TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

All the things you need to know about Intel MPI Library

Jerome Vienne
viennej@tacc.utexas.edu
Texas Advanced Computing Center
The University of Texas at Austin
Austin, TX

November 12th, 2016

A Heterogeneous Environment

MPI performance depends on many factors

- CPUs (Number of cores, Cache sizes, Frequency)
- Memory (Amount, Frequency)
- Network Speed (10,20,40 ... Gbit/s)
- Size of the job
- Type of code: Hybrid (ex: OpenMP+MPI) or Pure MPI

A Heterogeneous Environment

MPI performance depends on many factors

- CPUs (Number of cores, Cache sizes, Frequency)
- Memory (Amount, Frequency)
- Network Speed (10,20,40 ... Gbit/s)
- Size of the job
- Type of code: Hybrid (ex: OpenMP+MPI) or Pure MPI

MPI libraries have to make choices

- Why ? Because the number of combinations is too large.
- Are these choices optimal for my application ? Not necessarily.
- Can we change them ? Yes, this is why we are there.

Aim of this talk

- ”How to tune MPI” cannot be found easily inside books.
- Show that MPI libraries are not black boxes.
- Describe concepts that are common inside MPI libraries.
- Understand the difference between MPI libraries.
- Provide some useful commands for Intel MPI.
- Result: Help you to reduce the time and memory foot print of your MPI application

Before to start

Warnings !!!

- Talk based on Intel MPI (few references to MVAPICH2 and OpenMPI).
- All experiments were done on Stampede supercomputer at TACC.
- Tuning options are specific to a MPI library ! But concepts are common.
- Options can have counter-effects !
- MPI libraries have lot of options for tuning, we will only cover the most important ones.
- Tuning could be time consuming, but long-term, it might be worth it

Plan

- Basic Tuning
  The Choice of the Benchmark
  Profiling
  Hostfile
  Process Placement
  To conclude

- Intermediate Tuning
  Inter-node Point-to-Point Optimization
  Intra-node Point-to-Point Optimization
  Collective Tuning
  To conclude

- Conclusion

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

The Choice of Benchmarks

Different MPI library = Tuning Based on Different Benchmarks

- Intel MPI: Intel MPI Benchmarks (IMB)
- MVAPICH2: OSU Micro-Benchmarks (OMB)

The Choice of Benchmarks

Different MPI library = Tuning Based on Different Benchmarks

- Intel MPI: Intel MPI Benchmarks (IMB)
- MVAPICH2: OSU Micro-Benchmarks (OMB)

IMB or OMB, which one is the best to use ?

- Both are communication intensive without computation
- Depend on your application
- **The best benchmark is your application !**

The Choice of Benchmarks

Different MPI library = Tuning Based on Different Benchmarks

- Intel MPI: Intel MPI Benchmarks (IMB)
- MVAPICH2: OSU Micro-Benchmarks (OMB)

IMB or OMB, which one is the best to use ?

- Both are communication intensive without computation
- Depend on your application
- **The best benchmark is your application !**

But... let's take a look at them in detail !

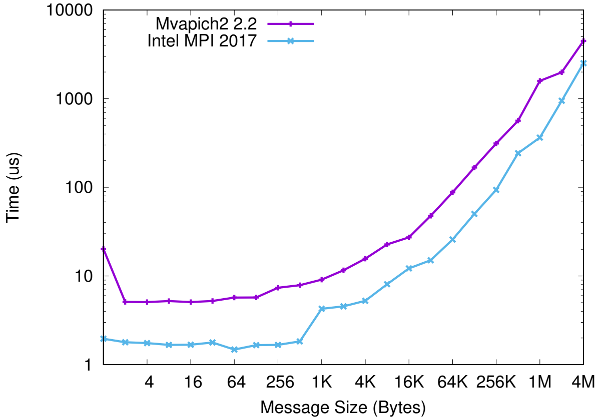
Intel MPI Benchmarks (IMB)

Details

- Originally know as Pallas MPI Benchmarks (PMB)
- Support Point-to-Point and Collective operations
- 1 program with lot of options for classical MPI functions (IMB-MPI1)
- Root changes after each iteration for collectives

Intel MPI Benchmarks (IMB)

Intel MPI vs MVAPICH2 using IMB Bcast with 256 cores



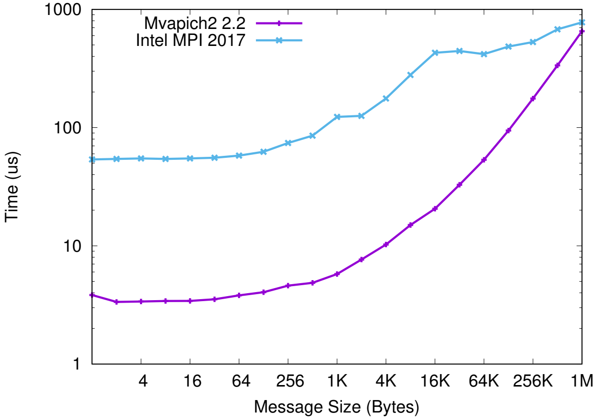
OSU Micro-Benchmarks (OMB)

Details

- Very simple to use
- Support Point-to-Point and Collective operations
- Multiples programs with simple options
- Keep the same root during all iterations + use barrier

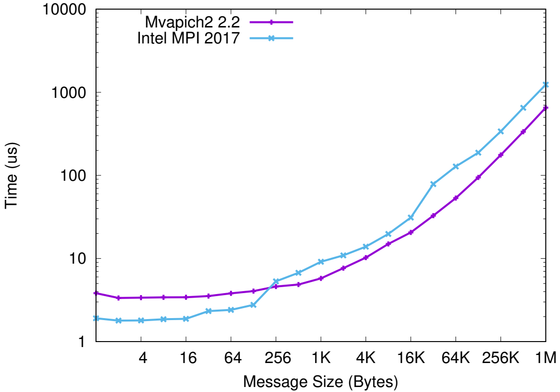
OSU Micro-Benchmarks (OMB)

Intel MPI vs MVAPICH2 using OMB Bcast with 256 cores



OSU Micro-Benchmarks (OMB)

Tuned Intel MPI vs MVAPICH2 using OMB Bcast with 256 cores



Benchmarks: What you need to know

To resume

- Don't trust them !
- They have different behaviors: so, KNOW your benchmark !
- Don't provide you necessarily the best results by default.
- Be sure that you tune things correctly if you want to compare two MPI libraries
- Collective tuning for a particular benchmark/application could be painful, we will see it later :)

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

To know what you need to tune first

Why MPI profiling is important ?

- To identify which MPI functions are used, you have two choices:
  - Look at the code
  - Profile your application
- Profiling provides you all the information regarding MPI communications (size, time spent, functions called etc...)
- Could be integrated in the MPI library (ex: Intel MPI)
- Lot of tools can help you to profile your application (TAU, Scalasca, IPM, mpiP ...)

How to profile ?

With Intel MPI at runtime

mpiexec -genv I_MPI_STATS=ipm I_MPI_STATS_FILE=myprofile.txt ....

Tools

- MPI Performance Snapshots (MPS)
- Intel Trace Analyzer and Collector (ITAC)

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Impact of the hostfile

Example of command:

mpirun -np 4 -hostfile host ./a.out

- Hostfile provides the list of nodes that will be used
- Depending on the MPI library, **the same hostfile** could lead to different results !

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

```
mpirun -np 4 -hostfile host ./sp-mz.C.4
```

node1
node2

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

node1
node2

Mvapich2

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

```
node1
node2
```

Mvapich2

- Default: 176 sec.

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

node1
node2

Mvapich2

- Default: 176 sec.
- Correct Hostfile: 176 sec.

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

node1
node2

Mvapich2

- Default: 176 sec.
- Correct Hostfile: 176 sec.
- + Process Placement: 19 sec.

Intel MPI

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

```
node1
node2
```

Mvapich2

- Default: 176 sec.
- Correct Hostfile: 176 sec.
- + Process Placement: 19 sec.

Intel MPI

- Default: 51 sec.

A Quick Performance Example

NAS SP-MZ on Stampede

2 nodes, 2 MPI tasks/node with 8 OpenMP threads

mpirun -np 4 -hostfile host ./sp-mz.C.4

node1
node2

Mvapich2

- Default: 176 sec.
- Correct Hostfile: 176 sec.
- \+ Process Placement: 19 sec.

Intel MPI

- Default: 51 sec.
- Correct Hostfile/Command: 19 sec.

How the MPI tasks are propagated?

mpirun -np 4 -hostfile host ./a.out on Stampede

node1
node2

Mvapich2	Open MPI	Intel MPI
Rank 0 on node1	Rank 0 on node1	Rank 0 on node1
Rank 1 on node2	Rank 1 on node1	Rank 1 on node1
Rank 2 on node1	Rank 2 on node2	Rank 2 on node1
Rank 3 on node2	Rank 3 on node2	Rank 3 on node1

How to set correctly your hostfile?

Mvapich2

node1:2
node2:2

Open MPI

node1 slots=2
node2 slots=2

Intel MPI

I_MPI_PERHOST=2
or
mpirun -perhost 2
or
mpirun -ppn 2

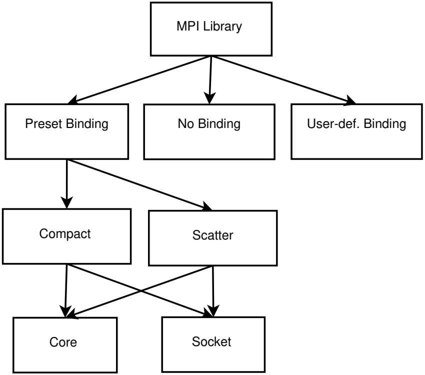
Result

Rank 0 on node1
Rank 1 on node1
Rank 2 on node2
Rank 3 on node2

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Generic Way to Map Processes



MVAPICH2

Compact-Core

Intel MPI

Scatter-Shared Socket or Core

Open MPI

Before 1.7.4: No Binding
After 1.7.4: Scatter-Core|Socket

Quick Examples with Intel MPI

Osu_latency results between cores

	Latency (us)		Description
Pair	0 byte	8 k-byte	
2-4	0.2	1.77	Same socket, shared L3, best performance
0-2	0.2	1.80	Same socket, shared L3, but core 0 handles interrupts
2-10	0.41	2.52	Different sockets, does not share L3
0-8	0.42	2.53	Different sockets, does not share L3, core 0 handles interrupts

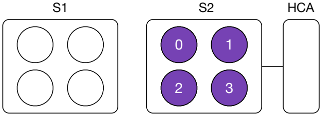
NAS SP-MZ (Do you remember it ?)

2 Nodes, 2 MPI Tasks/node with 8 OpenMP threads using MVAPICH2

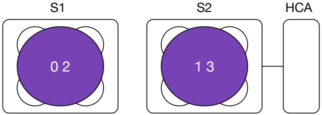
- Default Mapping: 176 seconds
- Optimal Mapping: 19 seconds

TACC

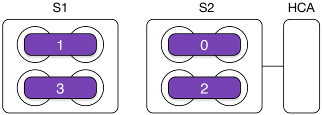
Default MPI library mappings of four MPI tasks (each with two OpenMP threads) to the two 4-core processors of a dual-socket node.



(a) MVAPICH2 2.1

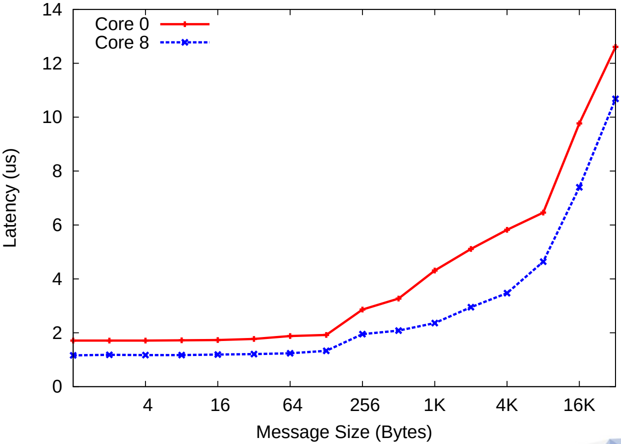


(b) Open MPI 1.8.8

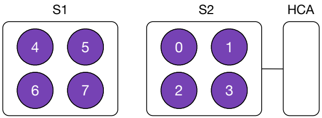


(c) Intel MPI 5.0.2

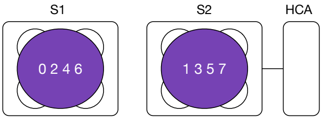
Impact of Mapping (MVAPICH2 / osu_latency)



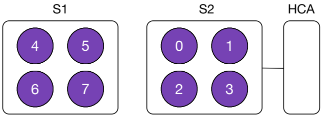
Default MPI library mappings of eight pure MPI tasks to the two 4-core processors of a dual-socket node.



(a) MVAPICH2 2.1



(b) Open MPI 1.8.8



(c) Intel MPI 5.0.2

Setting the Mapping

Intel MPI

- I_MPI_PIN_PROCESSOR_LIST: Define a processor subset and mapping rules for MPI processes pinning to separate processors of this subset
- I_MPI_PIN_DOMAIN (For Hybrid code)

MVAPICH2

- MV2_CPU_BINDING_POLICY=*bunch*|*scatter*
- MV2_CPU_BINDING_LEVEL=*core*|*socket*|*numanode*
- Manual: MV2_CPU_MAPPING=0:8:9-15:1-7

Reporting the Mapping

Most of the MPI libraries provide a mechanism to report the mapping

- Intel MPI: -print-rank-map
- MVAPICH2: MV2_SHOW_CPU_BINDING=1
- OpenMPI: --report-bindings

Example

```
c421-502$ mpirun_rsh -np 2 -hostfile hosts MV2_CPU_MAPPING=2-4:10-12 MV2_SHOW_CPU_BINDING=1 ./osu_latency
-------------CPU AFFINITY-------------
RANK:0 CPU_SET: 2 3 4

RANK:1 CPU_SET: 10 11 12
```

Basic Level

Benchmarking Needed to evaluate the performance and to tune a MPI library, but it is important to know the behavior of the benchmark to do a correct evaluation

Basic Level

Benchmarking
: Needed to evaluate the performance and to tune a MPI library, but it is important to know the behavior of the benchmark to do a correct evaluation

Profiling
: Useful to know where most of the MPI time is spent and to know what you need to tune

Basic Level

Benchmarking	Needed to evaluate the performance and to tune a MPI library, but it is important to know the behavior of the benchmark to do a correct evaluation
Profiling	Useful to know where most of the MPI time is spent and to know what you need to tune
Hostfile	Do it correctly (or use the right command) or your performance will be bad

Basic Level

Benchmarking	Needed to evaluate the performance and to tune a MPI library, but it is important to know the behavior of the benchmark to do a correct evaluation
Profiling	Useful to know where most of the MPI time is spent and to know what you need to tune
Hostfile	Do it correctly (or use the right command) or your performance will be bad
Mapping	Important specially for hybrid code.

Basic Level

Benchmarking Needed to evaluate the performance and to tune a MPI library, but it is important to know the behavior of the benchmark to do a correct evaluation

Profiling Useful to know where most of the MPI time is spent and to know what you need to tune

Hostfile Do it correctly (or use the right command) or your performance will be bad

Mapping Important specially for hybrid code.

These 4 'basic' things are easy to do and can really improve the performance of your code !

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Eager/Rendezvous protocol I

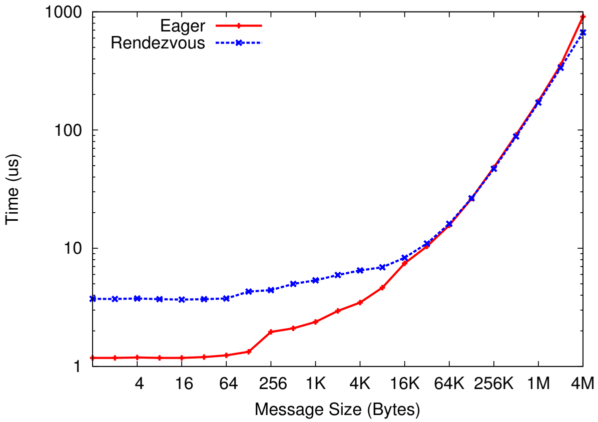
- There are multiple different protocols for sending messages.
- We will only focus here on eager / rendezvous protocol.
- Switch point between these two protocols can be called *threshold*, *eager threshold* or *eager limit*.
- It is an implementation technique, it is not part of the MPI standard

Eager
: The sender process *eagerly* sends the entire message to the receiver. Typically used for 'short' messages.

Rendezvous
: Based on 'Request To Send' / 'Clear To Send' (RTS/CTS) techniques. Typically used for 'long' messages.

Eager vs Rendezvous

Osu_latency and MVAPICH2



Eager/Rendezvous: Pro/Con

Eager

Pro: Reduces synchronization delays
Best for latency

Con: Significant buffering may be required to provide space for messages
Can cause memory exhaustion / program termination when receive process buffer is exceeded

Rendezvous

Pro: Scalable
Robustness by preventing memory exhaustion

Con: Delay due to handshaking between sender and receiver

Setting the eager threshold

Intel MPI

I_MPI_EAGER_THRESHOLD= $<$ *nbytes* $>$

MVAPICH2

MV2_IBA_EAGER_THRESHOLD= $<$ *nbytes* $>$

Open MPI

--mca btl_openib_eager_limit $<$ *nbytes* $>$
--mca btl_openib_rndv_eager_limit $<$ *nbytes* $>$

The threshold could be, by default, platform specific (MVAPICH2, OpenMPI) or identical for all platforms (IMPI)

Plan

- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Intra-node optimization

Different mechanisms exist:

With the number of cores per node increasing in modern clusters, an efficient implementation of intra-node communications is critical for application performance.

We will introduce here two different mechanisms:

- Shared Memory
- Kernel Assisted

Shared Memory

- Used by all MPI implementations
- Double-copy implementation involves a shared buffer space used by local processes to exchange messages.
- Best approach for small messages
- Not ideal for large messages (tie down CPU, cache pollution)

Kernel Assisted I

- Single copy mechanism
- Preferred approach for medium or large messages
- You need to use:
  - Kernel module: LiMIC or KNEM
  - Kernel feature: CMA

Kernel Assisted II

CMA

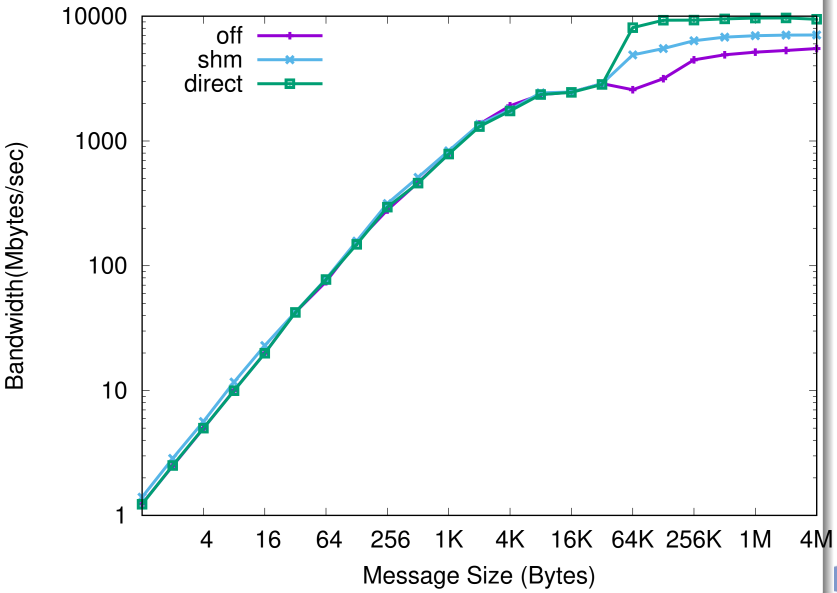
- **C**ross **M**emory **A**ttach
- Introduced with Linux kernel 3.2 and has been back-ported to some Linux distribution
- Available on Stampede, supported by Intel MPI (since 5.0u2), MVAPICH2, OpenMPI etc...
- CMA will be enable automatically for large messages since 5.1u2.

To resume

Which one is the best ?

- For short messages (eager protocol): Shared Memory is better
- For large messages (rendezvous protocol): Kernel assisted is better
- I_MPI_SHM_LMT= shm | direct | off

IMB Pingpong on Compute Node, Intra-socket



NAS results on large memory node with 32 cores using Intel MPI

Benchmark	Class	Shared (s)	CMA (s)	Speedup
CG	C	10.29	9.66	+6.12%
EP	C	3.89	3.88	+0%
FT	C	16.04	12.07	+24.75%
IS	C	1.37	1.04	+24.08%
CG	D	381.95	382.03	-0.02%
EP	D	62.07	62.08	+0.8%
FT	D	365.84	289.32	+20.91%
IS	D	26.1	20.92	+19.8%

Plan

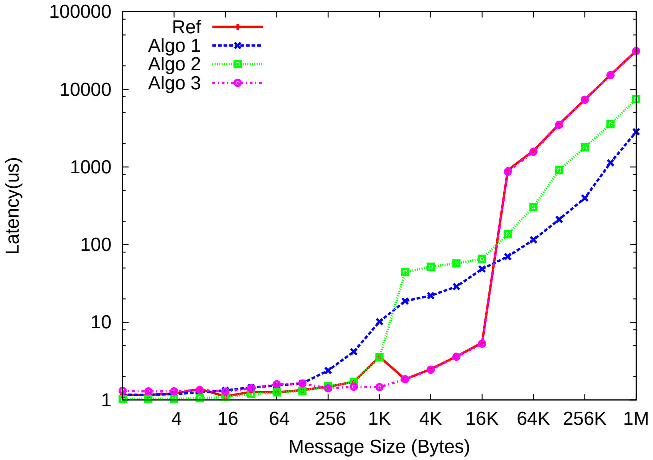
- Basic Tuning
  - The Choice of the Benchmark
  - Profiling
  - Hostfile
  - Process Placement
  - To conclude
- Intermediate Tuning
  - Inter-node Point-to-Point Optimization
  - Intra-node Point-to-Point Optimization
  - Collective Tuning
  - To conclude
- Conclusion

Collective tuning

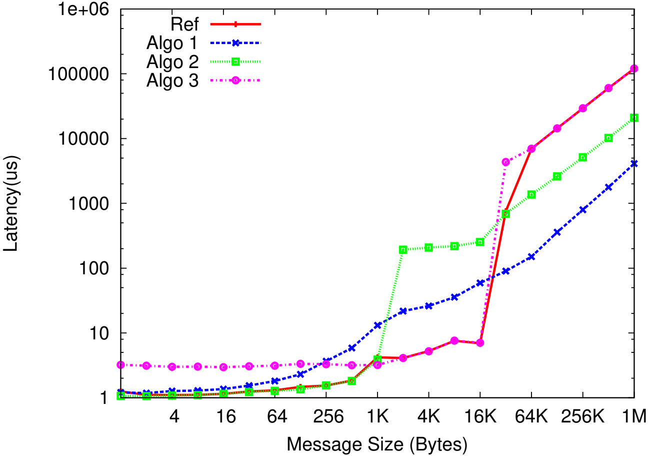
Collective communications

- Used when a communication involves more than 2 MPI tasks
- Behind each collective, there are many algorithms (Binomial Tree, Recursive Doubling, Ring Exchange etc...)
- The choice of the algorithm depends on many parameters (number of cores, network speed, architecture, message size etc...)
- Default tuning is not necessarily the best one for you.
- MPI libraries provide mechanisms to select which algorithms need to be used.

osu_gather with Intel MPI 256 cores (I_MPI_ADJUST_GATHER)



osu_gather with Intel MPI 1024 cores (I_MPI_ADJUST_GATHER)



Intermediate Level

Inter-node
: You can play with the eager threshold to improve the performance of your communication, but it will cost you more memory !

Intermediate Level

Inter-node
: You can play with the eager threshold to improve the performance of your communication, but it will cost you more memory !

Intra-node
: Different mechanisms exist to improve the performance of large messages.

Intermediate Level

Inter-node
: You can play with the eager threshold to improve the performance of your communication, but it will cost you more memory !

Intra-node
: Different mechanisms exist to improve the performance of large messages.

Collectives
: There are different algorithms for each collectives, if you see that your code is spending a lot of time inside one of them, try to see if changing the algorithm could help.

Intermediate Level

Inter-node: You can play with the eager threshold to improve the performance of your communication, but it will cost you more memory !

Intra-node: Different mechanisms exist to improve the performance of large messages.

Collectives: There are different algorithms for each collectives, if you see that your code is spending a lot of time inside one of them, try to see if changing the algorithm could help.

Inter-node and collective tuning can be time consuming but intra-node optimization could be simple and provides very good results !

Plan

Conclusion

Every good things has an end...

We saw today a lot of things, don't panic if you don't remember of everything. Keep in mind the following things:

- Read the documentation :)
- MPI libraries behave differently
- Mechanisms exists to improve easily the performance of your code (Process Mappping, CMA...)
- Mechanisms exists also to reduce the memory footprint of your MPI library
- Don't be afraid to ask for help