23 tips for performance tuning with the Intel® MPI Library

October 11th, 2011

Legal Disclaimer

Optimization Notice

Optimization Notice

Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the "Intel® Compiler User and Reference Guides" under "Compiler Options." Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.

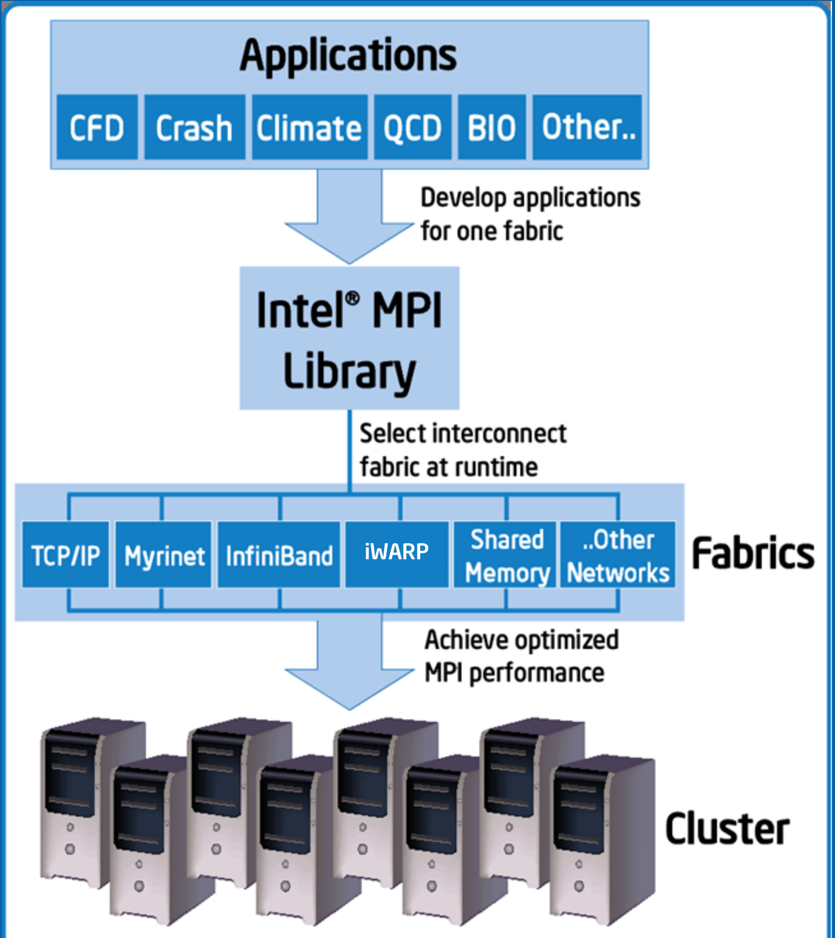
Intel® compilers, associated libraries and associated development tools may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.

While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.

Notice revision #20101101

Intel® MPI Library 4.0 Update 1

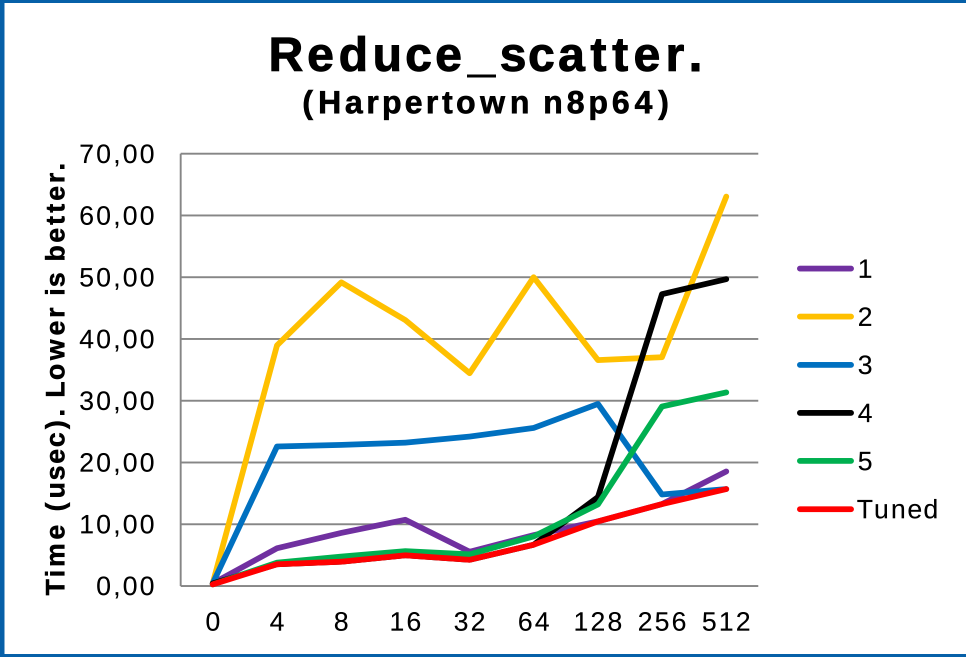
- High performance MPI-2.1 implementation
- Linux* and Windows* support
- Interconnect independence
- Smart fabric selection and performance optimization
- Multi-rail and failover support
- Thread-safety and fault tolerance
- Free Runtime Environment
- Close integration with the Intel and 3rd party development tools
- Internet based licensing and technical support



Intel® MPI library eliminates the need to develop, maintain, and test applications running on multiple fabrics.

The Holy Grail of higher Application Performance

- Why tuning?
  - Account for specifics of the target cluster hardware, application communication topology, and computational intensity



- Tuning areas:
  - Interconnect fabrics
  - Process placement and pinning
  - Point-to-point communication
  - Collective algorithms

- Tuning technique:
  - Ensure the cluster is sane and the application is built at high optimization level
  - Automatically tune Intel® MPI Library for the given cluster
  - Learn your application communication characteristics thru stats gathering
  - Use MPI run-time options and environment variables for application tuning

Sample: Intel® MPI and a popular Automotive Industry Application

- Benchmark profile (I_MPI_STATS):
  - Intensive Bcast and Reduce operations
  - Optimal MPI_Reduce algorithm boosts application performance

- Fine-tuned configuration:

  I_MPI_FABRICS=shm:dapl

  I_MPI_ADJUST_REDUCE=2

  I_MPI_DAPL_SCALABLE_PROGRESS=1

- Performance benefit:
  - Up to 140% on 512 MPI ranks

23 tuning tips

Prerequisites

1. Use Intel MPI automatic tuning utility
2. Build application for highest performance
3. Make sure your cluster is properly configured

Basics

4. Use best available communication fabric
5. Use multi-rail capability
6. Use connectionless communication
7. Disable fallback device for benchmarking
8. Select proper process layout
9. Manage process pinning
10. Enable MPI/OpenMP* mixed mode for threaded apps

Advanced

11. Disable dynamic connection mode for small jobs
12. Use scalable RDMA progress for large jobs
13. Apply wait mode to oversubscribed jobs
14. Use Intel MPI lightweight statistics
15. Adjust eager/rendezvous protocol threshold
16. Bypass shared memory for intranode transfers
17. Choose the best collective algorithms

Black belt

18. Bypass cache for intranode transfers
19. Tune message passing progress engine
20. Disable RDMA translation cache
21. Reduce size of pre-reserved memory for RDMA/RDSSM communication device
22. Allow dynamic enlargement of pre-reserved memory for RDMA/RDSSM
23. Tune TCP/IP connection

1. Use Intel® MPI automatic tuning utility

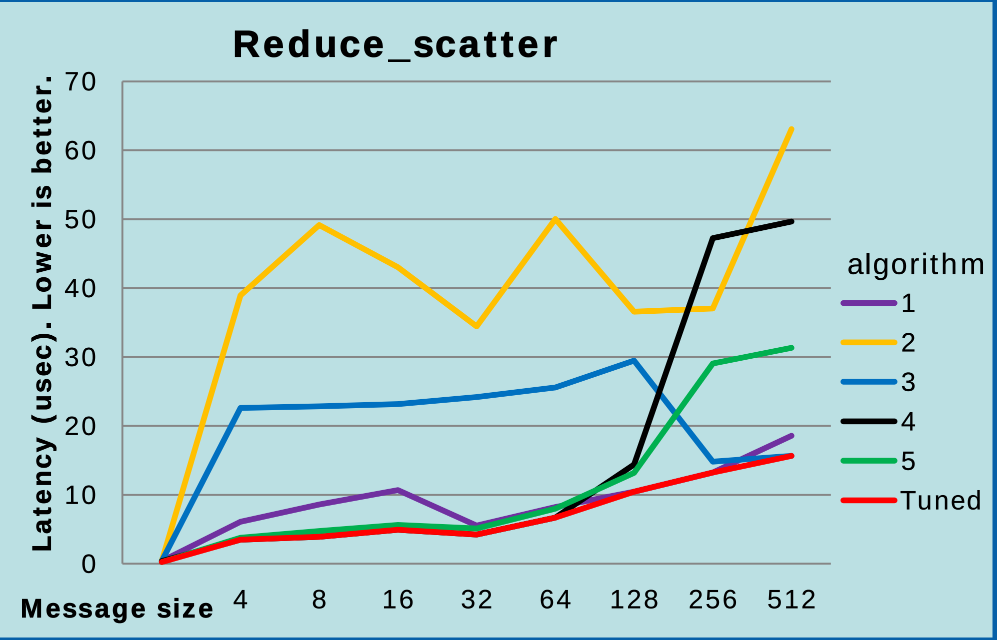
- Find optimal values for library tuning knobs on the particular cluster or application environment with the automated tuning utility

Cluster-specific tune

- Run it once after installation and each time after cluster configuration change
- Best configuration is recorded for each combination of communication device, number of nodes, MPI ranks and process distribution model

```
# Collect configuration values:
$ mpitune

# Reuse recorded values:
$ mpiexec –tune –n 32 ./your_app
```



Example of tuned values:
-genv I_MPI_ADJUST_REDUCE_SCATTER '5:0-0;4:0-87;1:87-345;3:345-4194304'

2. Make sure your cluster is properly configured

- Install the latest Intel® MPI Library. Get free evaluation license from software.intel.com
- Check Intel MPI Library installation (see Getting Started), especially proper selection of the desired fast fabrics (IB, 10GigE, GigE, etc.)
- Use Intel® Cluster Checker for cluster validation on ICR platforms. Find Cluster Checker at http://softwareproducts.intel.com/ILC
- Alternatively, check intended fast fabrics for availability and expected performance across as many nodes as possible (ideally, all)

```
$ mpirun –r ssh –RDMA -n <# of processes> -env I_MPI_DEBUG 5 IMB-MPI1
```

- When using a job management system, always do comparison runs within the same job session. Varying node subsets may lead to performance anomalies
- Capture platform details and keep all logs for future reference

3. Build application for highest performance

- Use Intel® C++ and Fortran compilers that offer highest performance on the new Intel platforms
  - Intel compiler and “-xsse4.2 -O3 -no-prec-div” options are currently recommended for latest Intel® Core™2 processors (like Intel® XEON™ 5600)
- Select proper MPI compiler driver scripts to build application, depending on the underlying compiler
  - For example, use ‘mpiicc’ to compile C Language applications with Intel® C/C++ Compiler for Linux
- If you have to use native compilers directly, use MPI compiler scripts with the –show option to learn the full compiler invocation line

```
$ mpiicc   -xsse4.1 -O3 -no-prec-div func.c –c –o func.o
$ mpiifort –xsse4.1 -O3 -no-prec-div main.f func.o -o your_app
```

4. Use best available communication fabric

- Use default fabric selection if you can.
  Check the result thru I_MPI_DEBUG set to 2

I_MPI_DEVICE	I_MPI_FABRICS	Description
sock	tcp	TCP/IP-capable network fabrics, such as Ethernet and InfiniBand* (through IPoIB*)
shm	shm	Shared-memory only
ssm	shm:tcp	Shared-memory + TCP/IP
rdma	dapl	DAPL–capable network fabrics, such as InfiniBand*, iWarp*, Dolphin*, and XPMEM* (through DAPL*)
rdssm	shm:dapl	Shared-memory + DAPL + sockets
	ofa	OFA-capable network fabric including InfiniBand* (through OFED* verbs)
	tmi	TMI-capable network fabrics including Qlogic*, Myrinet*, (through Tag Matching Interface)

Select network interface for socket communications IP over IB:

I_MPI_TCP_NETMASK=ib0 for IP over IB

I_MPI_TCP_NETMASK=192.169.0.0 for particular subnet

```
$ mpirun –genv I_MPI_DEBUG 2 –genv I_MPI_FABRICS dapl -n <number of
processes> ./your_app
```

5. Disable fallback for benchmarking

- Intel MPI library falls back from ‘dapl’ or ‘shm:dapl’ fabric to ‘shm’ and/or ‘tcp’ if DAPL provider initialization failed.
- Detect real failure of the fabrics setting I_MPI_FALLBACK to ‘disable’.
- Same effect with command line option:

```
$ mpirun –DAPL –n <number of processes> ./your_app
```

Capital letters of ‘DAPL’ means that I_MPI_FALLBACK will be set to ‘disable’
Shared memory will not be used in this example

6. Use multi-rail capability

- If each node of your cluster is equipped with several adapters or multi-port adapters you can get higher bandwidth and lower latency.

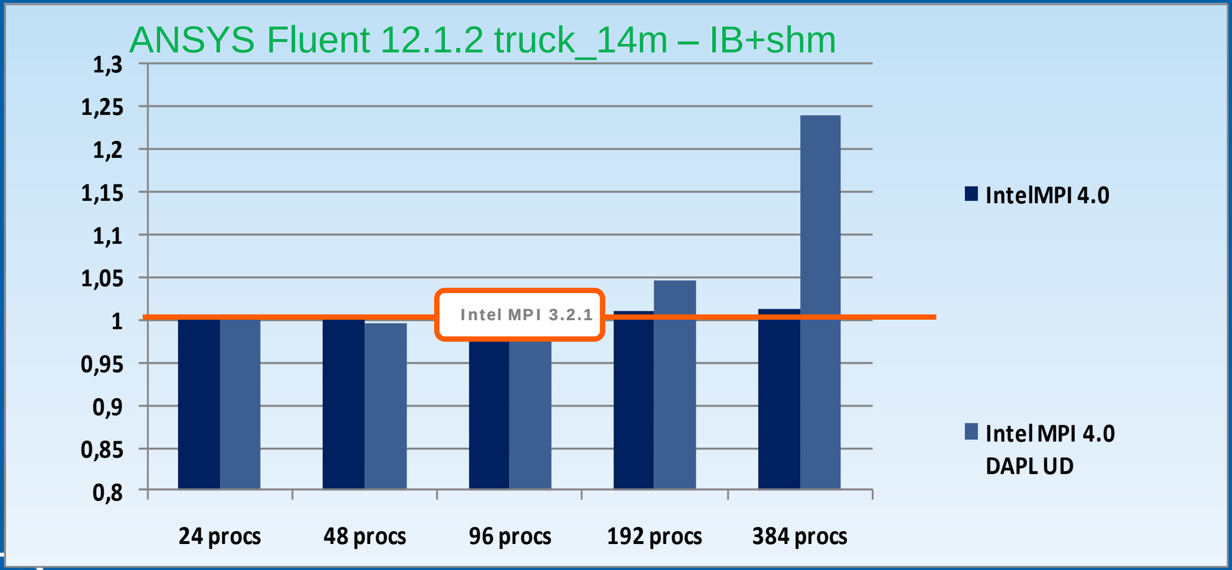
- Use the following settings:

```
$ export I_MPI_FABRICS=shm:ofa
$ export I_MPI_OFA_NUM_ADAPTERS=<n>  e.g. 2 (1 by default)
$ export I_MPI_OFA_NUM_PORTS=<n>     e.g. 1 (1 by default)
```

7. Use connectionless communication

- Connectionless feature works for DAPL fabrics only
- Works with OFED 1.4.2 and 2.0.24 or higher
- Provides better scalability
- Significantly reduces memory requirements

```
$ export I_MPI_FABRICS=dapl    (or shm:dapl)
$ export I_MPI_DAPL_UD=enable
```



Benchmark overview

- Intel® MPI Library 4.0 options
  - -genv I_MPI_ADJUST_REDUCE 2
  - -genv I_MPI_ADJUST_BCAST 0
  - -genv I_MPI_DAPL_SCALABLE_PROGRESS 1
- Fluent benchmark
  - Truck_14m

Hardware Configuration:

- Interconnect: InfiniBand, ConnectX adapters; QDR
- CPU: 2.93GHz B0-step Westmere Dual processor (6 cores per processor)
- RAM: 24Gb per system, 1333MHz (0.8ns) DDR3

8. Select proper process layout

- Default process layout is “group round-robin”

- Set I_MPI_PERHOST variable to override the default process layout:
  - I_MPI_PERHOST=1 makes round-robin distribution
  - I_MPI_PERHOST=all maps processes to all logical CPUs on a node
  - I_MPI_PERHOST=allcores maps processes to all physical CPUs on a node

- Or use an mpiexec options:

  -perhost <#> - "processes per host“, group round-robin distribution placing # of processes at every host

  -rr - "round-robin" distribution, same as ‘-perhost 1’

  -grr <#> - "group round-robin", same as ‘-perhost #’

  -ppn <#> - "processes per node", same as ‘-perhost #’

```
$ mpirun –perhost 2 -n <number of processes> ./your_app
```

9. Use proper process pinning

- Use I_MPI_PIN_PROCESSOR_LIST to define custom map of MPI processes to CPU cores pinning. Find pinning map optimal to your application.
- Use the ‘cpuinfo’ utility supplied with Intel MPI Library to see the processor topology, including inter-core cache sharing information.
- To pin the processes to the CPU0 and CPU3, use sequential identifiers starting from zero

```
$ mpirun –genv I_MPI_PIN_PROCESSOR_LIST=0,3 -n <procs>./your_app
```

- To place consecutive MPI processes to cores sharing L2 cache and occupy different physical packages for consecutive pairs of processes, use “grain=cache2,shift=sock”:

```
$ mpirun –genv I_MPI_PIN_PROCESSOR_LIST=‘grain=cache2,shift=sock’
    -n <procs>./your_app
```

10. Enable MPI / OpenMP* mixed mode for threaded apps

- Check command line for application building
  - Use the thread safe version of the Intel® MPI Library (**-mt_mpi** option)
  - Use the libraries with SMP parallelization (i.e. parallel MKL)
  - Use **–openmp** compiler option to enable OpenMP* directives

```
$ mpiicc –openmp –mt_mpi -o ./your_app
```

- Set application execution environment for hybrid applications
  - Set **OMP_NUM_THREADS** to threads number
  - Use **–perhost** option to control process pinning

- For POSIX threaded apps turn off pinning by setting I_MPI_PIN to `disable` to inherit default shell affinity mask.

```
$ export I_MPI_DAPL_SCALABLE_PROGRESS=1
$ export I_MPI_FABRICS=shm:dapl
$ export KMP_AFFINITY=compact

$ mpirun -perhost 4 -n <N> ./wrf
```

11. Disable dynamic connection mode

- I_MPI_DYNAMIC_CONNECTION set to ‘0’ disables dynamic connections mode (on-demand connection establishment) of Intel MPI Library. It’s set by default to ‘0’ for <64 processes

12. Use scalable DAPL progress for large jobs

- Set I_MPI_DAPL_SCALABLE_PROGRESS variable to 1 to enable scalable algorithm for DAPL read progress engine. It offers performance advantage for large (>64) numbers of processes.

13. Apply wait mode to oversubscribed jobs

- Set I_MPI_WAIT_MODE to ‘enable’ to try wait mode of the progress engine. The processes that waits for receiving messages without polling of the fabric(s) can save CPU time.

14. Use Intel MPI lightweight statistics

- Set I_MPI_STATS set to non-zero integer value to gather MPI communication statistics
- See file 'stats.txt' or any other specified by the I_MPI_STATS_FILE.
- Manipulate with I_MPI_STATS_SCOPE to increase effectiveness of the analysis.
- Reasonable values to adjust collective operations algorithm are I_MPI_STATS=3 I_MPI_STATS_SCOPE=coll

```
~~~~ Process 0 of 256 on node C-21-23

Data Transfers
Src   --> Dst   Amount(MB)    Transfers
-----------------------------------------
000 --> 000    0.000000e+00   0
000 --> 001    1.548767e-03   60
000 --> 002    1.625061e-03   60
000 --> 003    0.000000e+00   0
000 --> 004    1.777649e-03   60
…
=========================================
Totals    3.918986e+03  1209

Communication Activity
Operation Volume(MB)     Calls
-----------------------------------------
P2P
Csend     9.147644e-02   1160
Send      3.918895e+03   49
Collectives
Barrier           0.000000e+00   12
Bcast             3.051758e-05    6
Reduce          3.433228e-05  6
Allgather       2.288818e-04  30
Allreduce       4.108429e-03  97
```

```
$ mpiexec –genv I_MPI_STATS 3 –I_MPI_STATS_SCOPE coll …
```

15. Adjust eager/rendezvous protocol threshold

- Two communication protocols:
  - “Eager” sends data immediately regardless of receive request availability.
  - “Rendezvous” notices receiving site on data pending and transfers when receive request is set.

- Two protocol levels:
  - high-level, common for all communication devices
  - low-level RDMA protocol.

- One environment variable:
  - I_MPI_EAGER_THRESHOLD controls high level protocol switchover point. Short message are sent using the eager protocol, larger are sent by using the more memory efficient rendezvous protocol.

16. Bypass shmem for intranode communication

- Set I_MPI_SHM_BYPASS* to ‘enable’ to turn on RDMA data exchange within single node that may outperform regular shared memory exchange. This is normally happens for large (350kb+) messages.

- Messages shorter than or equal in size to the threshold value of the I_MPI_INTRANODE_EAGER_THRESHOLD are transferred using shared memory, larger – through network fabric layer. Try to increase this threshold, which default value is equal to I_MPI_EAGER_THRESHOLD (~256kb).

*This mode is available for dapl and tcp fabrics in MPI 4.0

17. Choose the best collective algorithms

- Use one of the I_MPI_ADJUST_<opname> knobs to change the algorithm for example:
  - I_MPI_ADJUST_ALLREDUCE controls MPI_Allreduce algorithm, which could be (1) recursive doubling algorithm, (2) Rabenseifner's algorithm, (3) Reduce + Bcast, (4) Topology aware Reduce + Bcast algorithm, (5) Binomial gather + scatter algorithm, (6) topology aware binominal gather + scatter algorithm and (7) Ring algorithm.
- Section “3.5 Collective Operation Control” of Intel MPI Reference Manual defines the full set of variables and algorithm identifiers.
- Recommendations:
  - Focus on the most critical collective operations (see stats output).
  - Run Intel MPI Benchmarks selecting various algorithms to find out the right protocol switchover points for hot collective operations.

```
$ mpirun -genv I_MPI_ADJUST_REDUCE 4 -n 256 ./fluent
```

An example:
-genv I_MPI_ADJUST_ALLGATHER '1:4-11;4:11-15;1:15-27;4:27-31;1:31-32;2:32-51;3:51-5988;4:5988-13320'

18. Bypass cache for intranode communication

- Control a message transfer algorithm for shm device: generic copying or cache bypass (using non-temporal store)

- Each case have own threshold pair (read/write in shm queue) and can be tuned by I_MPI_SHM_CACHE_BYPASS_THRESHOLDS (see Reference Manual for details). Some default thresholds are set to half of L2. One can start to tune beginning L1 cache size

```
$ export I_MPI_SHM_CACHE_BYPASS_THRESHOLDS=16384,16384,-1,16384,-1,16384
$ mpiexec –n 2 –genv I_MPI_FABRICS shm IMB-MPI1 PingPong
```

19. Tune message passing progress engine

- Try to increase I_MPI_SPIN_COUNT value – number of times communication progress engine spins waiting for a message or connection request before it yields to the OS. Default value is 1 when more than one process runs per processor/core or for shm on IA-64, otherwise – 250.

- An application that actively uses MPI Put/Get operations may benefit from decreasing values of I_MPI_FAIR_READ_SPIN_COUNT (default is 100) and I_MPI_FAIR_CONN_SPIN_COUNT (default is 1000) those control how often inactive channels are pulled, otherwise try to increase the values.

20. Disable memory registration cache

- Intel® MPI Library enhances message-passing performance on DAPL*-based interconnects by maintaining a cache of virtual-to-physical address translations in the MPI DAPL* data transfer path.

- The cache substantially increases performance but may lead to correctness issues in certain rare situations. In this case translation cache could be disabled setting I_MPI_DAPL_TRANSLATION_CACHE variable to value 0.

- An application actively allocating, sending and deallocating memory regions may benefit from disabled translation cache.

21. Reduce size of pre-reserved memory for DAPL communication device

- Large-scale applications may experience memory resource pressure due to a big number of pre-allocated buffers pinned to physical memory pages.
- Use I_MPI_DAPL_BUFFER_NUM variable to change the number of buffers for each pair in a process group. The default value is 16.
- Decreasing I_MPI_DAPL_BUFFER_NUM one can save memory and avoid application memory swapping. Another may benefit from higher number of buffers when intensively exchanging small messages.

22. Allow dynamic enlargement of pre-reserved memory for DAPL path

- Set I_MPI_DAPL_BUFFER_ENLARGEMENT variable to 1 to enable two-phase enlargement of DAPL buffers.

- If enabled, small size internal pre-registered DAPL buffers are allocated and enlarged later if data size exceeds the threshold defined by I_MPI_DAPL_BUFFER_ENLARGEMENT_THRESHOLD (default value 580 bytes)

23. Tune TCP/IP connection

- On most Linux distributions TCP/IP stack tuned for 100 Mb/s networks
- Settings in /etc/modprobe.conf for Intel 1000 NIC:
  - alias eth0 e1000
  - options e1000 InterruptThrottleRate=0 TxIntDelay=0 TxDescriptors=512 RxDescriptors=512
- Settings common for any GigE:
  - $ ifconfig eth0 txqueuelen 5000
- TCP/IP stack tuning
  - Edit the /etc/sysctl.conf file
    - net.ipv4.tcp_sack = 0
    - net.ipv4.tcp_fack = 0
    - net.core.netdev_max_backlog=3000
    - net.core.rmem_max = 16777216
    - net.core.rmem_default = 4194394
    - net.core.wmem_max = 16777216
    - net.core.wmem_default = 2097197
    - net.ipv4.tcp_rmem = 4096 4194394 8388608
    - net.ipv4.tcp_wmem = 4096 2097197 8388608
    - net.ipv4.tcp_window_scaling = 1
    - vm.min_free_kbytes=65536
    - net.ipv4.tcp_moderate_rcvbuf=0
  - In /etc/rc.local we have:
    echo 5 > /proc/sys/net/ipv4/route/gc_timeout
    echo 5 > /proc/sys/net/ipv4/route/gc_min_interval
    echo 32 > /proc/sys/net/ipv4/route/gc_thresh
    echo 2 > /proc/sys/net/ipv4/route/gc_elasticity

Ready Application Settings

- Use application-specific configuration files provided with Intel MPI Library package, that contain optimized tuning parameters

- Some popular application settings:

  LS-DYNA http://www.ls-dyna.com/

I_MPI_FAIR_CONN_SPIN_COUNT	= 2147483647
I_MPI_FAIR_READ_SPIN_COUNT	= 2147483647

  Fluent http://www.fluent.com

  I_MPI_ADJUST_COLLECTIVES = bcast:0;reduce:2

  HPCC http://icl.cs.utk.edu/hpcc/

I_MPI_EAGER_THRESHOLD	= 128000
I_MPI_FALLBACK_DEVICE	= disable
I_MPI_RDMA_RECV_QUEUE_SIZE	= 0
I_MPI_FAIR_READ_SPIN_COUNT	= 10000
RDMA_DEFAULT_MAX_WQE	= 500
RDMA_READ_RESERVE	= 20

See /opt/intel/mpi/4.0/etc*/*.conf for more examples