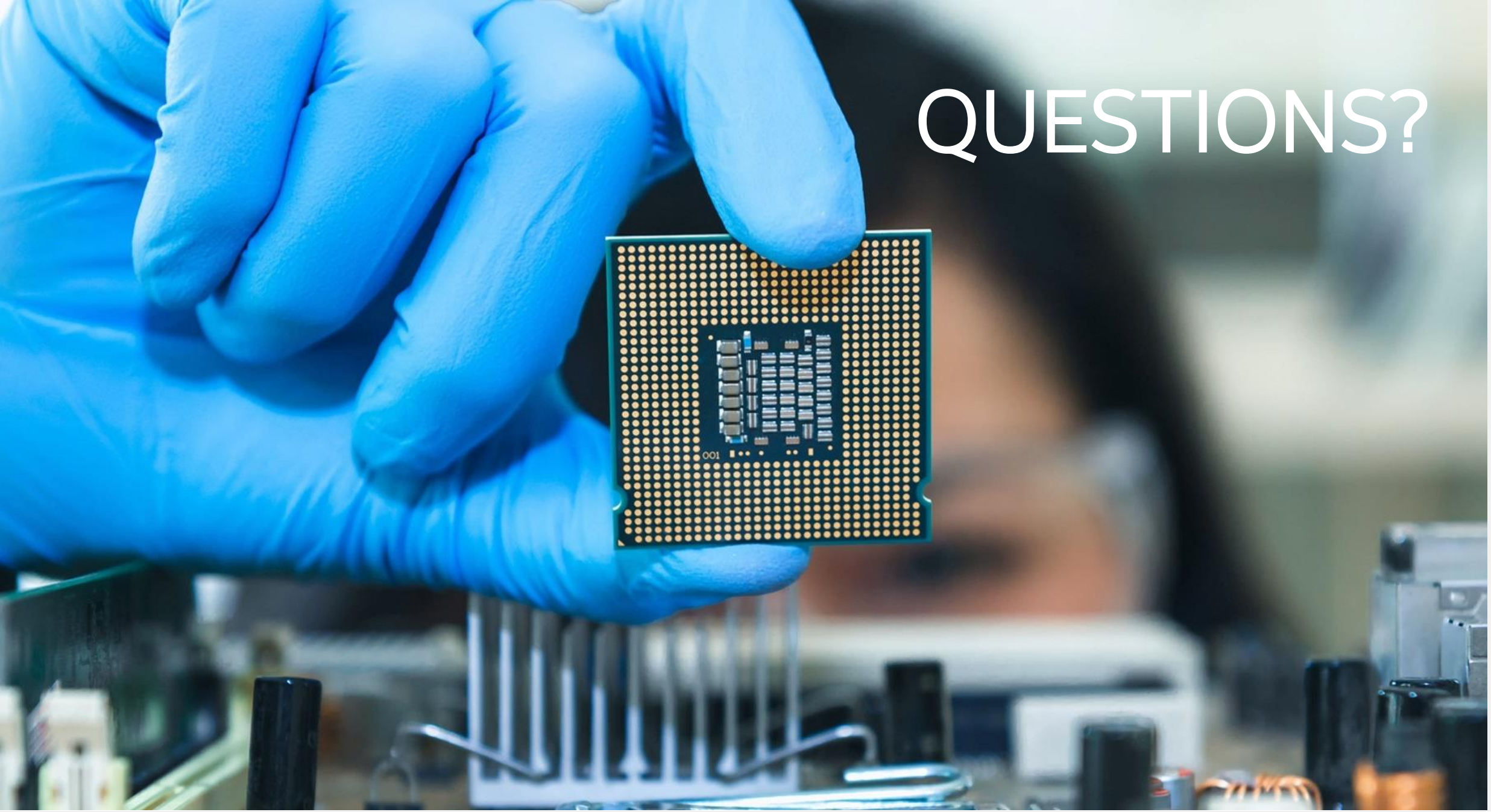
```
$ source .../mpi/intel64/bin/mpivars.sh release_mt
$ export I_MPI_ASYNC_PROGRESS=1
$ export I_MPI_PIN_PROCESSOR_LIST / I_MPI_PIN_DOMAIN ...
$ export I_MPI_ASYNC_PROGRESS_PIN=...
```

Documentation

- Developer Guide
- https://software.intel.com/en-us/mpi-developer-guide-linux-multiple-endpoints-support
- https://software.intel.com/en-us/mpi-developer-guide-linux-asynchronous-progress-control

- Developer Reference
- https://software.intel.com/en-us/mpi-developer-reference-linux-environment-variables-for-multi-ep
- https://software.intel.com/en-us/mpi-developer-reference-linux-environment-variables-for-asynchronous-progress-control

- Code examples
- $I_MPI_ROOT/doc/examples
- https://software.intel.com/en-us/mpi-developer-guide-linux-code-examples


QUESTIONS?

intel®

Backup

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)

Limitations of the Intel MPI mlx provider

IMPI 2019U5 puts UCX into DC transport mode, while InfiniBand* hardware older than EDR doesn't support DC.

Check support via
$ ucx_info -d | grep Transports
The output should include dc, rc, and ud transports.

As a workaround, select RC / UD manually e.g.
$ FI_MLX_TLS=UD

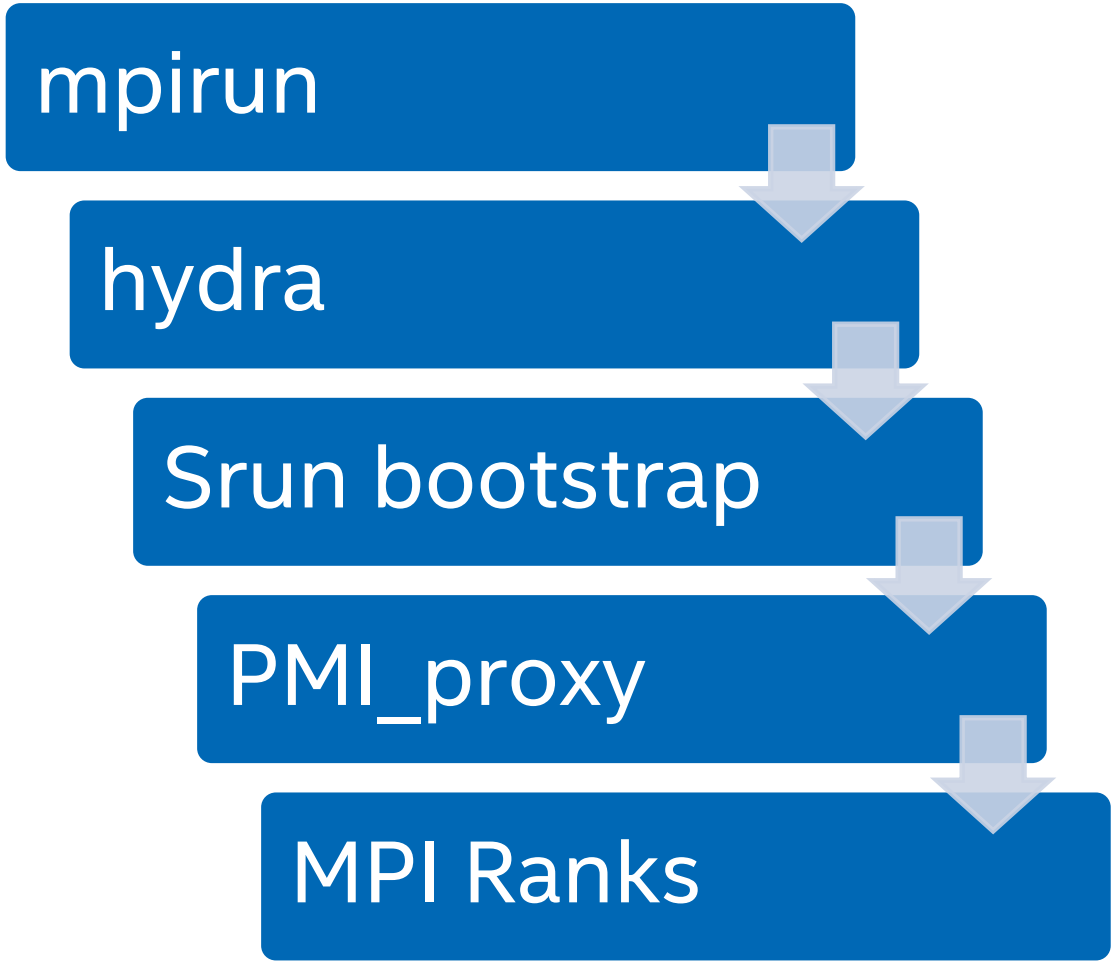
If none of the required transports are present, recheck your UCX configuration:
$ ibv_devinfo
$ lspci | grep Mellanox

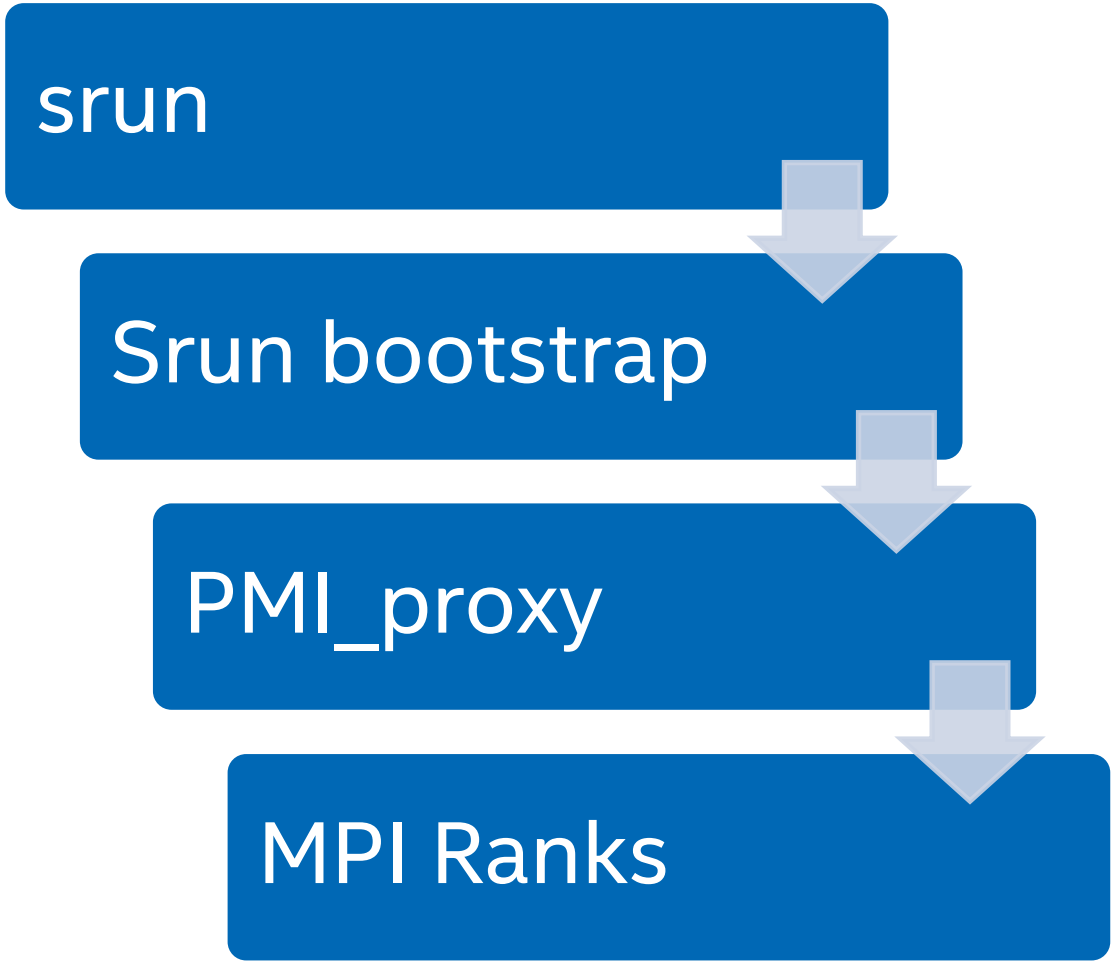
AWS Elastic Fabric Adapter (EFA) Support

- Starting with LibFabric 1.9.0 and IMPI 2019 U6
- EFA usage by IMPI can be confirmed using I_MPI_DEBUG
- OS-bypass using the Elastic Network Adapter (ENA) on Linux instances
- AWS supported Instances are
- c5n.18xlarge
- c5n.metal

SLURM Integration

SLURM Process Manager Integration





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

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.

Attaching Tools to MPI Ranks under SLURM

Intel MPI launcher

```
$ mpirun –n 512 –gtool “amplxe-cl -c hpc-performance –r r_hpc_imb:99“ IMB-MPI1 -
npmin 512 AllGather
```

SLURM launcher

```
$ cat << EOF > ./multiprog.conf
0-98          IMB-MPI1 -npmin 512 AllGather
99        amplxe-cl -c hpc-performance –r r_hpc_imb -- IMB-MPI1 -npmin 512 AllGather
100-511       IMB-MPI1 -npmin 512 AllGather
EOF

$ srun --multi-prog ./multiprog.conf
```

Distributed Asynchronous Object Storage (DAOS) Support

DAOS support – new in Intel MPI 2019

- Next generation file system support
- MPI IO primitives optimization
- I_MPI_EXTRA_FILESYSTEM_FORCE=daos

Intel MPI Library - Will be the first commercial MPI Library Supporting DAOS