TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Best Practices for MPI Usage

Intel MPI

PRESENTED BY:
Amit Ruhela (aruhela@tacc.utexas.edu)

IXPUG 2020

October 15, 2020

Outline

- Compile and run your MPI program at TACC Clusters
- Profiling your MPI program
- Best practices and performance tuning for Intel MPI

Current MPI Stacks at TACC Clusters

1. Stampede2 :
    1. Intel MPI (IMPI) 18.0.2 with Intel 18.0.2; Intel MPI 17.0.3 with GCC 7.1.0; and Intel MPI 19.0.7 with Intel 19.1.1
    2. MVAPICH2 2.3.3 with Intel 18.0.2 (From The Ohio State University)
2. Frontera
    A. CPU
        1. Intel MPI 18.0.5, 19.0.5, and Intel MPI 19.0.7 (Intel 18.0.5, Intel 19.0.5, Intel 19.1.1, and GNU 9.1)
        2. MVAPICH2-X 2.3 (Intel 18.0.5, Intel 19.1.1 and GNU 8.3, GNU 9.1)
    B. GPU (RTX Queues)
        1. Intel MPI 18.0.5, 19.0.5, and Intel MPI 19.0.7 (Intel 18.0.5, Intel 19.0.5, Intel 19.1.1, and GNU 9.1)
        2. MVAPICH2-GDR 2.3 (Intel 18.0.5, Intel 19.0.5, and GNU 8.3)
3. Longhorn
    1. Spectrum MPI 10.3 with XL/16.1.1, GNU 9.1 and PGI 19.10 Compilers
    2. MVAPICH2-GDR 2.3.4 with XL/16.1.1 and GCC 7.3.0

Compile your MPI Programs on Clusters

- Use the (intel or gcc) + impi module on the system (or intel + MVAPICH2)

- Compile your program with appropriate wrappers
  - C Programs : mpicc / mpigcc / mpiicc
  - C++ Programs : mpicxx / mpicxx / mpigxx / mpiicpc
  - Fortran Programs : mpif77 / mpif90 / mpifc / mpiifort

- Architecture-specific flags

  include "-xCORE-AVX512 -qopt-zmm-usage=high" as a build option

- Specify an optimization flag “-O3” for highest performance

- Do not use the “-fast” or “-static” flag for the Intel compiler.

- Use “mpicc -show” to see the full compiler invocation line

Run with ibrun (Both IMPI and MVAPICH2)

- TACC's universal MPI job starter on all the systems
- Supports multiple MPI stacks
  - Intel MPI: mpiexec.hydra
  - MVAPICH2: mpirun_rsh
  - OpenMPI: mpirun
- Sets up the environment needed for network (hostfile, core count, etc.)
- Provides a set of default options for good performance
- Obtains fairly good performance by default most of the time
- Avoid the worst performance due to bad inefficient processor mappings.

Syntax :

ibrun <Executable>

Application Binary Interface Compatibility

ABI compatible with both Intel MPI and MVAPICH2.

- Your compiled applications can link with pre-built (binary) libraries
- Compile once and run with any MPI stack in the future
- System module will be automatically switched when necessary
- Many tricky settings through Lmod and other TACC settings.
  - users do not need to worry about

Personal tip:

✓ No need to recompile your program when switching the MPI stacks

Outline

- Compile and run your MPI program at TACC Clusters
- Profiling your MPI program
- Best practices and performance tuning for Intel MPI

MPI Performance

MPI Performance depends on many factors

- CPUs (Frequency, Number of cores, Cache sizes)
- Memory (Amount, Frequency)
- Network speed
- Size of the jobs
- Communication pattern
- Type of program: Pure MPI, Hybrid (MPI + OpenMP)

MPI libraries have to make choices!

- Why? Because the number of combinations is too large!
- Are these choices optimal for my application? Maybe, but not necessarily.
- Can we change them? Yes

General Information

1. Understand the Hardware architecture
   - lscpu
   - lstopo --output-format pdf lstopo_frontera.pdf
   - cpuinfo > cpuinfo_arch.txt (IMPI Only)
2. Understand the Interconnect information
   1. fi_info ofed_info, ucx_info
   2. ibv_devices , ibv_devinfo
3. Understand MPI environments variables and their default values
   - module show impi
   - Intel MPI : impi_info > impiinfo_arch.txt

- Understand application requirements and identify yourself best tunings.

General Information

impi_info > impiinfo_arch.txt

- I_MPI_PIN = on
- I_MPI_PIN_CELL = unit
- I_MPI_PIN_DOMAIN = auto:compact
- I_MPI_PIN_ORDER = compact
- I_MPI_PIN_PROCESSOR_LIST
- I_MPI_PIN_PROCESSOR_EXCLUDE_LIST

The Choice of Benchmarks

Different MPI Libraries means “tuning based on different benchmarks”

- Intel MPI: Intel MPI Benchmarks (IMB)
  https://software.intel.com/en-us/articles/intel-mpi-benchmarks/
- MVAPICH2: OSU Micro-Benchmarks (OMB)
  http://mvapich.cse.ohio-state.edu/benchmarks/

IMB or OMB, which one is the best?

- Both are communication intensive without computation.
- It depends. Your application may be similar to one/some of the tests.
- The best benchmark is your own application!

Optimizing/Tuning

MPI Profiling is important

- To identify which MPI functions are used
  - Read the original source code if possible
  - Profile your application
- Profiling provides you the information regarding MPI communications (message size, time spent, MPI operations, etc.)
- Could be integrated in the MPI library
- Lots of tools that can help you to profile your applications
  - Intel Vtune, Intel Advisor, ITAC
  - Remora, TAU, Scalasca, IPM

Profile with Intel MPI (2018)

Gather MPI communication statistics :

I_MPI_STATS=[native:][n-]m | ipm
I_MPI_STATS_SCOPE="<subsystem>[:<ops>][;<subsystem>[:<ops>][...]]"

I_MPI_STATS verbosity levels	
Statistics level (n,m)	**Output information**
1	Amount of data sent by each process
2	Number of calls and amount of transferred data
3	Statistics combined according to actual call arguments
4	Statistics defined by a bucket list
10	Collective operation statistics for all communication contexts
20	Additional timing information for all MPI functions

Courtesy : Intel

Profile with Intel MPI (2018)

Communication Activity

Operation	Volume(MB)	Calls	Min time	Avr time	Max time	Total time
Collectives						
Allgather	0.000000e+00	0	0.00	0.00	0.00	0.00
Allgatherv	0.000000e+00	0	0.00	0.00	0.00	0.00
Allreduce	0.000000e+00	0	0.00	0.00	0.00	0.00
Alltoall	2.354677e+02	16170	12.87	117.08	14312.03	1893112.42
Alltoallv	0.000000e+00	0	0.00	0.00	0.00	0.00
Alltoallw	0.000000e+00	0	0.00	0.00	0.00	0.00
Barrier	0.000000e+00	16212	0.95	4.14	842.81	67178.96
Bcast	0.000000e+00	0	0.00	0.00	0.00	0.00
Exscan	0.000000e+00	0	0.00	0.00	0.00	0.00
Gather	0.000000e+00	0	0.00	0.00	0.00	0.00
Gatherv	0.000000e+00	0	0.00	0.00	0.00	0.00
Reduce_scatter	0.000000e+00	0	0.00	0.00	0.00	0.00
Reduce	4.806519e-04	63	0.95	3.46	55.07	217.68
Scan	0.000000e+00	0	0.00	0.00	0.00	0.00
Scatter	0.000000e+00	0	0.00	0.00	0.00	0.00
Scatterv	0.000000e+00	0	0.00	0.00	0.00	0.00
Iallgather	0.000000e+00	0	0.00	0.00	0.00	0.00
Iallgatherv	0.000000e+00	0	0.00	0.00	0.00	0.00
Iallreduce	0.000000e+00	0	0.00	0.00	0.00	0.00
Ialltoall	0.000000e+00	0	0.00	0.00	0.00	0.00
Ialltoallv	0.000000e+00	0	0.00	0.00	0.00	0.00
Ialltoallw	0.000000e+00	0	0.00	0.00	0.00	0.00
Ibarrier	0.000000e+00	0	0.00	0.00	0.00	0.00
Ibcast	0.000000e+00	0	0.00	0.00	0.00	0.00
Iexscan	0.000000e+00	0	0.00	0.00	0.00	0.00
Igather	0.000000e+00	0	0.00	0.00	0.00	0.00
Igatherv	0.000000e+00	0	0.00	0.00	0.00	0.00
Ireduce_scatter	0.000000e+00	0	0.00	0.00	0.00	0.00
Ireduce	0.000000e+00	0	0.00	0.00	0.00	0.00
Iscan	0.000000e+00	0	0.00	0.00	0.00	0.00
Iscatter	0.000000e+00	0	0.00	0.00	0.00	0.00
Iscatterv	0.000000e+00	0	0.00	0.00	0.00	0.00
Comm	0.000000e+00	0	0.00	0.00	0.00	0.00

Gather MPI communication statistics

export I_MPI_STATS=10

Select/filter the information you are interested in

export I_MPI_STATS_SCOPE="p2p;coll"

export I_MPI_STATS_SCOPE="p2p;coll:bcast,reduce"

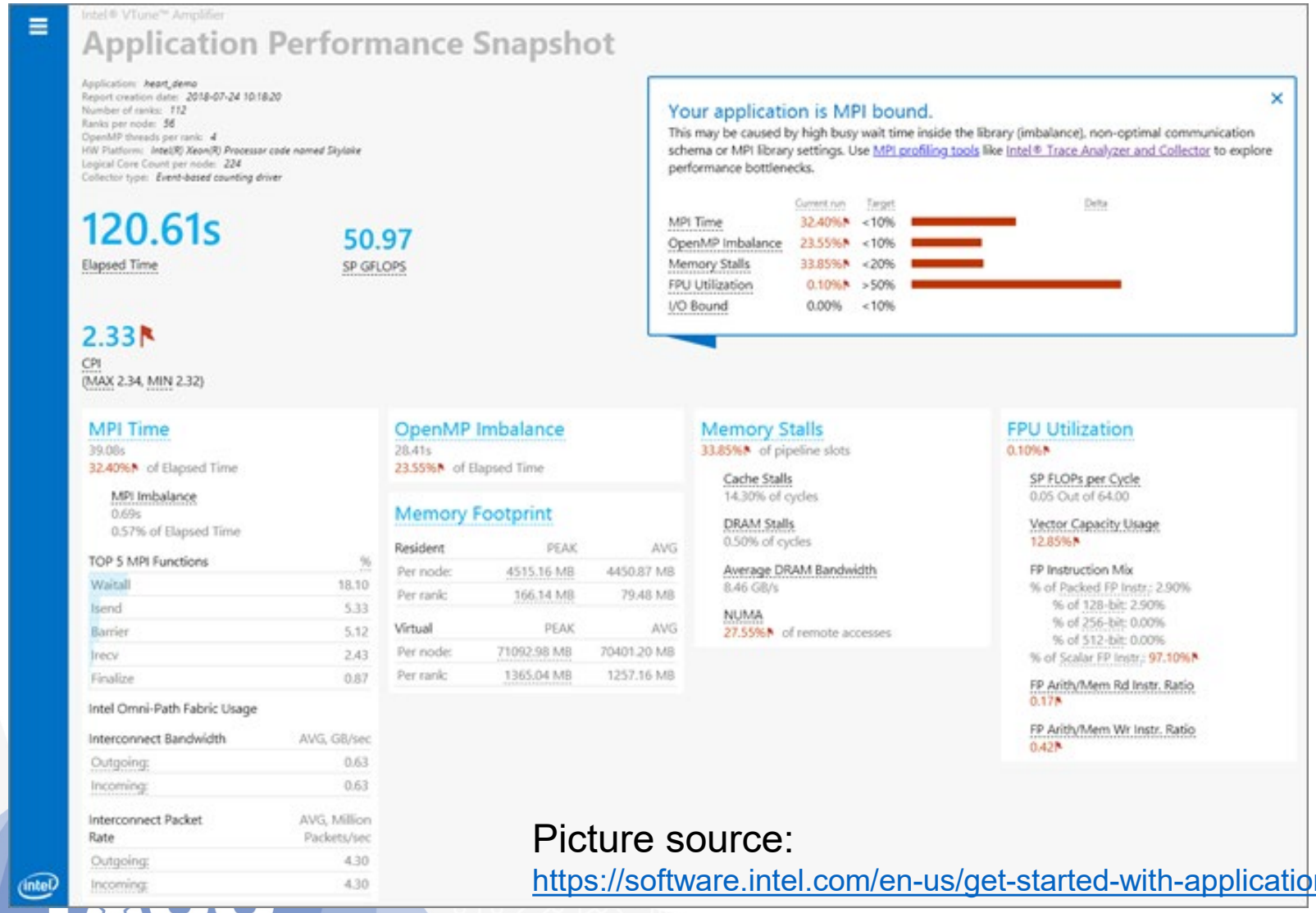
Specify the profile output

export I_MPI_STATS_FILE=myprofile

Links : link1 link2

Intel Application Performance Snapshot (APS)

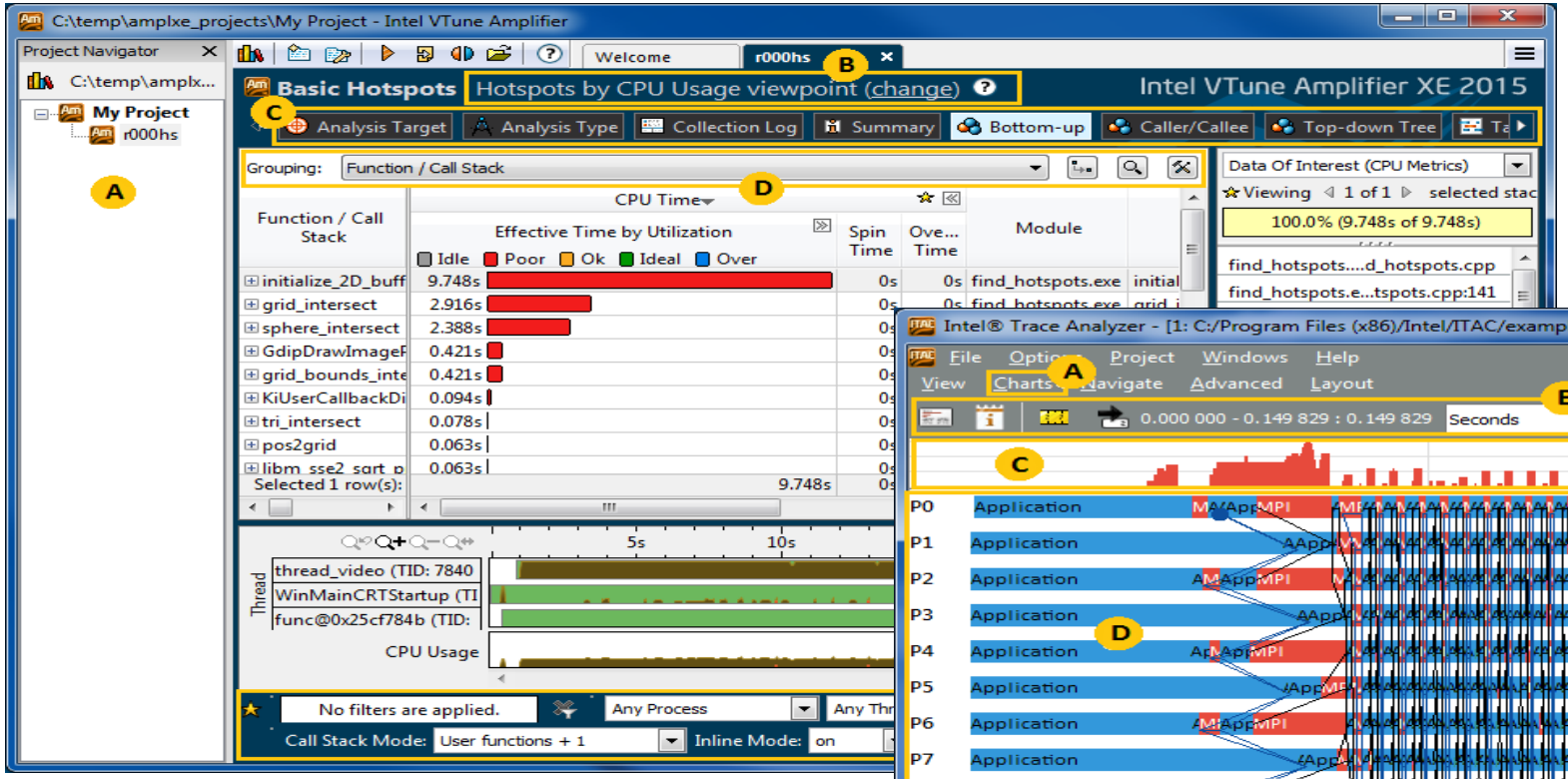
APS is part of Intel® Parallel Studio or Intel® VTune™ Amplifier in 2019 version.



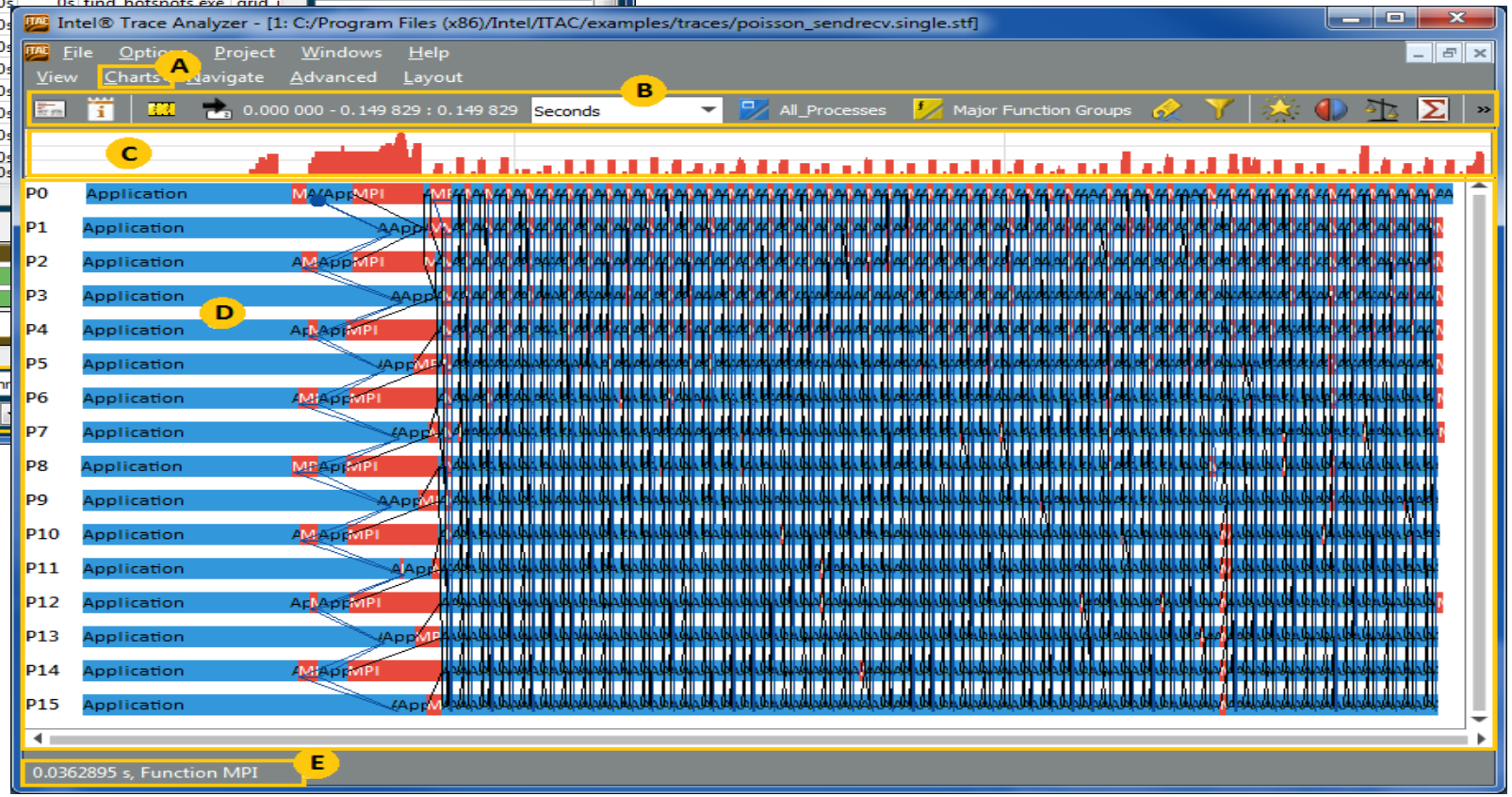
- Analyzes application's time spent in MPI, OpenMP, memory, I/O, etc.
- Display basic performance enhancement opportunities

Picture source:
https://software.intel.com/en-us/get-started-with-application-performance-snapshot

Profile with your favorite Tools (1)



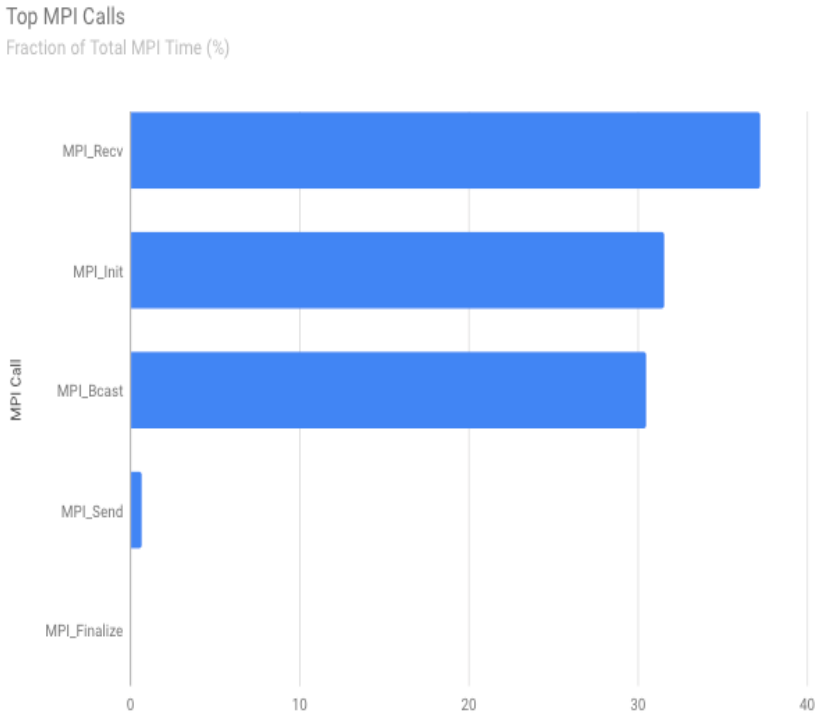
Intel® VTune Amplifier performance profiler (VTune)

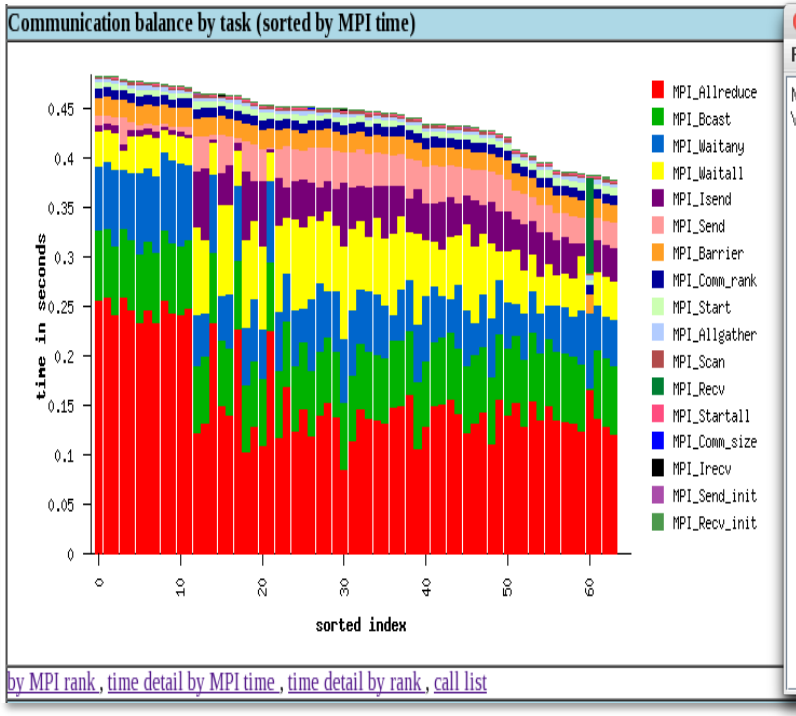


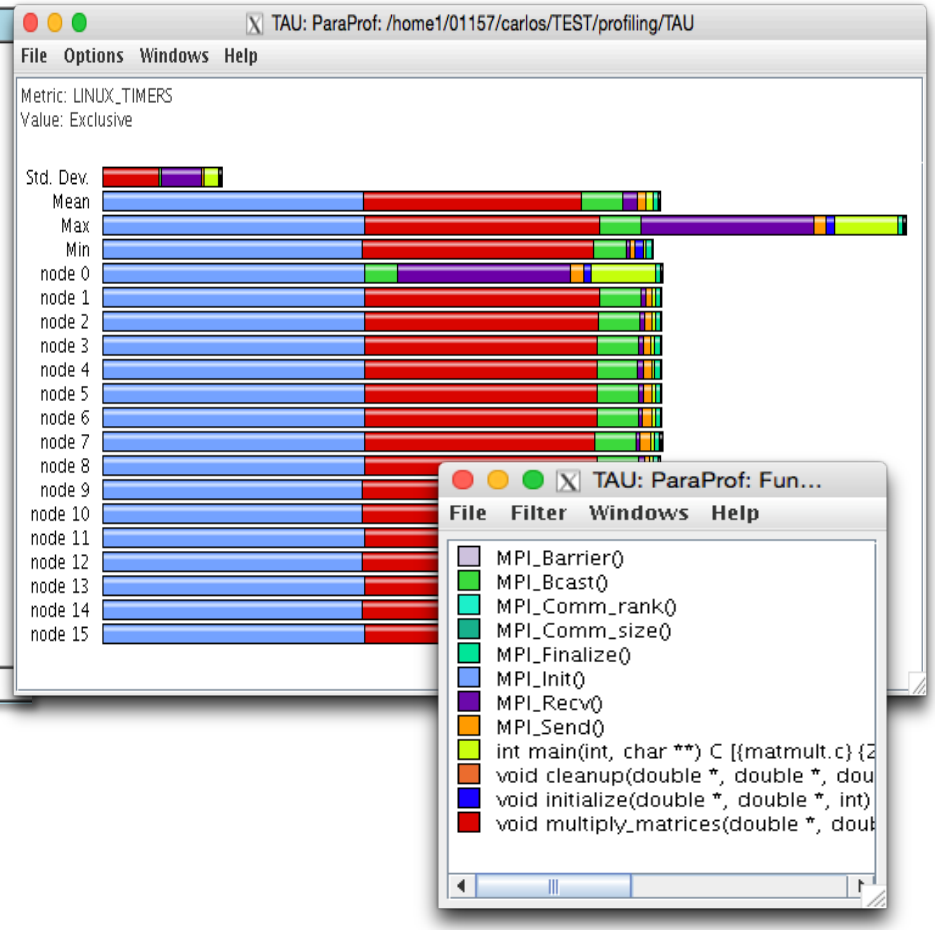
Intel® Trace Analyzer and Collector (ITAC)

Image source https://software.intel.com/en-us/node/561851 https://software.intel.com/en-us/node/561850

Profile with your favorite Tools (2)







Remora (Resource Monitoring for Remote Applications)

IPM (Integrated Performance Monitoring)

TAU (Tuning and Analysis Utilities)

Processor Affinity / CPU Pinning / Process Mapping

- Binding a process or a thread to a computing resource.
- Mostly handled by launchers. But this could be tricky when using
  - Hybrid Programs
  - on multi-core systems
  - on hyperthreaded systems
  - using complicated workflow (extra wrappers, launchers, etc.)
- Use environment variables (KMP_AFFINITY, OMP_PROC_BIND, MV2_USE_AFFINITY, I_MPI_PIN_DOMAIN) carefully
- Monitor run-time & process/thread affinity using tools like core_usage, show_affinity, amask, remora

Personal tip: Use all possible computing resources when possible and do not bind all process to few physical cores!

Outline

- Compile and run your MPI program at TACC Clusters
- Profiling your MPI program
- Best practices and performance tuning for Intel MPI

Tuning Intel MPI

- Node Pinning
- Processor Pinning
  - Pin less Jobs
  - Pure MPI Jobs : (Explicit, Predefined, and Generalized Formats)
  - Hybrid/Threading jobs : (Multi-core Shape, Explicit Shape, Explicit Shape Mask Formats)
- Debug Information
- Collective Operations Control
- Useful Environment variables in Intel 18
- Useful Environment variables in Intel 19

Node Pinning

Option 1 : IBRUN_TASKS_PER_NODE (ibrun specific)

export IBRUN_TASKS_PER_NODE=2; ibrun hellompi

Option 2 : I_MPI_PERHOST

export I_MPI_PERHOST=2 I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0 ; mpiexec hellompi

Option 3 : -pernode or -ppn

e.g 1 : export I_MPI_JOB_RESPECT_PROCESS_PLACEMENT=0 ; mpiexec -ppn 2 hellompi

e.g 2 : mpiexec -n 4 -ppn 2 hellompi

Option 4 : -f <machinefile>

mpiexec -n 4 -ppn 2 -f hosts hellompi

Option 5 : I_MPI_HYDRA_HOST_FILE=<arg> or I_MPI_HYDRA_HOST_GROUP=<arg>

e.g. "hostA[01-05],hostB,hostC[01-05,07,09-11]"

Hostfile Format -1

$ cat hosts

c591-104
c591-104
#c591-104
c591-105
c591-105
c591-105

Hostfile Format -2

$ cat hosts

c591-104:2
c591-105:2

Processor Pinning in IMPI : Three formats

1. I_MPI_PIN=0 : No Pinning
2. I_MPI_PIN_PROCESSOR LIST : Pure MPI applications
3. I_MPI_PIN_DOMAIN : Hybrid MPI + threading applications

Processor Pinning – Enable/Disable

I_MPI_PIN : Turn on/off process pinning

Syntax : I_MPI_PIN=<arg>

Arguments :

<arg>	Binary indicator
enable | yes | on | 1	Enable process pinning. This is the default value.
disable | no | off | 0	Disable processes pinning

Processor Pinning - Mode

I_MPI_PIN_MODE : Choose the pinning method

Syntax : I_MPI_PIN_MODE=<pinmode>

Arguments :

<pinmode>	Choose the CPU pinning mode
mpd	Pin processes inside the Multipurpose Daemon (Deprecated)
pm	Pin Processes inside Hydra launcher. Pin before MPI Processes launch.
lib	Pin processes inside the Intel MPI Library

Processor Pinning - Granularity

I_MPI_PIN_CELL : Pinning resolution granularity

Syntax : I_MPI_PIN_CELL=<cell>
(For non hyper-threaded systems, both are equal)

Arguments :

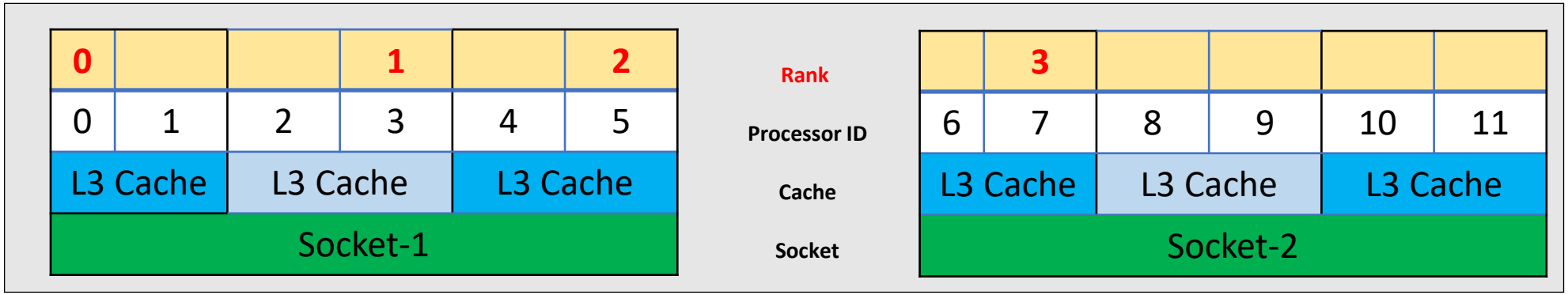
<cell>	**Specify the resolution granularity**
unit	Basic processor unit (**logical CPU**) Default for I_MPI_PIN_DOMAIN
core	**Physical processor core** (Each core contain one or more units/ basic processors)

I_MPI_PIN_PROCESSOR_LIST : Explicit Mapping format

Explicit Mapping : <proclist> : **<l>** or **<l>-<m>** or **<l>-<m>,<n> ; l, m, n are logical processors**

Example : To specify that processes must be bound to logical CPUs 0, 3 , 5, and 7:

$ mpirun –genv **I_MPI_PIN_PROCESSOR_LIST 0,3,5,7** -n <# total processes> ./app



I_MPI_PIN_PROCESSOR_LIST : Predefined Mapping format

Predefined Mapping Schemes : [<procset>][:map=<map>]

<procset> all : Use all logical cores
<procset> allcores : Use all logical processors that belong to different cores
<procset> allsocks : Use all logical processors that belong to different sockets

<map> Bunch : Maps processes closely on sockets that allows sharing of resources like caches etc.
<map> Scatter : Maps processes as far as possible on different sockets so as not to share resources.
<map> Spread : Maps processes consecutively with the possibility not to share common resources.

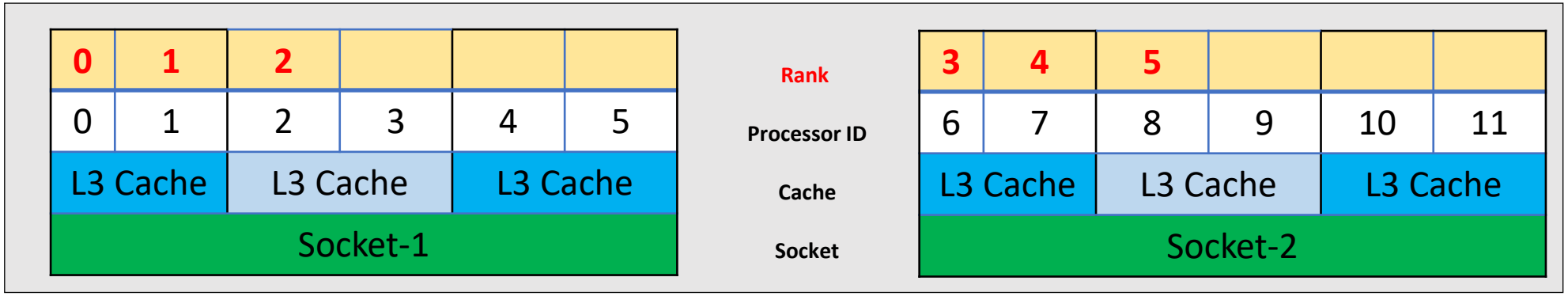
Examples :

1. To place the processes exclusively on physical cores regardless of Hyper Threading mode,
   $ mpirun –genv **I_MPI_PIN_PROCESSOR_LIST allcores** -n <# total processes> ./app

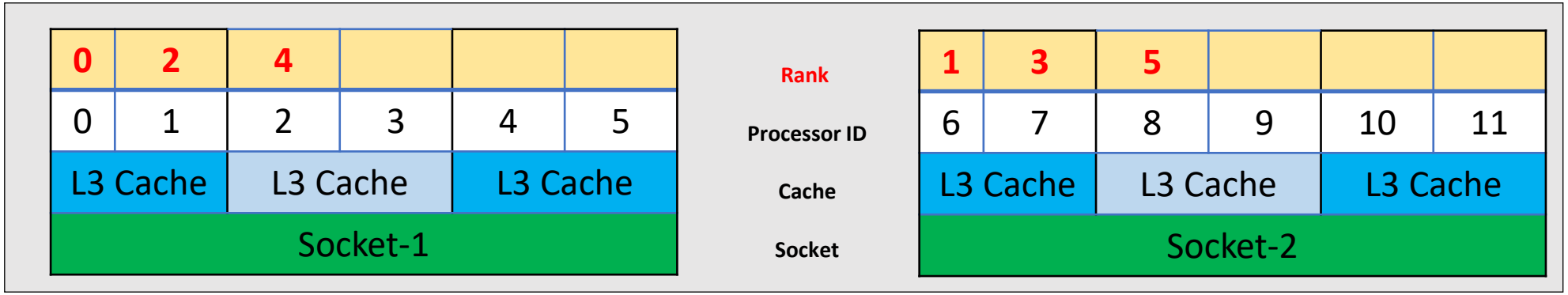
2. To avoid sharing of common resources by adjacent MPI processes, use map=scatter setting
   $ mpirun –genv **I_MPI_PIN_PROCESSOR_LIST map=scatter** -n <# total processes> ./app

Predefined Mapping Schemes

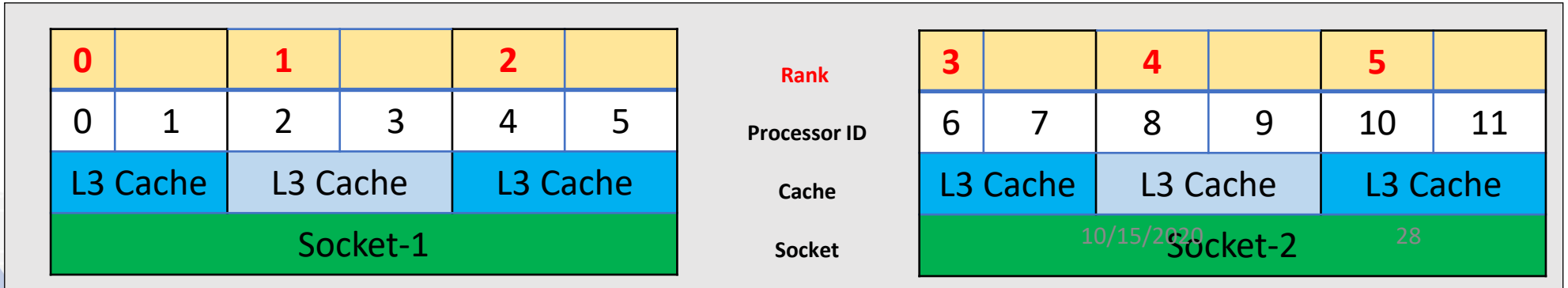
all:map=bunch



all:map=scatter



all:map=spread



I_MPI_PIN_PROCESSOR_LIST : Generalized Mapping format

Generalized Mapping

[<procset>][:[grain=<grain>][,shift=<shift>][,preoffset=<preoffset>][,postoffset=<postoffset>]

<procset> : all | **allcores** | allsocks
<grain> : Pinning granularity cell within procset.
<shift> : Default = 1
<preoffset> : Default = no shift
<postoffset> : Default = no shift ; alias : offset
Values of grain, shift, preoffset, postoffset are : (<n>, fine, core, cache1, cache2, cache3, sock etc.)

Example :

To place adjacent MPI processes on different sockets, use shift=socket:
$ mpirun –genv **I_MPI_PIN_PROCESSOR_LIST shift=socket** -n <# total processes> ./app

To place adjacent MPI processes with grain=2, shift=3 and offset=0:
$ mpirun –genv **I_MPI_PIN_PROCESSOR_LIST all:grain=2,shift=3** -n <# total processes> ./app

Generalized Mapping : Grain/Shift/Offset

Rank ->

Processor ID ->

A)

0 1			2 3			…	2n-2 2n-1		
0 1	2 3	4 5	6 7	8 9	10 11	…	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

B)

0 1	2n 2n+1		2 3	2n+2 2n+3		…	2n-2 2n-1	4n-2 4n-1	
0 1	2 3	4 5	6 7	8 9	10 11	…	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

Place process according to Grain=2, Shift=3, Offset=0

I_MPI_PIN_PROCESSOR_LIST all:grain=2,shift=3

C)

0 1	2n 2n+1	4n 4n+1	2 3	2n+2 2n+3	4n+2 4n+3	…	2n-2 2n-1	4n-2 4n-1	6n-2 6n-1
0 1	2 3	4 5	6 7	8 9	10 11	…	6n-6 6n-5	6n-4 6n-3	6n-2 6n-1

D)

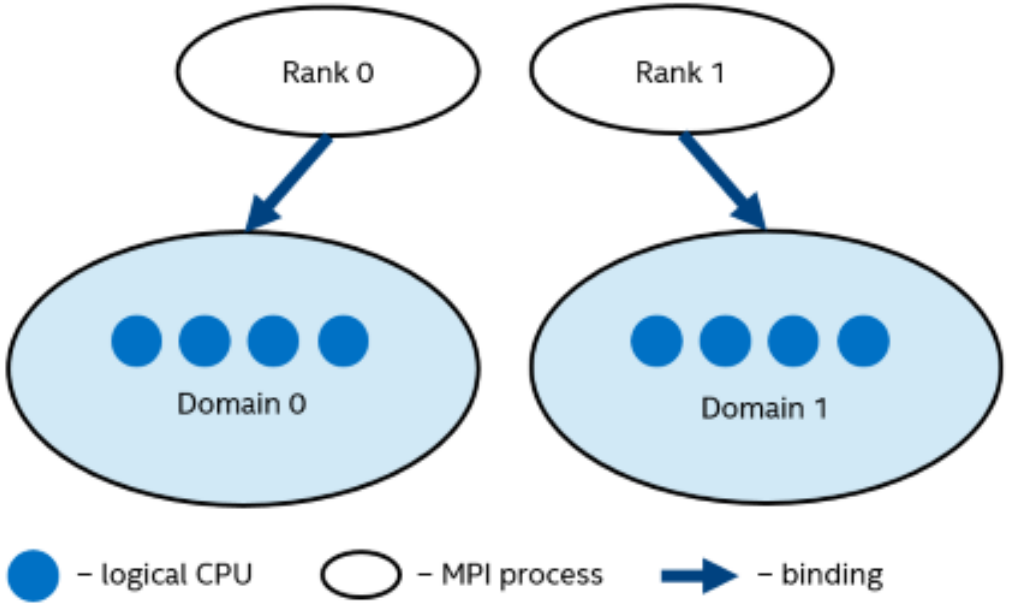
0 1	2 3	…	2n-2 2n-1	2n 2n+1	2n+2 2n+3	…	4n-2 4n-1	4n 4n+1	4n+2 4n+3	…	6n-2 6n-1
0 1	6 7	…	6n-6 6n-5	2 3	8 9	…	6n-4 6n-3	4 5	10 11	…	6n-2 6n-1

Custom Pinning for Hybrid MPI Jobs

I_MPI_PIN_DOMAIN: <mc_shape> or <Masklist> or <Size>:<Layout>

- Define non-overlapping subsets/domains of logical processors on a node, and a set of rules on how MPI processes are bound to these domains by the following formula: ***one MPI process per one domain***

I_MPI_PIN_ORDER : - Mapping order for MPI processes to domains. Values are Bunch, Spread, Scatter, Compact, Range



Courtesy : intel

Custom Pinning for Hybrid MPI Jobs

I_MPI_PIN_DOMAIN: <mc_shape> or <Masklist> or <Size>:<Layout>

<mc_shape>

- core
- socket
- numa
- node
- cache1
- cache2
- cache3
- cache

<masklist> $[m_1,m_2,\ldots,m_n]$

Each m_i is a hexadecimail bit mask defining an individual domain.

<size>[:<layout>]

a. size

- omp (OMP_NUM_THREADS)
- auto (size = # of Logical Processors / # of Processes)
- <n>

b. layout (Ordering of Domains)

- Platform (according to bios numbering)
- Compact (Share resources : Core, Cache, Sockets)
- Scatter

Pinning order

I_MPI_PIN_ORDER : - Mapping order for MPI processes to domains. Values are Bunch, Spread, Scatter, Compact, Range

<order>	Specify the ranking order of Domains
range	According to the processor's BIOS numbering (Platform-dependent)
scatter	Adjacent domains have minimal sharing of common resources
compact *	Adjacent domains share common resources as much as possible.
spread	Ordered consecutively with the possibility not to share common resources
bunch	Processes are mapped proportionally to sockets and the domains are ordered as close as possible on the sockets

Try : Pinning Simulator

Pinning threads within a domain

OMP_PLACES=threads

Example Run : 1 Node, 6 Ranks, 1 Rank per tile, 2 Threads per core on KNL

$ export OMP_NUM_THREADS=4

$ mpiexec -np 6 -env I_MPI_PIN_DOMAIN 4:compact -env OMP_PLACES=threads hybrid2

Display Pinning Information

1. Rank Mapping : -print-rank-map environment variable
2. Get detailed information about an MPI application at runtime.

   I_MPI_DEBUG = 0 to 1000.
3. Add “+” to prefix each line with string <rank>#<pid>@<hostname> e.g +3

<level>	Indicate level of debug information
0	Print no debugging information. This is the default value
1	Output verbose error diagnostics
2	Confirm which I_MPI_DEVICE was used
3	Output effective MPI rank, pid and node mapping table
4	Print process pinning information
5	Print Intel MPI specific environnent variables
> 6	Add extra levels of debug information

Useful Environment Variables for Intel MPI

Intel MPI Collective Operations Control

Choose the best collective algorithms

I_MPI_ADJUST_<*opname*> control collective operation algorithm selection

Recommended workflow:

- Use your favorite profiler to locate the most time-consuming collective operations in your programs
- Use the micro benchmarks (with different algorithms) to find out potential best algorithm(s) for the target collective operations
- Switch the algorithm for your own MPI programs

Intel MPI Collective Operations Control

Env Variable	Collective	Algorithms
I_MPI_ADJUST_BCAST	MPI_Bcast	1. Binomial algorithm 2. Recursive doubling algorithm 3. Ring algorithm 4. Topology aware binomial algorithm 5. Topology aware recursive doubling algorithm 6. Topology aware ring algorithm 7. Shumilin's bcast algorithm
I_MPI_ADJUST_GATHERV	MPI_Gatherv	1. Linear 2. Topology aware linear 3. Knomial
I_MPI_ADJUST_SCATTER	MPI_Scatter	1. Binomial 2. Topology aware binomial 3. Shumilin's

Fine Control :

I_MPI_ADJUST_BCAST=4:0-100,5001-10000;**1:101-3200;**2:3201-5000;3

Select Algorithm 4 for message size 0 to 100 and 5001 to 10K, Algorithm 1 for message size 101 to 3200, Algorithm2 for 3201to 5000 message size, and Algorithm3 in all other cases.

Link

Message Exchange Protocols

Eager

- Asynchronous protocol that allows to send data immediately irrespective of receiver state
- Send operation completes without acknowledgement from a matching receive
- Best communication performance for smaller messages

Rendezvous

- Synchronous protocol which requires an acknowledgement from a matching receive for the send operation to complete
- Best communication performance for larger messages

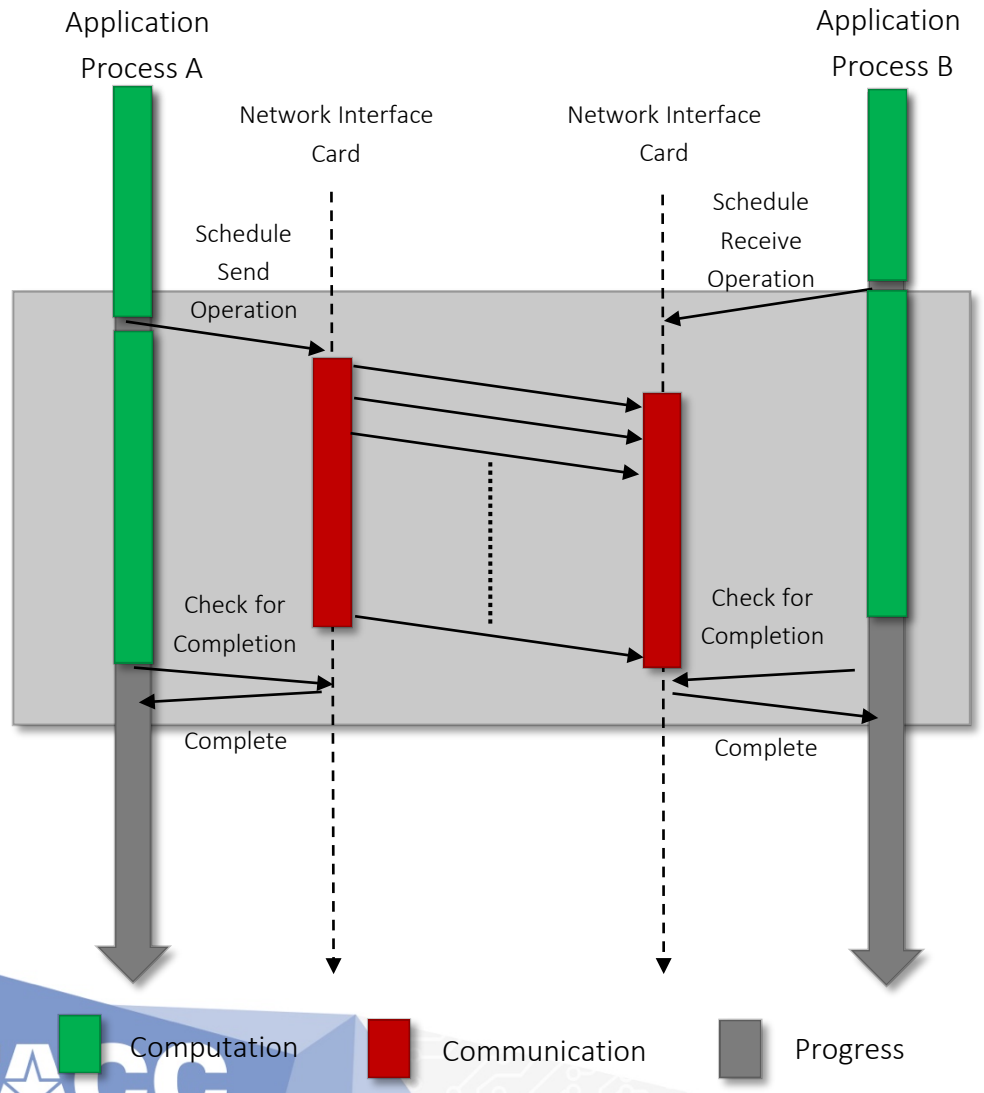
Environment Variable :

Inter/Intra Node : I_MPI_EAGER_THRESHLOD

Intra Node : I_MPI_INTRANODE_EAGER_THRESHLOD

But what about overlap?

Analyzing Overlap Potential of Eager Protocol



Application processes schedule communication operation

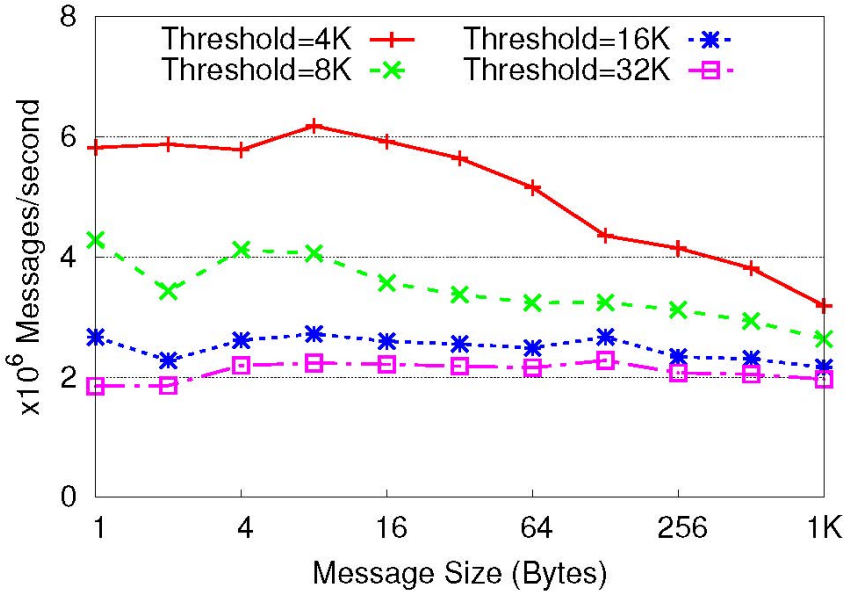
Network adapter progresses communication in the background

Application process free to perform useful compute in the foreground

Overlap of computation and communication => Better Overall Application Performance

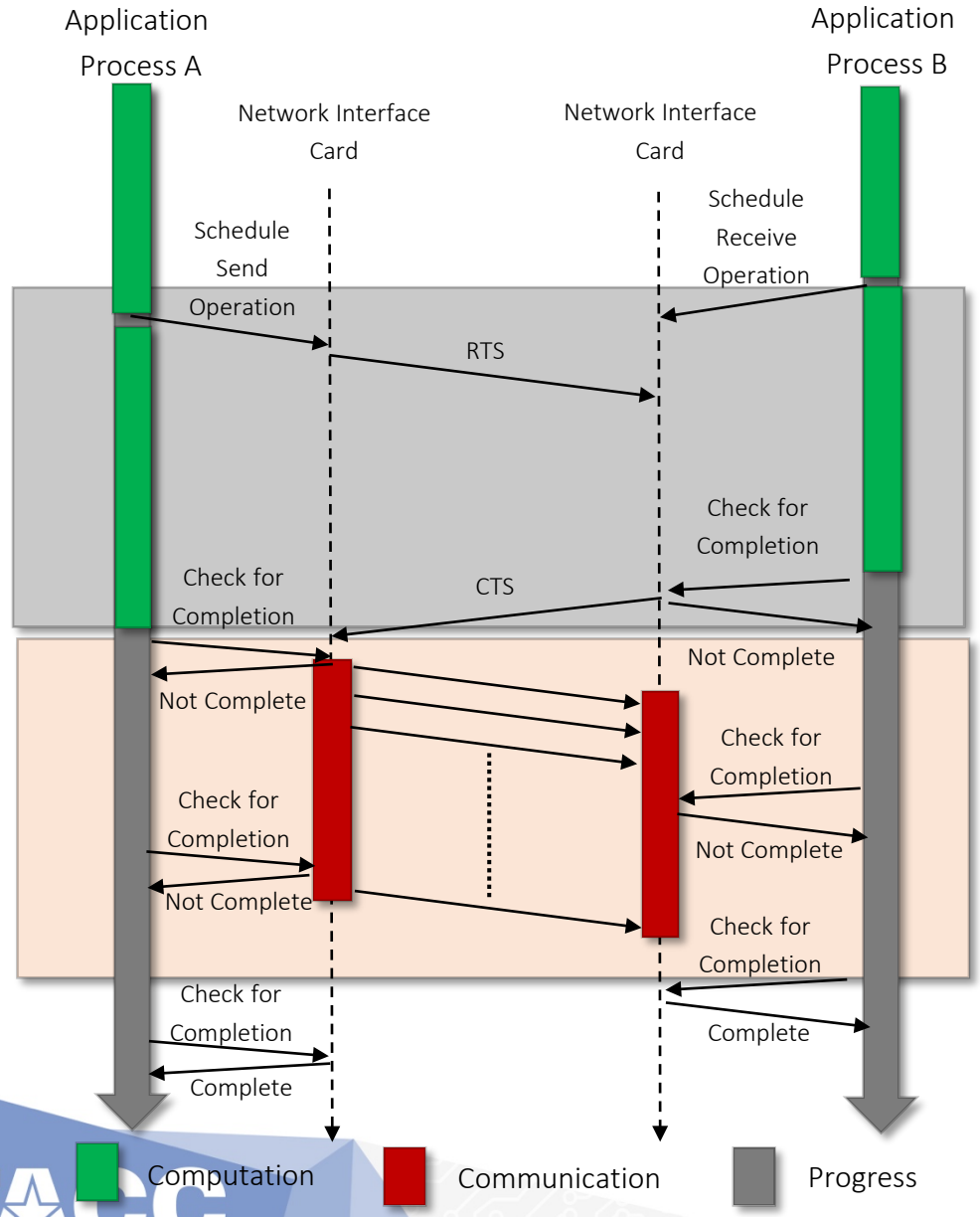
Increased buffer requirement

Poor communication performance if used for all types of communication operations



Impact of changing Eager Threshold on performance of multi-pair message-rate benchmark with 32 processes on Stampede

Analyzing Overlap Potential of Rendezvous Protocol



Application processes schedule communication operation

Application process free to perform useful compute in the foreground

Little communication progress in the background

All communication takes place at final synchronization

Reduced buffer requirement

Good communication performance if used for large message sizes and operations where communication library is progressed frequently

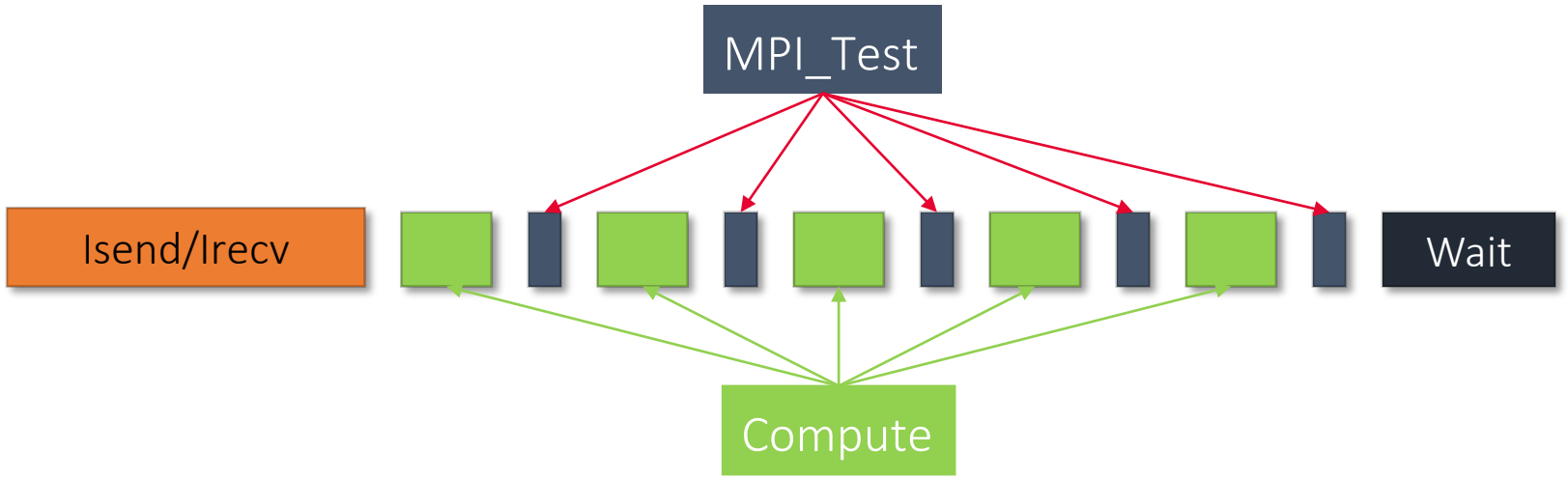
Poor overlap of computation and communication => Poor Overall Application Performance

Asynchronous Progress Methods

- Hardware-based progression – Not generic
- Software-based progression
  - Host application based (Manual progression)
  - Kernel assisted: Require root privileges
  - Thread/Process based

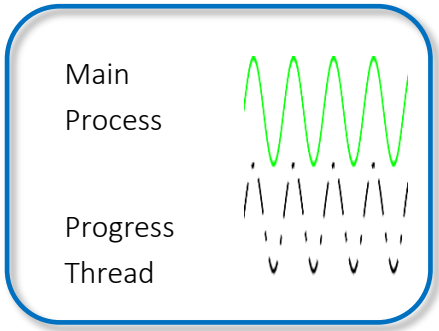
Asynchronous Progress: Host Application based

- MPI_Test() calls inserted between compute operations
- Difficult to identify where MPI_Test() to be inserted
- Require domain knowledge as application code has to be modified

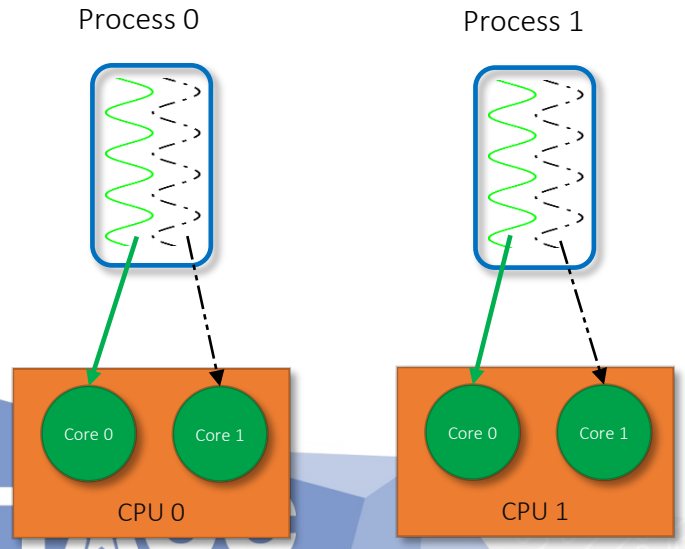


Methods of Asynchronous Progress : Thread/Process based

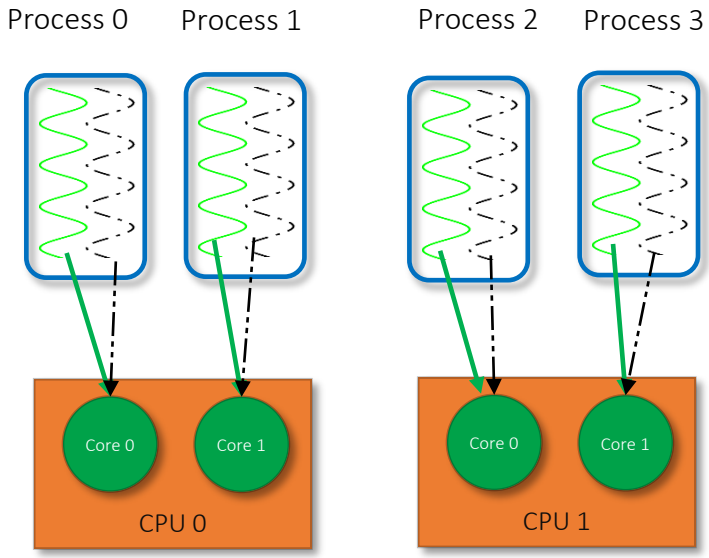
- Progress threads are created for non-blocking message communication
- Two approaches
  - Individual progress thread for each user process - 1:1
    - Partially Subscribed
    - Fully subscribed
  - Separate progress processes for a group of user processes - 1:N



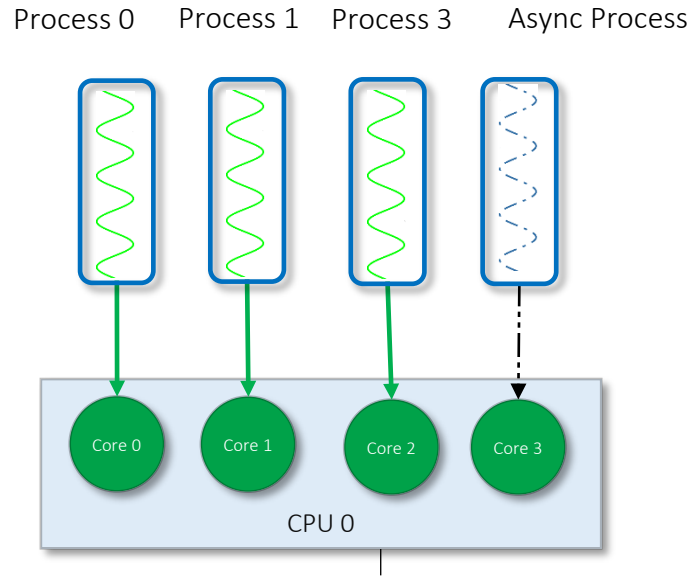
Partial Subscription



Full Subscription



Separate progress processes



Asynchronous Progress Options

- **I_MPI_ASYNC_PROGRESS=1**
  - Sets to 1 for enabling progress threads per process
- **I_MPI_ASYNC_PROGRESS_THREADS=n**
  - Sets number of asynchronous progress for each process

"*Efficient design for MPI asynchronous progress without dedicated resources*" A. Ruhela, H. Subramoni, S. Chakraborty, M. Bayatpour, P. Kousha, and D.K. Panda, Parallel Computing, Volume 85, 2019,

Useful Environment Variables (Intel 18)

- **I_MPI_FABRICS=shm:ofa (ofi for Intel 19)**

  Select the best available communication fabric : {shm, dapl, tcp, tmi, ofa, ofi}
  - Use shared memory for the process running on same node
  - Use OFA-capable network fabric including InfiniBand for processes between nodes

- **I_MPI_FALLBACK=0**

  Disable I_MPI_FALLBACK to avoid unexpected performance slowdowns.

  If enabled, fallbacks to the first available fabric (tcp or shm:tcp).

- **I_MPI_JOB_FAST_STARTUP=1**

  Turn on the faster Intel MPI Library process startup algorithm

- **I_MPI_HYDRA_PMI_CONNECT=alltoall**

  Define the processing method for process management interface (PMI) messages

- **I_MPI_WAIT_MODE=1**

  Enable the wait mode of the progress engine in order to wait for messages without polling the fabrics.
  Saves polling time but increase message-response rate.

- **I_MPI_LARGE_SCALE_THRESHOLD**

  Set for connectionless communication (By default triggered after 4096 processes)

Useful Environment Variables (Intel 2019)

- **I_MPI_PLATFORM_CHECK=0**
  Prevents MPI from testing for homogeneity of nodes and saves initialization time

- **I_MPI_EXTRA_FILESYSTEM=enable I_MPI_EXTRA_FILESYSTEM_LIST=luster**
  Use accelerated MPI I/O

- **I_MPI_JOB_STARTUP_TIMEOUT :** Sets the mpiexec.hydra job startup timeout.

- **I_MPI_JOB_TIMEOUT or (MPIEXEC_TIMEOUT) :** Sets the jobs time.

- **I_MPI_TUNING_MODE :** Automatically find the best algorithms for collective operations

At the End of this Talk ... …

- "**How to tune MPI**" cannot be found easily inside a notebook.
- Some tuning options are specific to corresponding MPI library.

  But the concepts are mostly common.
- Some options can have counter-effects!

  different MPI stack, different applications, different job size
- Tuning could be time consuming, but in long term, it might be worth it.

References

- Intel MPI Developer Reference for Linux

  https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top.html

- TACC staff, TACC Frontera user guide
  1. https://portal.tacc.utexas.edu/user-guides/frontera
  2. https://portal.tacc.utexas.edu/user-guides/stampede2
  3. https://portal.tacc.utexas.edu/user-guides/longhorn

- OFI : https://github.com/ofiwg/ofi-guide/blob/master/OFIGuide.md

TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

LAB –
Best Practices for MPI Usage

PRESENTED BY:

Amit Ruhela (aruhela@tacc.utexas.edu)

Oct 15, 2020

Agenda

1. Get System Information and Terminology : System Details, MPI Library, Interconnects (HCA),
2. Depict Pinning Information
    1. show_affinity
    2. amask
    3. Environment variables : I_MPI_DEBUG, MV2_SHOW_CPU_BINDING
3. Profile MPI Applications
    1. I_MPI_STATS
    2. APS
    3. Remora
4. Configure CPU Pinning
    1. MPI Programs
    2. Hybrid Programs
5. Manual Selection of Collective Algorithms
6. Asynchronous Progress Communication

Folder Contents

Location : ~train00/IXPUG20.tar.gz

```
├── docs
│   ├──Userguides_for_IMPI
├── hwinfo
│   ├── cpuinfo, lstopo information of Frontera and Stampede2 Clusters
├── exercises
│   ├── clean_npb.sh  install-npb.sh  npb
│   ├── examples/hellompi.c examples/hybrid2.c examples/omp.c examples/gen_hostfile.sh
```

Understand System Details

- Verify default modules : module list

- System Details
  - lscpu
  - lstopo --output-format pdf lstopo_CascadeLake.pdf
  - cpuinfo > cpuinfo_CascadeLake.txt

- MPI Details
  - module show impi
  - impi_info (Frontera)
  - mpiicc -v
- Interconnect details
  - PSM (Stampede2)
    - fi_info –l # list available libfabric providers
    - fi_info –e # print libfabric environment variables
  - InfiniBand (Frontera)
    - ibv_devices
    - ibv_devinfo # use –v for verbosity
    - ibdev2netdev
    - ofed_info
    - ucx_info -d | grep Transport # -s for system details

Terminology

- Each core contains one or more logical processors.
- Physical Id in "/proc/cpuinfo" command denote socket id.
- Core id is Physical identification of a Core whereas proc id or processor is logical core which are always numbered sequentially.
- Kernel calculate core id(physical) by referring to APIC id assigned by firmware and is therefore irregular.
- "/proc/cpuinfo"

```
physical id             : 0    # Socket ID
core id                 : 6    #  Physical Core id
processor/procid        : 2    # Logical Core
```

Intel MPI : Show Pinning Information

Method 1 show_affinity:

Open two terminals on Stampede2 or Frontera(One for program execution and other to visualize the affinity).

Terminal 1 :

- # Create an interactive session using idev command
  $ idev -N 1 -p development -A Frontera-Training
- # Run Intel Microbenchmark (IMB) alltoall
  $ ibrun $TACC_IMPI_BIN/IMB-MPI1 alltoall -npmin 56

Terminal 2 :

Find node-id (NODES field) using “squeue -u <uid>” command. Replace <uid> with your username in squeue command.

$ ssh <NODES>

$ show_affinity

Environment variables to explore :

1. export I_MPI_PIN=1
2. export I_MPI_PIN=0

Intel MPI : Show Pinning Information - Cont.

Method 2 :

IMPI

```
# Specify I_MPI_DEBUG=4 for Intel MPI
export I_MPI_DEBUG=4; ibrun $TACC_IMPI_BIN/IMB-MPI1 alltoall -npmin 56
```

Intel MPI : Show Pinning Information - Cont.

Method 3 amask

```
module load amask
export OMP_NUM_THREADS=4
export OMP_PLACES=cores
mpiexec -np 1     amask_omp      # Pure OpenMP
mpiexec -np 6     amask_mpi       #  Pure MPI
mpiexec -np 6     amask_hybrid    # Hybrid Program
```

Intel MPI : Processor Pinning of MPI Programs

Method : I_MPI_PIN_PROCESSOR_LIST

cd NPB3.4.1/NPB3.4-MPI/bin

export I_MPI_DEBUG=4
module load impi

mpiexec -np 36 -env I_MPI_PIN_PROCESSOR_LIST **0-18** bt.A.x
mpiexec -np 36 -env I_MPI_PIN_PROCESSOR_LIST **map=bunch** bt.A.x
mpiexec -np 36 -env I_MPI_PIN_PROCESSOR_LIST **map=scatter** bt.A.x
mpiexec -np 36 -env I_MPI_PIN_PROCESSOR_LIST **map=spread** bt.A.x
mpiexec -np 36 -env I_MPI_PIN_PROCESSOR_LIST **all:grain=2,shift=3** bt.A.x

. Additional Arguments :

- -print-rank-map for Intel 18

Manual Selection of Collective Algorithms

For Intel MPI

```
module load impi
export I_MPI_ADJUST_REDUCE=1 # Shumilin's Algorithm
ibrun IMB-MPI1 reduce -npmin 112

export I_MPI_ADJUST_REDUCE=2 # Binomial Algorithm
ibrun IMB-MPI1 reduce -npmin 112

…
```

Asynchronous Progress Communication

idev -N 2 -p development -A Frontera-Training

mpiexec -np 56 -ppn 28 -env I_MPI_ASYNC_PROGRESS=0 IMB-NBC -npmin 112 ialltoallv

mpiexec -np 56 -ppn 28 -env I_MPI_ASYNC_PROGRESS=1 IMB-NBC -npmin 112 ialltoallv

Exercise : Try with other non-blocking collective.

Asynchronous Progress Communication - iAlltoallv

```
#------------------------------------------------------------------------------
# Benchmarking Ialltoallv
# #processes = 56
#------------------------------------------------------------------------------
       #bytes #repetitions t_ovrl[usec] t_pure[usec]  t_CPU[usec]   overlap[%]
            0         1000         1.70         0.91         0.80         1.05
            1         1000        91.13        45.77        45.52         0.34
            2         1000        91.51        46.00        45.77         0.56
            4         1000        91.18        45.69        45.50         0.02
            8         1000        91.15        45.61        45.67         0.28
           16         1000        91.31        45.93        45.76         0.84
           32         1000        91.57        45.36        45.29         0.00
           64         1000        98.36        49.44        49.34         0.84
          128         1000       102.47        51.69        51.72         1.82
          256         1000       115.19        59.08        58.93         4.78
          512         1000       157.79        77.97        77.59         0.00
         1024         1000       207.18       106.20       110.74         8.82
         2048         1000       304.28       164.37       161.99        13.44
         4096         1000       556.00       311.53       307.08        20.10
         8192         1000      1056.95       609.11       607.64        26.23
        16384         1000      2074.70      1185.72      1182.55        24.76
        32768         1000      4249.80      2381.94      2374.40        21.27
        65536          640      8887.17      4856.06      4839.44        16.65
       131072          320     18811.18      9576.94      9439.20         2.14
       262144          160     36417.26     18789.37     18485.32         4.56
       524288           80     74589.96     38681.14     38091.88         5.64
      1048576           40    146735.08     75545.86     74427.53         4.29
      2097152           20    293865.87    149983.46    147509.04         2.42
      4194304           10    587221.77    293028.75    291689.43         0.00
```

```
#------------------------------------------------------------------------------
# Benchmarking Ialltoallv
# #processes = 56
#------------------------------------------------------------------------------
       #bytes #repetitions t_ovrl[usec] t_pure[usec]  t_CPU[usec]   overlap[%]
            0         1000        16.25         8.35         8.10         2.38
            1         1000       210.07       140.75       139.89        50.13
            2         1000       194.53       146.99       146.83        67.55
            4         1000       240.24       120.98       121.51         1.85
            8         1000       254.07       123.49       136.73         4.50
           16         1000       176.48       132.64       132.09        66.54
           32         1000       268.79       167.57       170.74        40.72
           64         1000       254.62       149.28       151.27        30.37
          128         1000       251.25       185.66       184.90        64.26
          256         1000       205.10       136.05       136.58        49.44
          512         1000       269.93       188.56       187.79        56.44
         1024         1000       327.88       226.84       225.73        54.97
         2048         1000       501.46       332.37       343.50        50.77
         4096         1000       710.23       572.04       571.27        75.71
         8192         1000      1162.77       946.22       944.55        76.94
        16384         1000      1930.83      1680.44      1676.44        84.86
        32768         1000      3543.53      3209.49      3256.00        89.74
        65536          640      6362.87      5876.47      5922.84        91.79
       131072          320     11743.01     10888.13     10884.78        92.12
       262144          160     19768.26     18389.42     18690.58        92.62
       524288           80     39038.73     37895.06     38871.20        97.06
      1048576           40     77355.71     75175.29     77242.41        97.18
      2097152           20    152840.14    147415.93    151607.84        96.42
      4194304           10    305034.92    295731.55    304771.44        96.95
```

NAS Parallel Benchmarks - continued

- Five kernels
  - IS - Integer Sort, random memory access
  - EP - Embarrassingly Parallel
  - CG - Conjugate Gradient, irregular memory access and communication
  - MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
  - FT - discrete 3D fast Fourier Transform, all-to-all communication
- Three pseudo applications
  - BT - Block Tri-diagonal solver
  - SP - Scalar Penta-diagonal solver
  - LU - Lower-Upper Gauss-Seidel solver

- Ranks / Processes
  - Power of 2
  - Square
  - Cube

NAS Parallel Benchmarks

NAS Parallel Benchmarks : Collection of 5 Kernels and 3 pseudo applications

Website : https://www.nas.nasa.gov/publications/npb.html

How to get it ?

```
wget https://www.nas.nasa.gov/assets/npb/NPB3.4.1.tar.gz
tar -xvzf NPB3.4.1.tar.gz
cd NPB3.4.1/NPB3.4-MPI
cp config/make.def.template config/make.def
make bt CLASS=A      # Block Tri-diagonal solver : Application
make ft CLASS=A      # 3D Fast Fourier Transform : Kernel
make is CLASS=A      #  Integer Sort : Kernel
cd bin
```

Profiling with Intel MPI STATS

Only for Intel 18

Syntax :

1. I_MPI_STATS=[native:][n-]m
2. I_MPI_STATS_SCOPE="<subsystem>[:<ops>][;<subsystem>[:<ops>][...]]"
   - Subsystems are : all, p2p, coll; Ops are Bcast, Reduce, …

```
module load impi
export I_MPI_STATS=ipm
export I_MPI_STATS_SCOPE="p2p:coll"
export I_MPI_STATS_FILE=profile_bt.txt
cd NPB3.4.1/NPB3.4-MPI/bin
ibrun -n 36 ./bt.A.x
```

More details

Simulator https://software.intel.com/content/www/us/en/develop/articles/pi

I_MPI_STATS verbosity levels	
Statistics level (n,m)	**Output information**
1	Amount of data sent by each process
2	Number of calls and amount of transferred data
3	Statistics combined according to actual call arguments
4	Statistics defined by a bucket list
10	Collective operation statistics for all communication contexts
20	Additional timing information for all MPI functions

Intel Application Performance Snapshot

1. cd NPB3.4.1/NPB3.4-MPI/bin
2. module load vtune
3. aps ibrun -n 36 ./bt.A.x # rank can be any perfect square
4. aps -report=aps_result_*
5. View Graphical Report (HTML) in the same folder.

Profiling with Remora

1. cd NPB3.4.1/NPB3.4-MPI/bin
2. module load remora
3. remora ibrun -n 36 ./bt.A.x # rank can be any perfect square
4. View Graphical Report (HTML) in the same folder.

Thanks

Intel MPI : Process Pinning of Hybrid Programs

Method : I_MPI_PIN_DOMAIN

cd examples

mpicc -fopenmp -o hybrid hybrid.c

export I_MPI_DEBUG=4
export OMP_NUM_THREADS=4
export OMP_PLACES=cores

mpiexec -np 6 -env I_MPI_PIN_DOMAIN core hybrid
mpiexec -np 6 -env I_MPI_PIN_DOMAIN omp hybrid
mpiexec -np 6 -env I_MPI_PIN_DOMAIN omp:scatter hybrid

Try with cache1 cache2 etc

mpiexec -np 6 -env I_MPI_PIN_DOMAIN 4:compact hybrid