NEXT GENERATION INTEL MPI PRODUCT FOR NEXT GENERATION SYSTEMS:

INTEL® MPI LIBRARY 2019 FEATURES AND OPTIMIZATION TECHNIQUES

DMITRY DURNOV (INTEL)
KLAUS-DIETER OERTEL (INTEL)

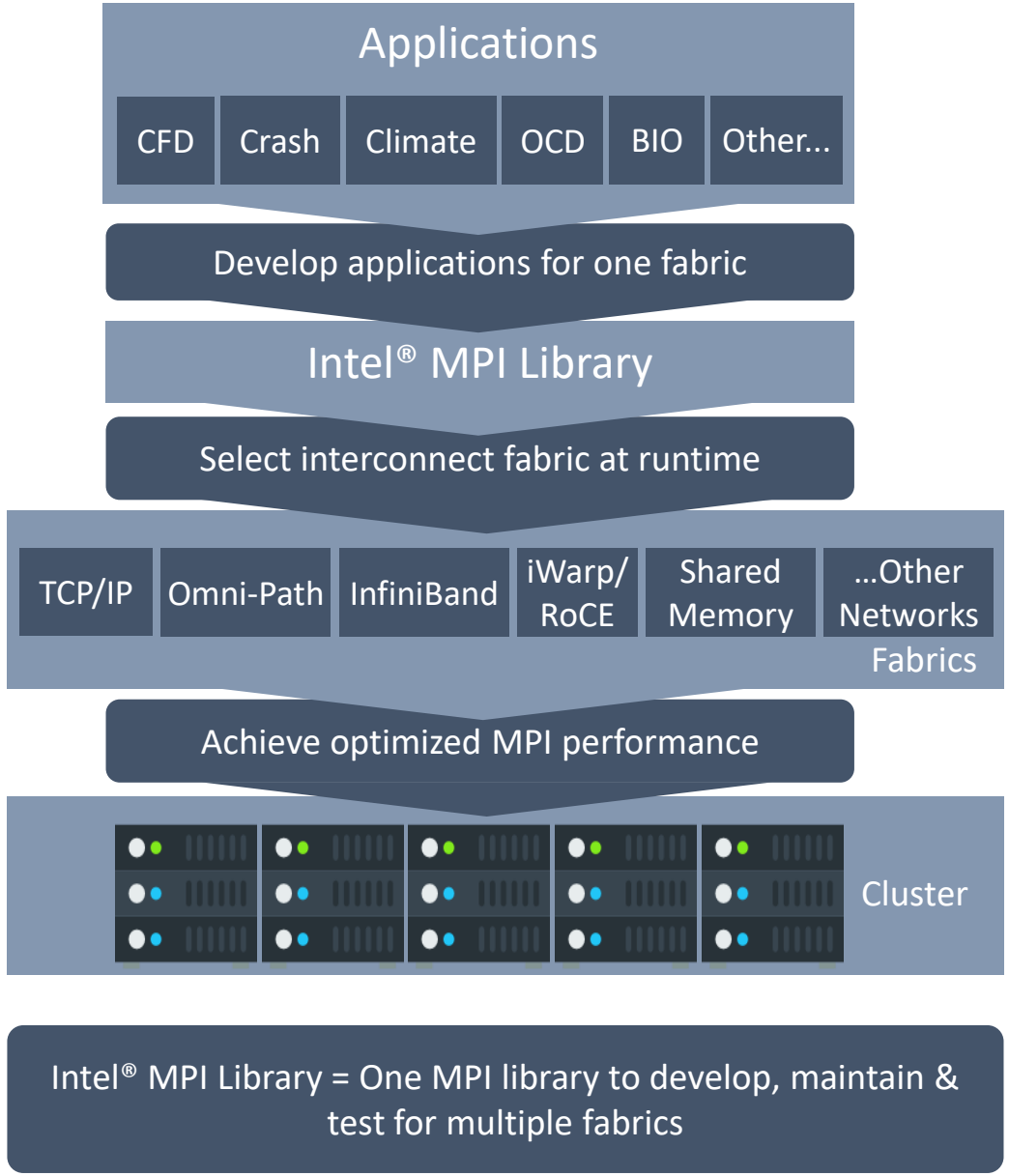
IXPUG ANNUAL CONFERENCE 2019, CERN

INTEL® MPI LIBRARY

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.

Learn More - software.intel.com/intel-mpi-library



Intel® MPI Library = One MPI library to develop, maintain & test for multiple 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 and Amazon AWS 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


intel
ARCHITECTURE

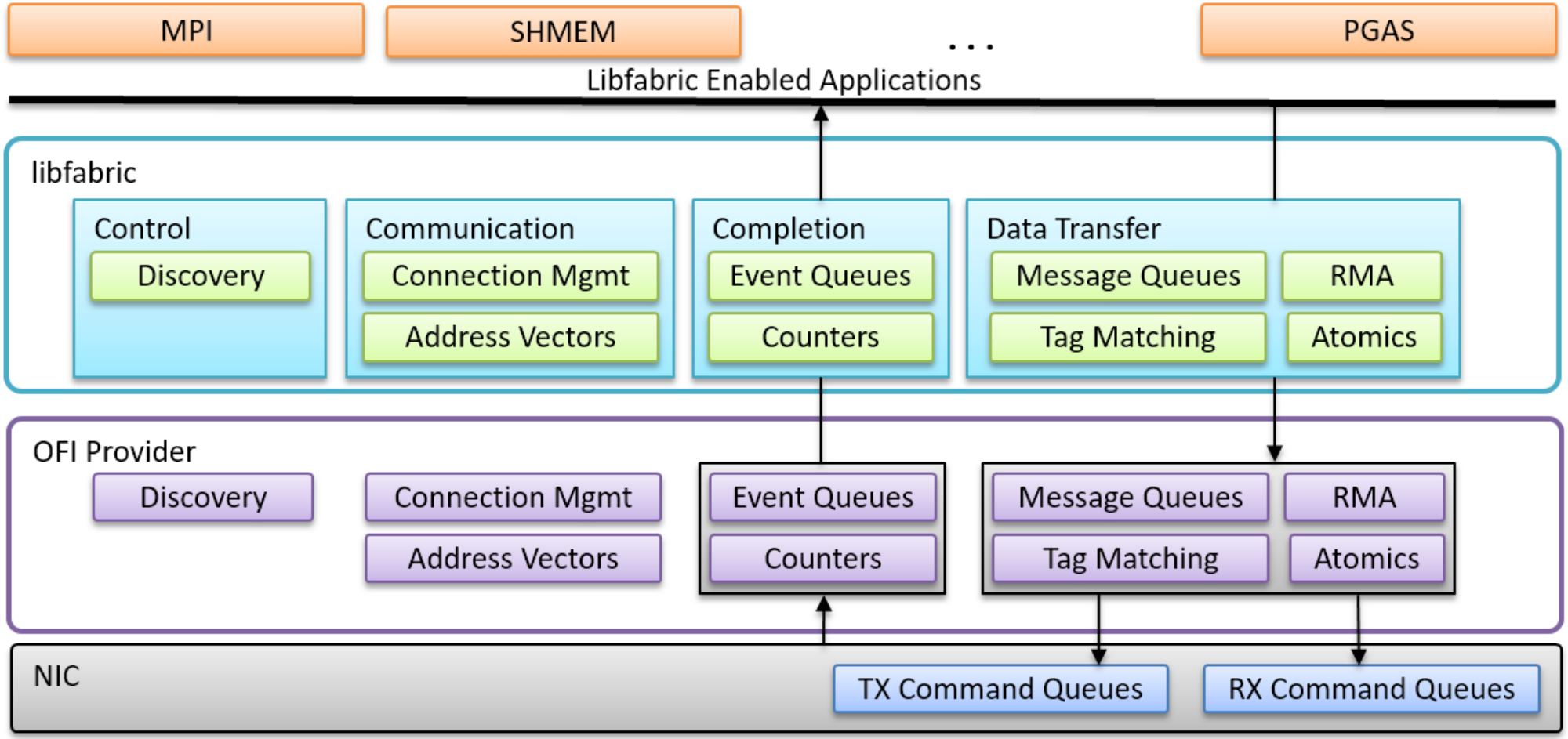
MPICH/CH4/OFI ARCHITECTURE IMPROVEMENTS

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

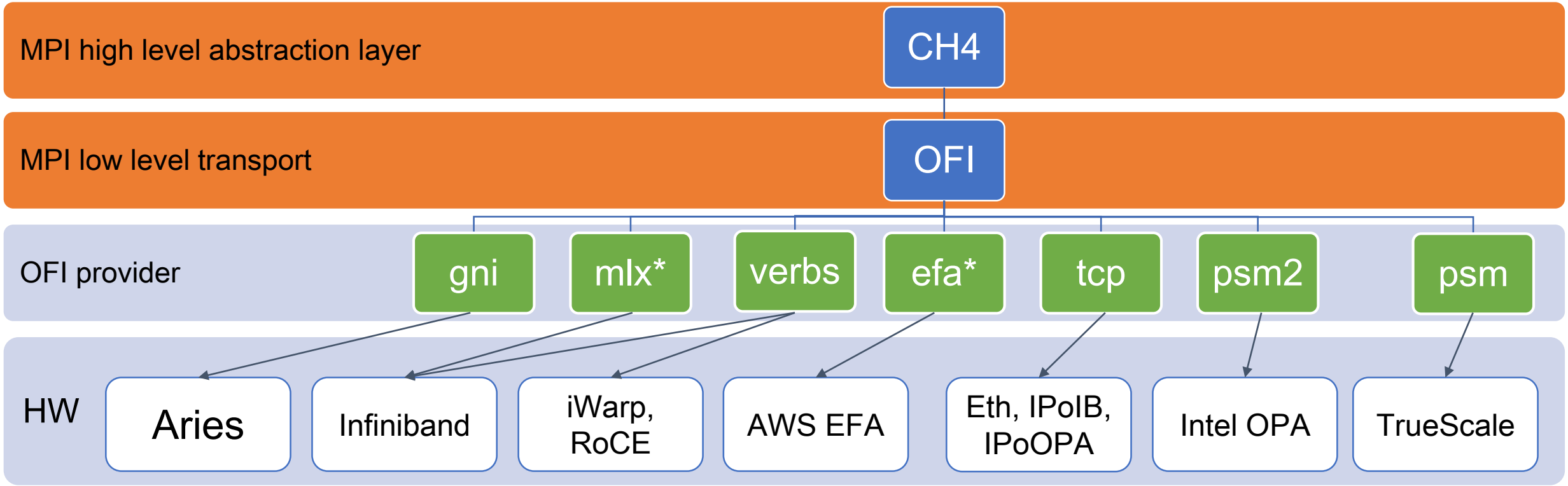
LIBFABRIC/OPEN FABRIC INTERFACES (OFI) STACK

OFI community
http://libfabric.org/



INTEL® MPI LIBRARY 2019 SW STACK/ECOSYSTEM

OFI community
http://libfabric.org/

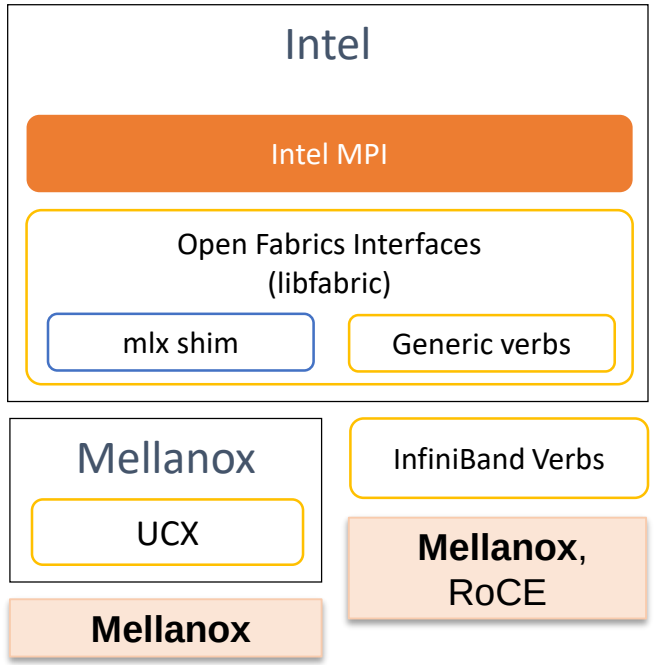


IMPI 2019 U5 is shipped with prebuilt libfabric 1.7.2 (psm2, verbs, mlx and tcp providers)

* - work in progress


MELLANOX SUPPORT IMPROVEMENT

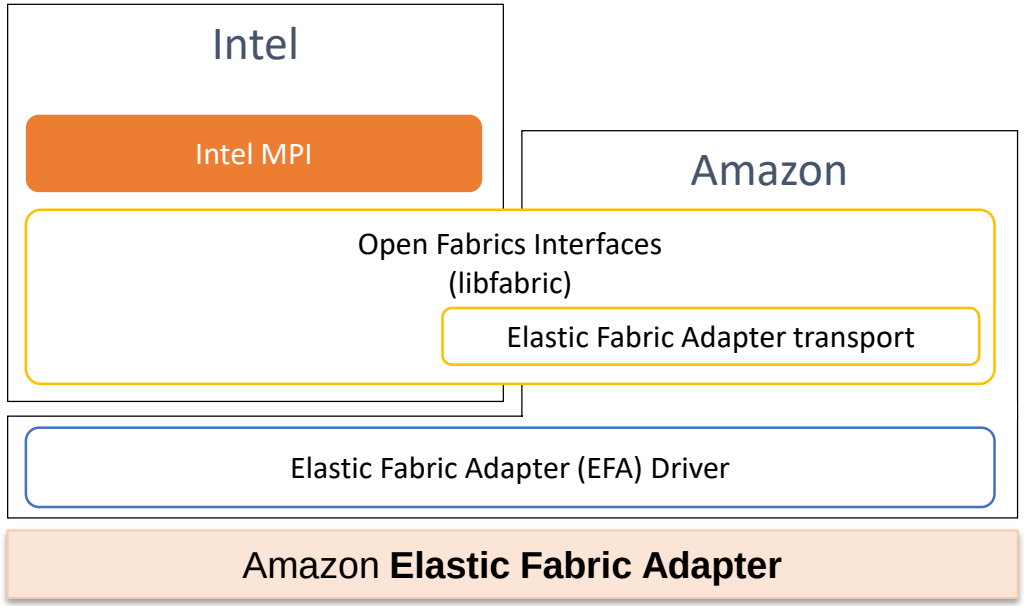
MELLANOX SUPPORT IMPROVEMENT



- **New** OFI/mlx provider (technical preview)
  - The provider is part of IMPI 2019 U5 distribution
  - Available via **FI_PROVIDER=mlx**
  - Validated with Mellanox EDR/HDR
  - Requires UCX 1.5+


intel
AMAZON AWS/EFA SUPPORT

AMAZON AWS/EFA SUPPORT



- **New** OFI/efa provider
  - Part of AWS environment
  - The provider is going to be shipped as part of IMPI distribution in the next IMPI release.
- Intel MPI 2019 U5 **OOB tuning** for OFI/efa

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/efa-start.html


AUTOTUNER

USABILITY FEATURES

- **New** tuning data is separated from the library:
  https://software.intel.com/en-us/node/807848
- **New** auto tuning capability (**autotuner**)
- **New** spell checker logic
- **New** impi_info tool (MPI_T based)

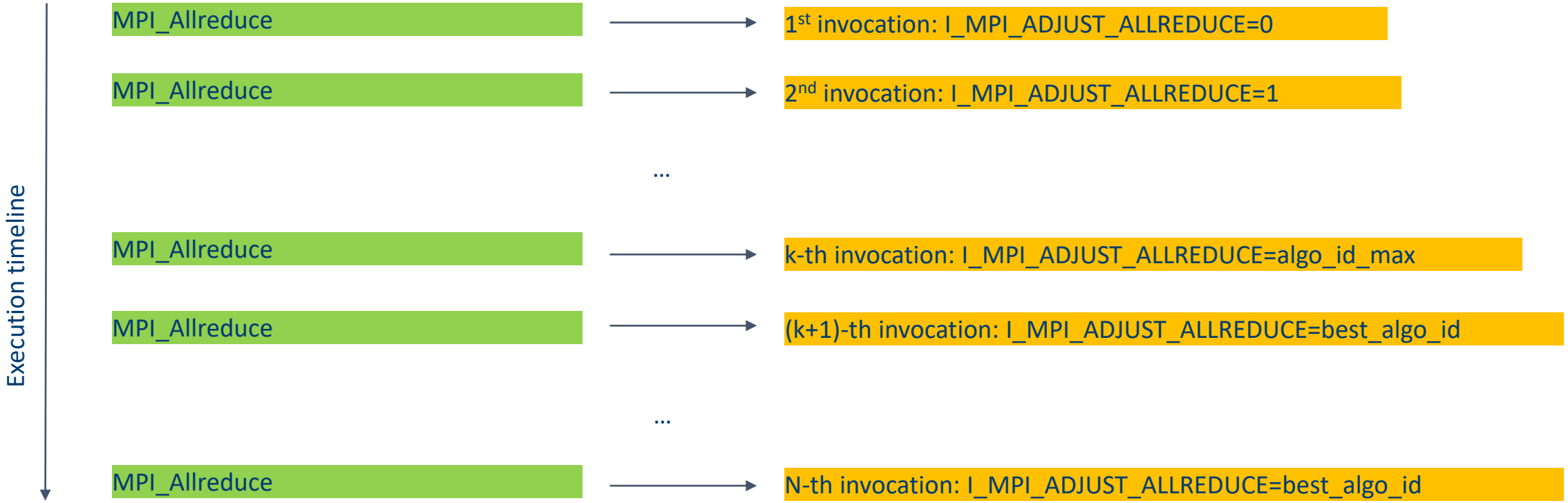
```
$ I_MPI_PIN_DMAIN=socket mpirun -hosts host01,host02 -n 2 -ppn 1
IMB-MPI1 barrier
[0] MPI startup(): I_MPI_PIN_DMAIN environment variable is not
supported.
[0] MPI startup(): Similar variables:
        I_MPI_PIN_UNIT
        I_MPI_PIN
        I_MPI_PIN_DOMAIN
[0] MPI startup(): To check the list of supported variables, use
the impi_info utility or refer to https://software.intel.com/en-
us/mpi-library/documentation/get-started.
```

```
$ impi_info | head -10
 | NAME                                  | DEFAULT VALUE  | DATA TYPE |
 ==========================================================================
 |I_MPI_PIN                              | on             | MPI_CHAR  |
 |I_MPI_PIN_SHOW_REAL_MASK               | on             | MPI_INT   |
 |I_MPI_PIN_PROCESSOR_LIST               | not defined    | MPI_CHAR  |
 |I_MPI_PIN_PROCESSOR_EXCLUDE_LIST       | not defined    | MPI_CHAR  |
 |I_MPI_PIN_CELL                         | unit           | MPI_CHAR  |
 |I_MPI_PIN_RESPECT_CPUSET               | on             | MPI_CHAR  |
 |I_MPI_PIN_RESPECT_HCA                  | on             | MPI_CHAR  |
 |I_MPI_PIN_DOMAIN                       | auto:compact   | MPI_CHAR  |
```

INTEL® MPI LIBRARY TUNING APPROACHES

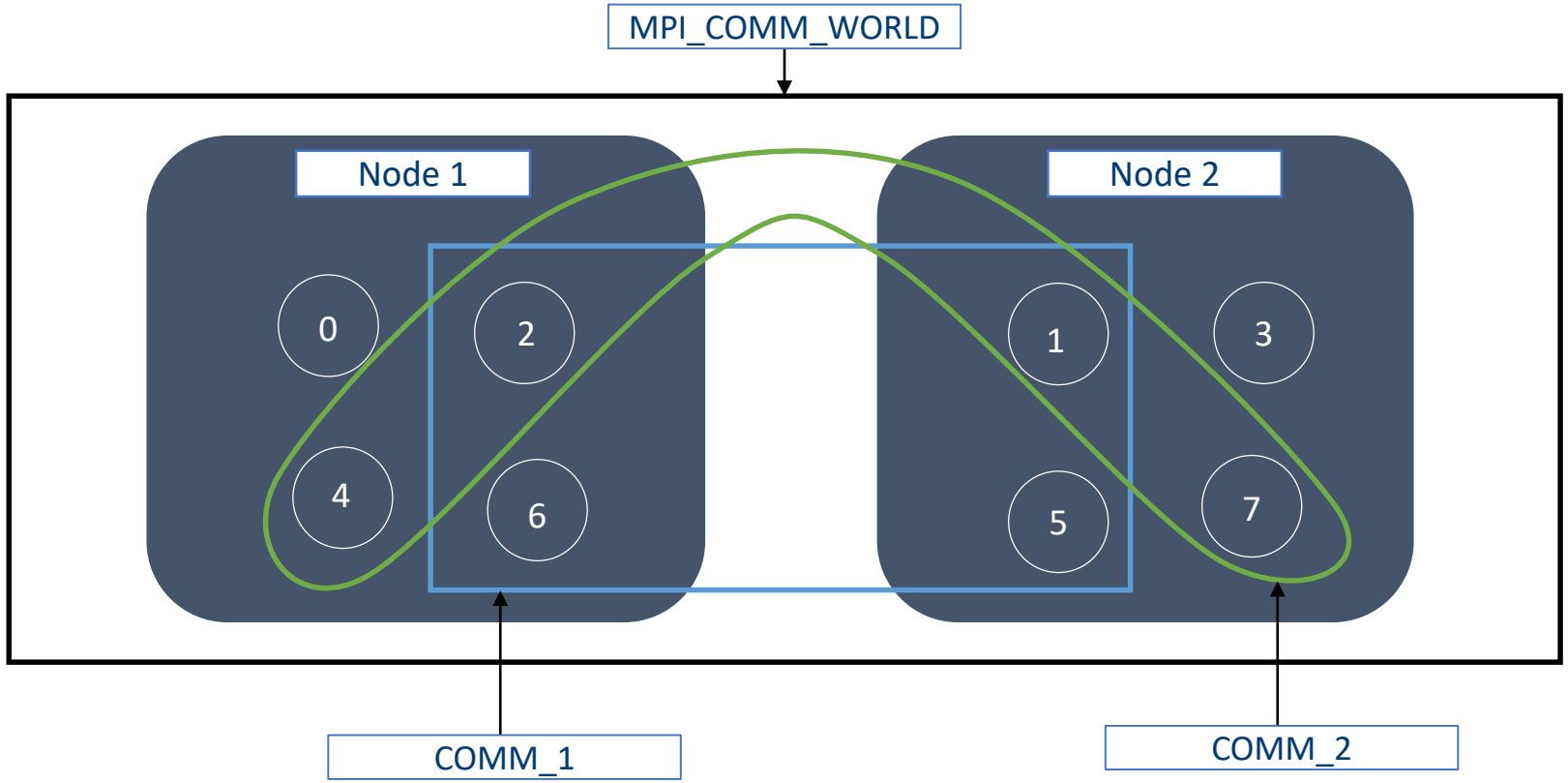
	mpitune	mpitune/ fast tuner	autotuner
Micro benchmark tuning			
Application tuning			
Easy of use			
Cluster time			
Adoption to environment			

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

AUTOTUNER COMMUNICATOR SPECIFIC TUNING



Each communicator has its own tuning. (E.g. COMM_1 and COMM_2 have independent tuning)

GET STARTED WITH AUTOTUNER

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
```

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|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)

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.

ENVIRONMENT VARIABLES. TUNING SCOPE AND STORAGE CONTROL

I_MPI_TUNING_AUTO_COMM_LIST=<comm_id_1, … , comm_id_k> List of communicators to be tuned (all communicators by default)

I_MPI_TUNING_AUTO_COMM_USER=<0|1> Enable user defined comm_id through MPI_Info object. (**disabled** by default)

I_MPI_TUNING_AUTO_COMM_DEFAULT=<0|1> Default/universal comm_ids. (**disabled** by default)

I_MPI_TUNING_AUTO_STORAGE_SIZE=<size> Max per-communicator tuning storage size (**512KB** by default)

NOTE: You may use Intel® VTune™ Amplifier's Application Performance Snapshot for per communicator MPI cost analysis and narrow tuning scope.
I_MPI_TUNING_AUTO_COMM_DEFAULT disables comm_id check (allows to get universal tuning)

INTEL® VTUNE™ AMPLIFIER'S APPLICATION PERFORMANCE SNAPSHOT (APS) PER COMMUNICATOR ANALYSIS

1. Source apsvars.sh:

```
$ source <path_to_aps>/apsvars.sh
```

2. Gather APS statistics:

```
$ export MPS_STAT_LEVEL=5
$ export APS_COLLECT_COMM_IDS=1
$ mpirun -n 4 -ppn 2 aps IMB-MPI1 allreduce -iter 1000,800
```

3. Generate an APS report:

```
$ aps-report aps_result_20190228/ -lFE
```

https://software.intel.com/sites/products/snapshots/application-snapshot/

Available with Intel® VTune™ Amplifier's Application Performance Snapshot Update 4

INTEL® VTUNE™ AMPLIFIER'S APPLICATION PERFORMANCE SNAPSHOT (APS) PER COMMUNICATOR ANALYSIS

4. Get the results:

Communicators used in the application			
Communicator Id	Communicator Size	Time (Rank Average)(sec)	Ranks
4611686018431582688	4	**1.80** (0.45)	0,1,2,3
4611686018431582208	4	0.59 (0.15)	0,1,2,3
4611686018429485552	2	0.51 (0.25)	0,1
4611686018429485520	2	0.01 (0.00)	0,1
4611686018431582672	4	0.00 (0.00)	0,1,2,3

INTEL® VTUNE™ AMPLIFIER'S APPLICATION PERFORMANCE SNAPSHOT (APS) INTEROPERABILITY

5. Specify communicators to be tuned:

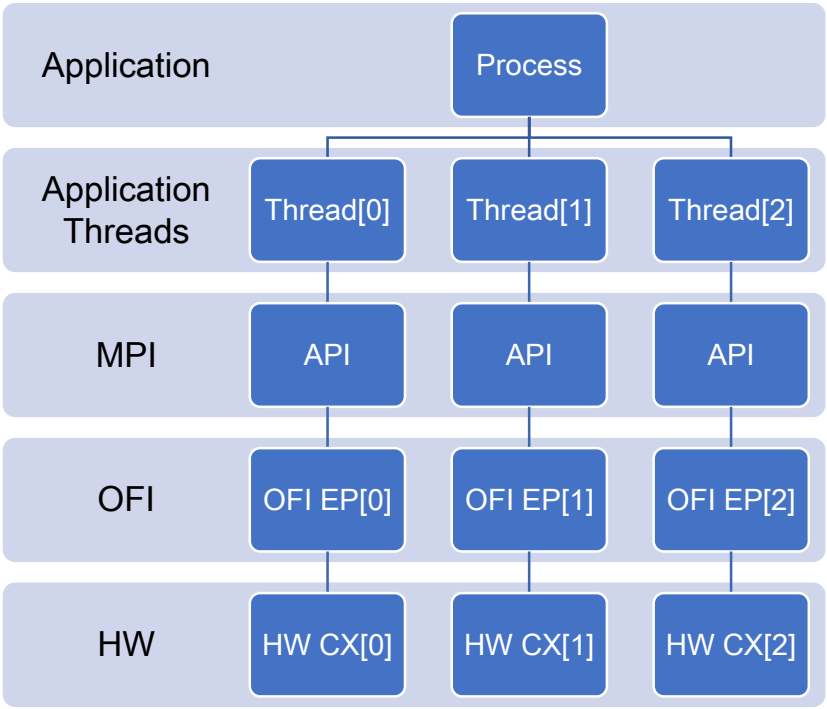
```
$ export I_MPI_TUNING_AUTO_COMM_LIST=4611686018431582688
$ export I_MPI_TUNING_MODE=auto
$ mpirun -n 96 -ppn 48 IMB-MPI1 allreduce -iter 1000,800 -time 4800
```

NOTE: I_MPI_TUNING_AUTO_ITER_POLICY may impact tuning cycle for large messages. Please check that you have enough application level invocations


intel
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



OFI EP - OFI endpoint
HW CX - Independent HW context

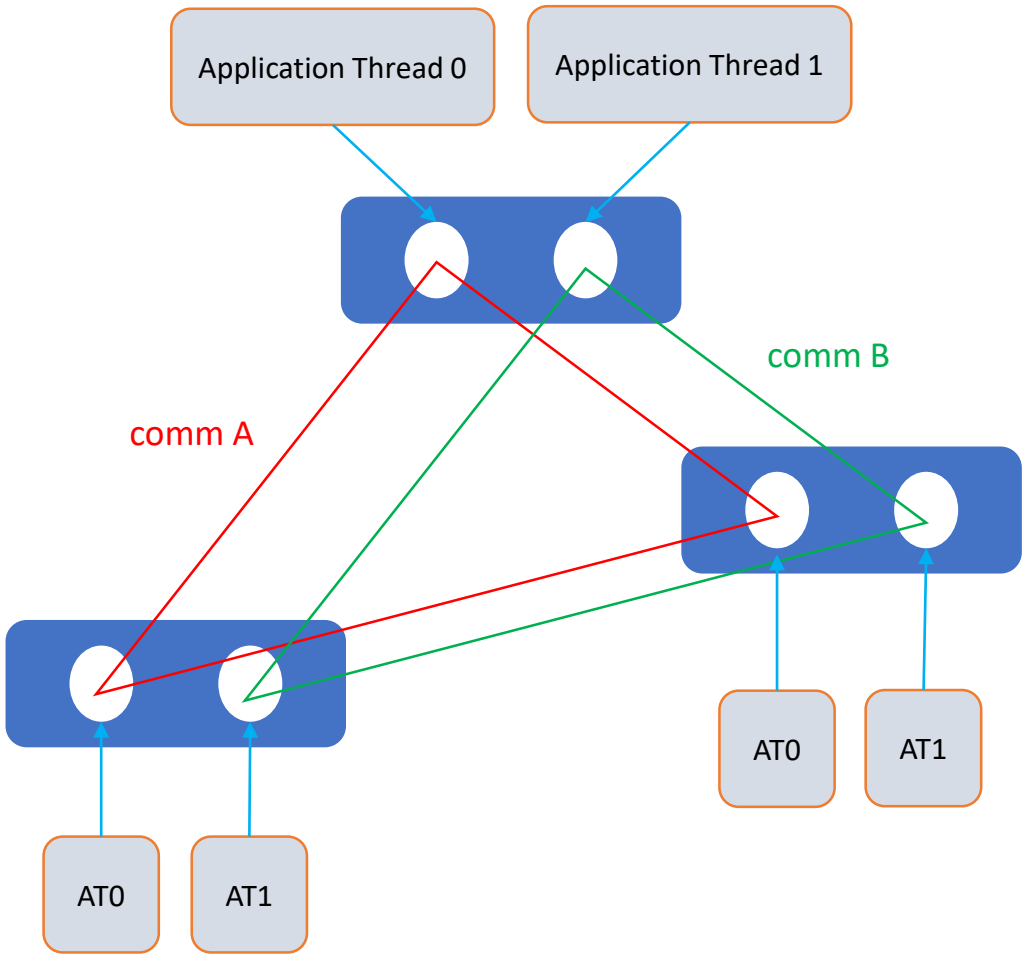
MULTIPLE ENDPOINTS BASED FEATURES IN INTEL® MPI LIBRARY 2019

- 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

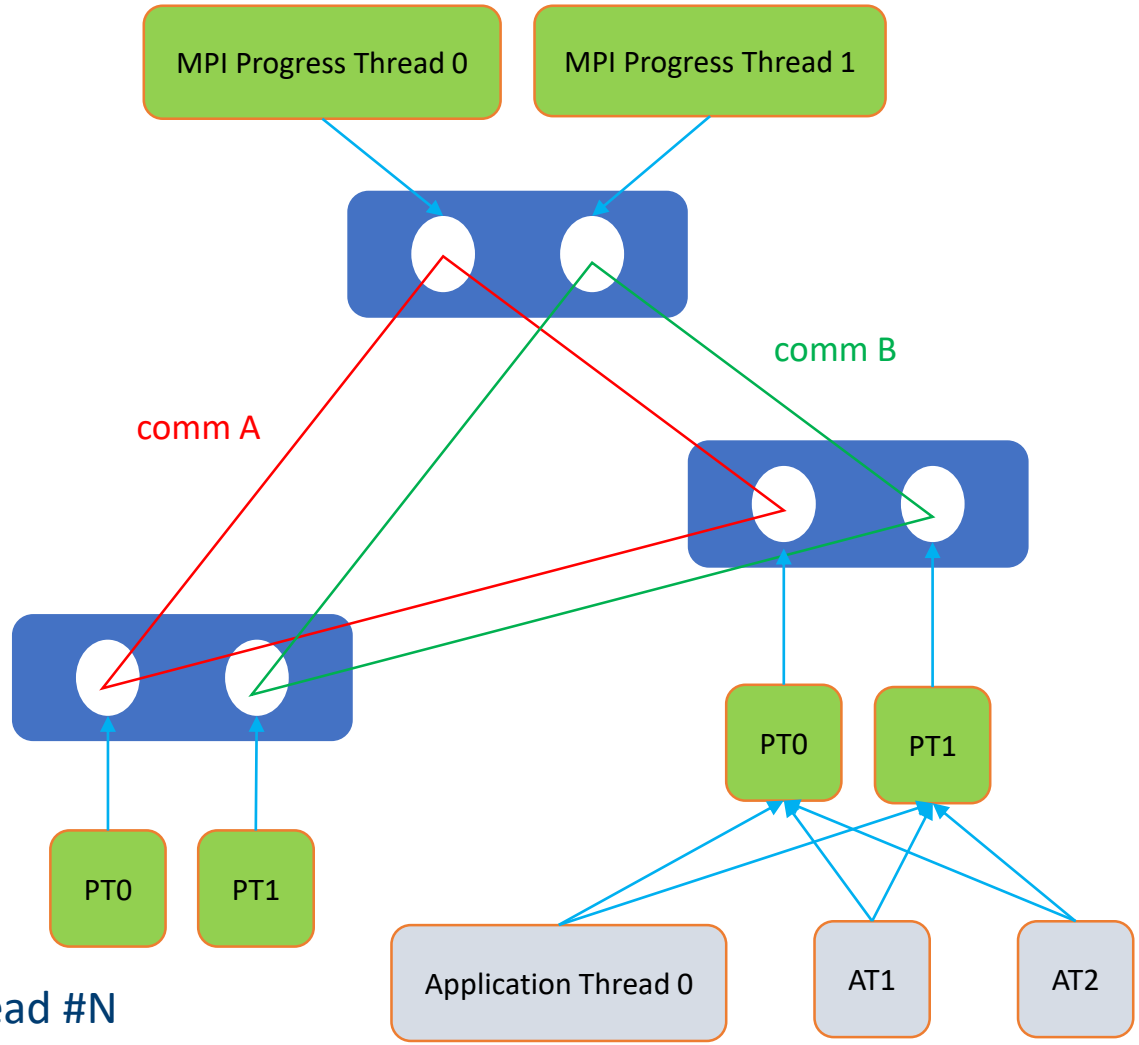
- Asynchronous progress threads
  - **Offload communication** from application threads to MPI progress threads
  - **Improve computation/communication** overlap
  - **Parallelize communication** by multiple MPI progress threads

Both features are available only for:
- Linux
- I_MPI_FABRICS=ofi
- release_mt (non default version)

Thread-split



Asynchronous progress threads

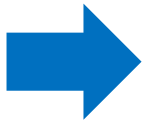


AT<N> - Application Thread #N

MPT<N> - MPI Progress Thread #N

THREAD-SPLIT – STRONG SCALING CODE MODIFICATIONS

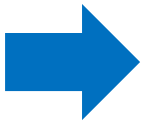
```
1. #define N 2
2.
3. int main() {
4.     int i;
5.     int buffer[N];
6.
7.
8.     MPI_Init(NULL, NULL);
9.
10.
11.
12.  #pragma omp parallel for num_threads(N)
13.     for (i = 0; i < N; i++) {
14.         // threaded partial computation
15.         // i-th thread contributes to buffer[i]
16.
17.
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.
17.
18.
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. }
```

ASYNCHRONOUS PROGRESS THREADS – STRONG SCALING CODE MODIFICATIONS

```
1. #define N        (4)
2. #define CHUNK_SZ (1024)
3.
4. int main()
5. {
6.     MPI_Request request;
7.     MPI_Status status;
8.     int sbuf[N*CHUNK_SZ], rbuf[N*CHUNK_SZ];
9.     int idx;
10.
11.    MPI_Init(NULL, NULL);
12.
13.
14.
15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.
26.
27.    MPI_Iallreduce(sbuf,
28.                   rbuf,
29.                   N * CHUNK_SZ, MPI_INT, MPI_SUM,
30.                   MPI_COMM_WORLD, &request);
31.
32.    MPI_Wait(request, status);
33.    MPI_Finalize();
34.    return 0;
35. }
```



```
1. #define N        (4)
2. #define CHUNK_SZ (1024)
3.
4. int main()
5. {
6.     MPI_Request requests[N];
7.     MPI_Status statuses[N];
8.     int sbuf[N*CHUNK_SZ], rbuf[N*CHUNK_SZ];
9.     int idx;
10.
11.    MPI_Init(NULL, NULL);
12.
13.    MPI_Info info;
14.    MPI_Comm comms[N];
15.    char thread_id_str[16];
16.    MPI_Info_create(&info);
17.
18.    for (idx = 0; idx < N; idx++) {
19.        MPI_Comm_dup(MPI_COMM_WORLD, &comms[idx]);
20.        sprintf(thread_id_str, "%d", idx);
21.        MPI_Info_set(info, "thread_id", thread_id_str);
22.        MPI_Comm_set_info(comms[idx], info);
23.    }
24.    MPI_Info_free(&info);
25.
26.    for (idx = 0; idx < N; idx++) {
27.        MPI_Iallreduce(send_buffer + idx * CHUNK_SZ,
28.                       recv_buffer + idx * CHUNK_SZ,
29.                       CHUNK_SZ, MPI_INT, MPI_SUM,
30.                       comms[idx], &requests[idx]);
31.    }
32.    MPI_Waitall(N, requests, statuses);
33.    MPI_Finalize();
34.    return 0;
35. }
```

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

THREAD-SPLIT. BIBAND

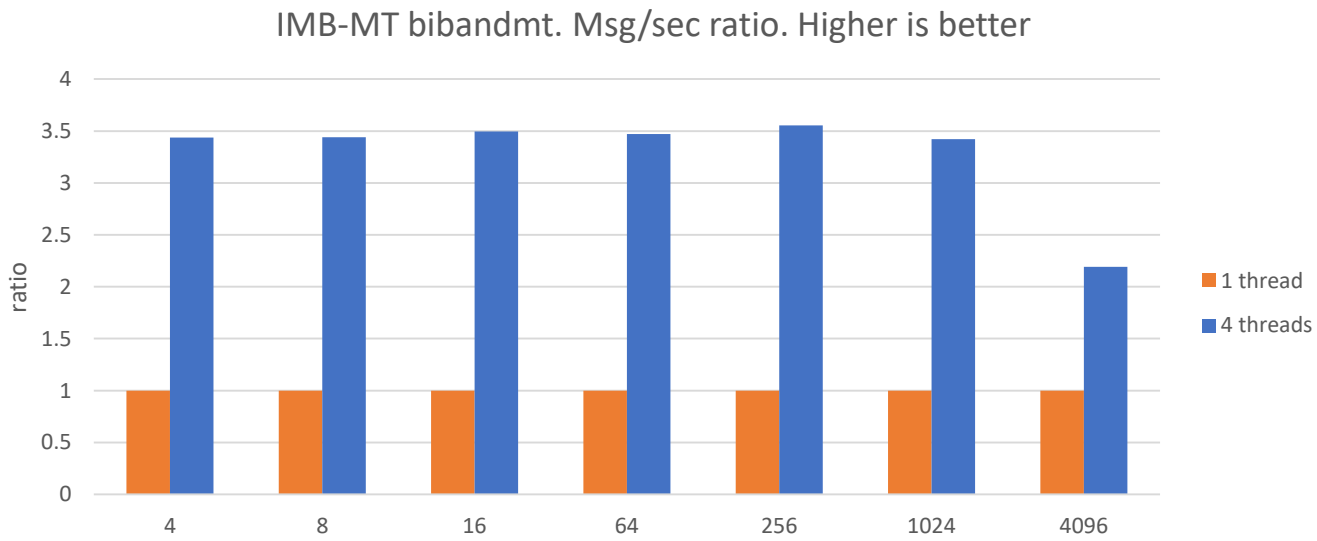
```
$ source <impi_2019_install_path>/intel64/bin/mpivars.sh release_mt
```

- 1 thread:

```
$ OMP_NUM_THREADS=1 OMP_PLACES=cores I_MPI_THREAD_SPLIT=1 I_MPI_THREAD_RUNTIME=openmp mpirun -n 2 -ppn 1 -hosts host1,host2 IMB-MT -thread_level multiple bibandmt -count 4,8,16,64,256,1024,4096 -repeat 10000 -datatype char -window_size 64
```

- 4 threads:

```
$ OMP_NUM_THREADS=4 … -window_size 16
```



Performance results are based on testing as of June 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure. 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. For more complete information visit http://www.intel.com/benchmarks.

Configuration:
Testing by Intel as of June, 2019. 2 nodes
Hardware: Intel® Xeon® Gold 6252 CPU @ 2.10GHz; 192 GB RAM. Intel® Turbo Boost Technology and Hyperthreading Technology enabled.
Interconnect: Intel® Omni-Path Host Fabric Interface
Software: RHEL* 7.6; IFS 10.9.0.0.210-957.12.2-2.10.6; Libfabric distributed with Intel® MPI 2019 Update 4; Intel® MPI Library 2019 Update 4

ASYNCHRONOUS PROGRESS THREADS. IALLREDUCE

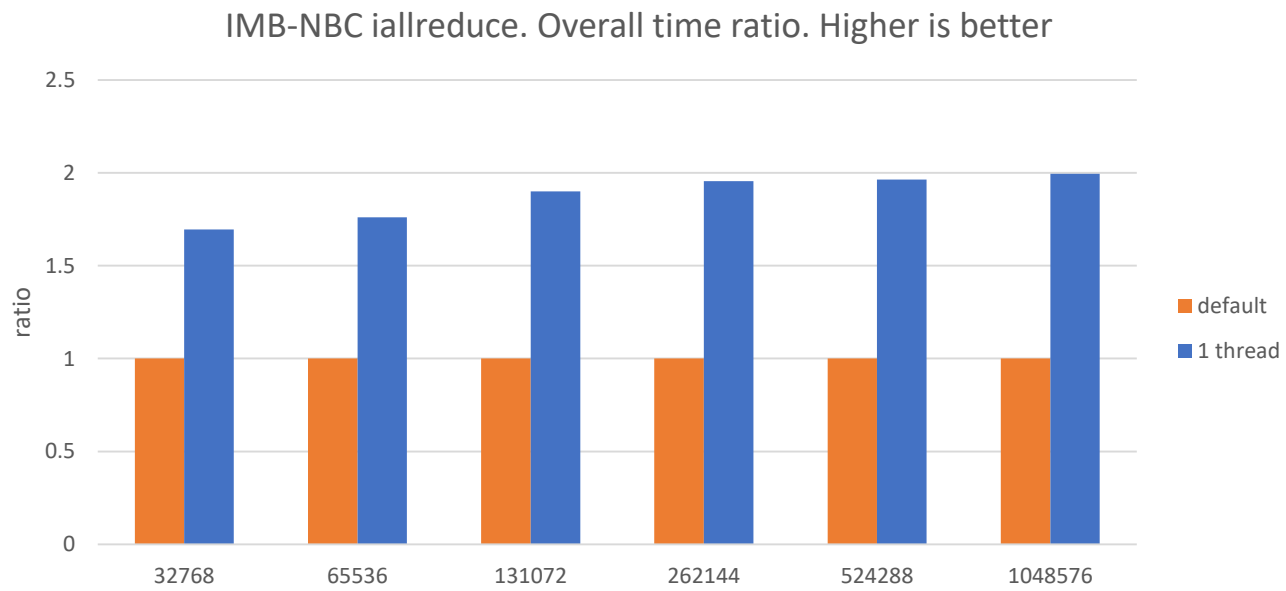
```
$ source  <impi_2019_install_path>/intel64/bin/mpivars.sh release_mt
```

- default

```
$ I_MPI_PIN_PROCESSOR_LIST=4 mpirun -n 2 -ppn 1 -hosts host1,host2 IMB-NBC -thread_level multiple
iallreduce -msglog 15:20
```

- 1 thread:

```
$ I_MPI_ASYNC_PROGRESS=1 I_MPI_ASYNC_PROGRESS_PIN=10 …
```



Configuration:
Testing by Intel as of June, 2019. 2 nodes
Hardware: Intel® Xeon® Gold 6252 CPU @ 2.10GHz; 192 GB RAM. Intel® Turbo Boost Technology and Hyperthreading Technology enabled.
Interconnect: Intel® Omni-Path Host Fabric Interface
Software: RHEL* 7.6; IFS 10.9.0.0.210-957.12.2-2.10.6; Libfabric distributed with Intel® MPI 2019 Update 4; Intel® MPI Library 2019 Update 4

USAGE IN APPLICATIONS

Thread-split:

- VASP (https://techdecoded.intel.io/resources/improving-vasp-materials-simulation-performance)
- GRID (https://github.com/paboyle/Grid/wiki/Dirac-ITT-Benchmarks)
- QCD (https://www.rdmag.com/news/2019/04/optimizing-network-software-advance-scientific-discovery)
- BQCD

Asynchronous progress threads:

- **Intel® Machine Learning Scaling Library** (https://github.com/intel/MLSL)
- GeoFEM

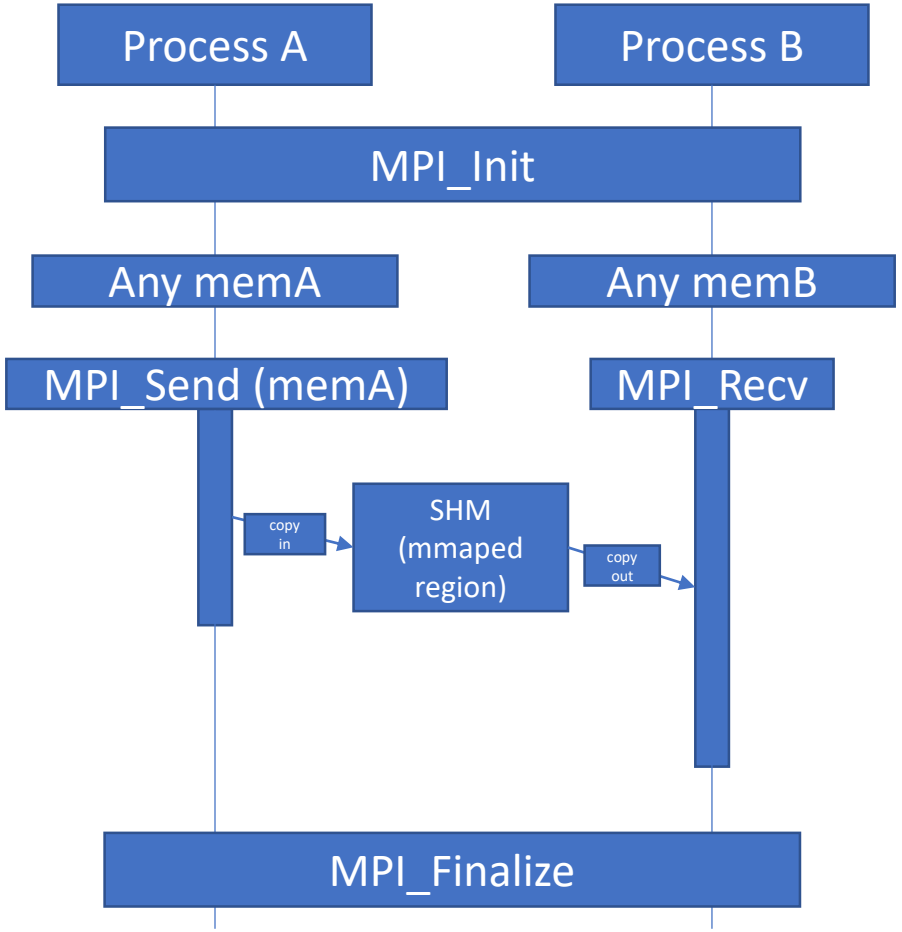

intel
SHM HEAP

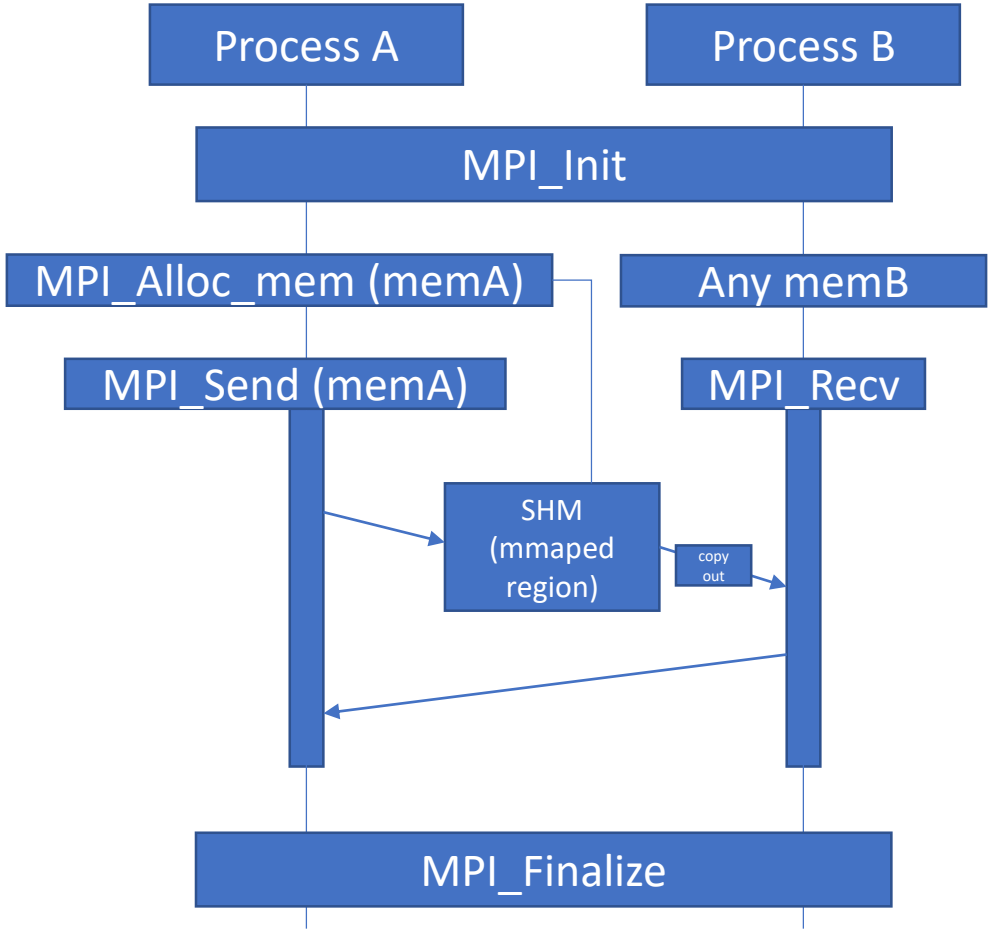
SHM HEAP INFRASTRUCTURE

- SHM pt2pt acceleration
  - **Removes copy-in** phase from pt2pt communication
  - **Efficient memory allocation** mechanism
    - Since kernel 4.7 we may get huge pages with shared pages as well

- **Multirail/multilead topology aware collective operations** (experimental)
  - Eliminates copy-in phase of topology aware collective operations
  - Allows ranks to share data w/o extra cost

SHM HEAP USE CASE





GET STARTED WITH SHM HEAP

Basic way 1 (application w/ MPI_Alloc_mem) – Enable SHM HEAP

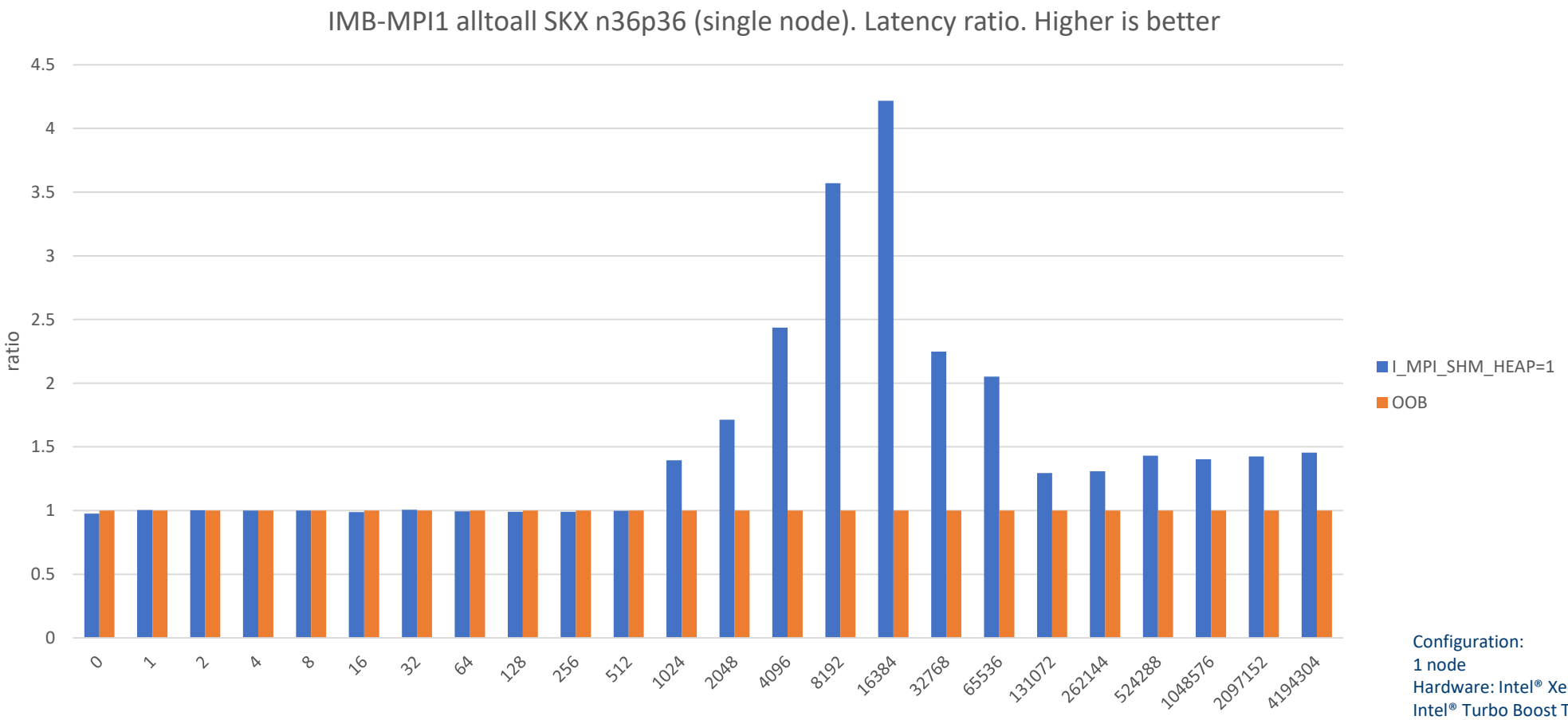
```
$ export I_MPI_SHM_HEAP=1
$ mpirun -n 36 -ppn 36 IMB-MPI1 alltoall -iter 1000,800 -time 4800
```

Basic way 2 (application w/o MPI_Alloc_mem) – Use proxy library to replace malloc with MPI_Alloc_mem:

```
$ export I_MPI_SHM_HEAP=1
$ export LD_PRELOAD=$I_MPI_ROOT/intel64/lib/libmpi_shm_heap_proxy.so
$ mpirun -n 36 -ppn 36 IMB-MPI1 alltoall -iter 1000,800 -time 4800
```

NOTE: IMB-MPI1 uses MPI_Alloc_mem for memory allocations

SHM HEAP IMPACT



Configuration:
1 node
Hardware: Intel® Xeon® Gold 6140 CPU @ 2.30GHz; 192 GB RAM. Intel® Turbo Boost Technology and Hyperthreading Technology enabled.
I_MPI_FABRICS=shm, Intel® MPI Library 2019 Update 5

SHM HEAP ENVIRONMENT VARIABLES.

I_MPI_SHM_HEAP=<0|1> SHM heap control. (**disabled** by default)

I_MPI_SHM_HEAP_VSIZE=<size in megabytes> SHM heap virtual memory pool size **per rank** (**4096 MB** by default)

I_MPI_SHM_HEAP_CSIZE=<size in megabytes> SHM heap cache size **per rank** (**12.5%** by default. **256 MB** – max size)

I_MPI_SHM_HEAP_OPT=<rank|numa> SHM heap optimizations/policy (**rank** by default)

Legal Disclaimer & Optimization Notice

Performance results are based on testing as of August and September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

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. For more complete information visit www.intel.com/benchmarks.

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


BACKUP

ENVIRONMENT VARIABLES. TUNING POLICY

I_MPI_TUNING_AUTO_ITER_POLICY=<0|1> Adaptive iterations number mode. (**enabled** by default)

I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD=<msg_size> Message size threshold for I_MPI_TUNING_AUTO_ITER_POLICY. (**64KB** by default).

I_MPI_TUNING_AUTO_POLICY=<max|min|avg> Autotuning strategy. (Use "**max**" time by default)

I_MPI_ADJUST_<opname>_LIST=<algid1>[-<algid2>][,<algid3>][,<algid4>-<algid5>] Collective operation algorithms filter list for autotuner. (All algorithms available by default)

NOTE: I_MPI_TUNING_AUTO_ITER_POLICY_THRESHOLD halves number of iterations. E.g. If I_MPI_TUNING_AUTO_ITER_NUM=256, then for 512KB message size number of iterations will be 32

THREAD-SPLIT. SENDRECVMT

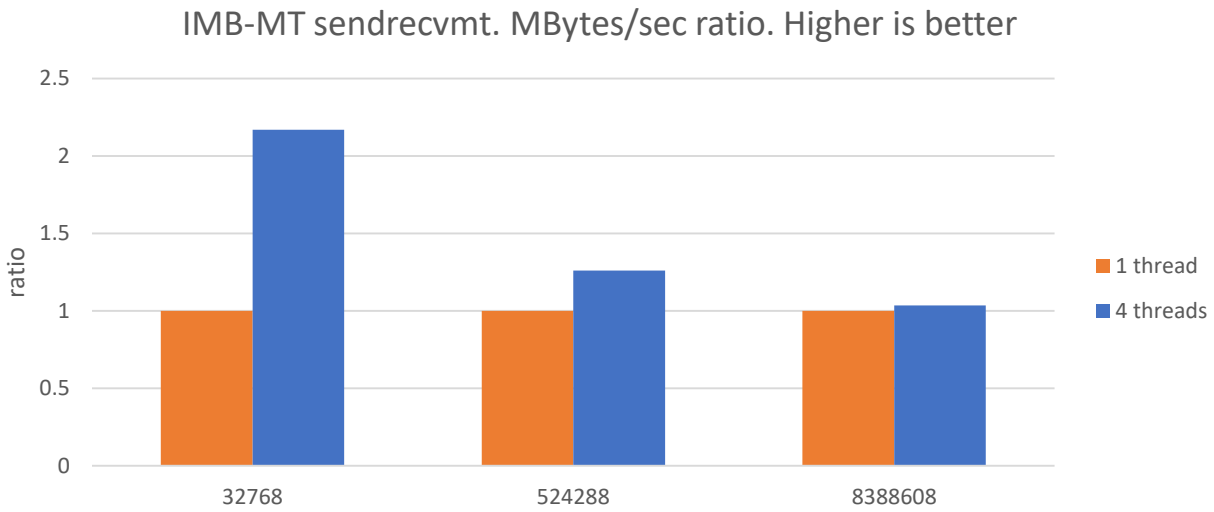
```
$ source  <impi_2019_install_path>/intel64/bin/mpivars.sh release_mt
```

- 1 thread:

```
$ OMP_NUM_THREADS=1 I_MPI_THREAD_SPLIT=1 I_MPI_THREAD_RUNTIME=openmp mpirun -n 2 -ppn 1 -hosts
host1,host2 IMB-MT -thread_level multiple -datatype char sendrecvmt -count 32768,524288,8388608 -
repeat 1000
```

- 4 threads:

```
$ OMP_NUM_THREADS=4 I_MPI_THREAD_SPLIT=1 I_MPI_THREAD_RUNTIME=openmp mpirun -n 2 -ppn 1 -hosts
host1,host2 IMB-MT -thread_level multiple -datatype char sendrecvmt -count 8192,131072,2097152 -
repeat 1000
```



Configuration:
Testing by Intel as of June, 2019. 2 nodes
Hardware: Intel® Xeon® Gold 6252 CPU @ 2.10GHz; 192 GB RAM. Intel® Turbo Boost Technology and Hyperthreading Technology enabled.
Interconnect: Intel® Omni-Path Host Fabric Interface
Software: RHEL* 7.6; IFS 10.9.0.0.210-957.12.2-2.10.6; Libfabric distributed with Intel® MPI 2019 Update 4; Intel® MPI Library 2019 Update 4


intel
experience
what's inside™