MPI Performance

Intel® MPI Library
Intel® Trace Analyzer and Collector
Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. https://software.intel.com/en-us/articles/optimization-notice

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. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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

- Distributed Performance with Intel® MPI Library
- Tuning MPI Application Performance with Intel® Trace Analyzer and Collector
- Related Tools – Intel® MPI Benchmarks
- Summary and Resources
Intel® MPI Library
Intel® MPI Library Overview

- Optimized MPI application performance
  - Support for all Intel® Xeon® and Intel® Xeon Phi™ processors
  - Optimized collectives with topology and architecture awareness

- Lower-latency and multi-vendor interoperability
  - Industry leading latency
  - Performance optimized support for the fabric capabilities through OpenFabrics® (OFI) / libfabric

- Sustainable scalability up to 340K cores
  - Efficient path by relying on libfabric
  - New: Faster startup and finalization

- More robust MPI applications
  - Seamless interoperability with Intel® Trace Analyzer and Collector

- Conditional Numerical Reproducibility
  - I_MPI_CBWR to control reproducible results across topologies and hardware
Streamlined product setup
- Install as root, or as standard user
- Environment variable script mpivars.(c)sh sets paths

Compilation scripts to handle details
- One set to use Intel compilers, one set for user-specified compilers

Environment variables for runtime control
- I_MPI_* variables control many factors at runtime
  - Process pinning, collective algorithms, device protocols, and more
Compiling MPI Programs

- Compilation scripts automatically passes necessary libraries and options to underlying compiler
  - `mpiifort`, `mpiicpc`, and `mpiicc` use the Intel compiler by default
  - `mpif77`, `mpicxx`, `mpicc`, and others use GNU compiler by default

- Multiple ways to specify underlying compiler
  - `I_MPI_F77`, `I_MPI_CXX`, etc. environment variables
  - `-f77`, `-cc`, etc. command line options
  - Useful for makefiles portable between MPI implementations

- All compilers are found via PATH
MPI Launcher

- Robust launch command
  
  \[
  \text{mpirun} \ <\mpi\ \text{args}>\ \text{executable}\ <\text{program\ args}>
  \]

- Options available for:
  
  - Rank distribution and pinning
  - Fabric selection and control
  - Environment propagation
  - And more
Process Placement

- **Layout Across Nodes**
  - Default placement puts one rank per core on each node
  - Use –ppn to control processes per node
  - Use a machinefile to define ranks on each node individually
  - Use arguments sets or configuration files for precise control for complex jobs

- **Pinning on Node**
  - Can pin to single or multiple cores
  - Multiple options for automatic distribution based on resources such as socket, shared cache level, NUMA arrangement
  - See documentation for details:
Fabric Control via libfabric

- `I_MPI_OFI_PROVIDER` chooses provider (select based on interconnect hardware):
  - Default is normally fine
  - `tcp` – Ethernet
  - `psm2` – Intel® Omni-Path Architecture
  - `mlx` – InfiniBand* (requires at least Intel® MPI Library 2019 Update 5 and UCX 1.4)
Conditional Numerical Reproducibility

- **I_MPI_CBWR**
  - 0 (default) – no reproducibility controls, utilize all optimizations
  - 1 (weak) – disable topology aware optimizations, reproducible across different rank placements/topologies
  - 2 (strict) – disables topology aware optimizations and hardware optimizations, reproducible across hardware and topology

- **MPI_Comm_dup_with_info**
  - “I_MPI_CBWR”=“yes”, sets strict mode for communicator
Automatic Tuning via Autotuner

- Tuning happens behind the scenes during application run
- Tuning is per communicator

To tune:
  - `I_MPI_TUNING_MODE=auto`
  - `I_MPI_TUNING_BIN_DUMP=<tuning file>` (optional)

To use tuning results:
  - `I_MPI_TUNING_BIN=<tuning file>`

Additional options for more control, see https://software.intel.com/en-us/mpi-developer-reference-linux-autotuning
Debugging MPI Applications

- **GDB***
  - `mpirun <mpi options> -gdb <application and options>`
  - `mpirun -n <nranks> -gdba <mpirun pid>`

- **Allinea* DDT***
  - `ddt mpirun ...`

  - Set via `-gtool` option, `-gtoolfile` option, or `I_MPI_GTOOL`
  - `<prepend>:<rank set>[=launch mode][@arch]`
Intel® Trace Analyzer and Collector

Event-based Tracing for Distributed Applications
Intel® Trace Analyzer and Collector Overview

- Intel® Trace Analyzer and Collector helps the developer:
  - Visualize and understand parallel application behavior
  - Evaluate profiling and identify load balancing
  - Identify communication hotspots

- Features
  - Event-based approach
  - Low overhead
  - Excellent scalability
  - Powerful aggregation and filtering functions
  - Performance Assistance and Imbalance Tuning
Strengths of Event-based Tracing

- **Predict**: Detailed MPI program behavior
- **Record**: Exact sequence of program states – keep timing consistent
- **Collect**: Collect information about exchange of messages: at what times and in which order

An event-based approach is able to detect temporal dependencies!
Summary page shows computation vs. communication breakdown

Is your application MPI-bound?

Resource usage

Largest MPI consumers

Next Steps
Views and Charts

- Helps navigate the trace data
- A View can show several Charts
- All Charts in a View are linked to a single:
  - time-span
  - set of threads
  - set of functions
- All Charts follow changes to View (e.g. zooming)
Get detailed impression of program structure

Display functions, messages, and collective operations for each rank/thread along time-axis

Retrieval of detailed event information
Quantitative Timeline

Get impression on parallelism and load balance
Show for every function how many threads/ranks are currently executing it
Flat Function Profile

Statistics about functions

<table>
<thead>
<tr>
<th>Flat Profile</th>
<th>Load Balance</th>
<th>Call Tree</th>
<th>Call Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>TTime</td>
<td>Total</td>
<td>#Calls</td>
</tr>
<tr>
<td>PRECON</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>OMP_SYNC</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MATDL</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>SOLVE</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>User_Code</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Init</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Initiate</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>0:00:00</td>
<td>0:00:00</td>
<td>0</td>
</tr>
</tbody>
</table>

One Intel Software & Architecture (OISA)
Call Tree and Call Graph

Function statistics including calling hierarchy

- Call Tree shows call stack
- Call Graph shows calling dependencies
Communication Profiles

Statistics about point-to-point or collective communication

Matrix supports grouping by attributes in each dimension

- Sender, Receiver, Data volume per msg, Tag, Communicator, Type

Available attributes

- Count, Bytes transferred, Time, Transfer rate
MPI Performance Assistant

- Automatic Performance Assistant
- Detect common MPI performance issues
- Automated tips on potential solutions
Checking MPI Application Correctness

Runtime Correctness Checks

Integration with Debuggers
MPI Correctness Checking

Solves two problems:

• Finding programming mistakes which need to be fixed by the application developer
• Detecting errors in the execution environment

Two aspects:

• Error Detection – done automatically by the tool
• Error Analysis – manually by the user based on:
  • Information provided about an error
  • Knowledge of source code, system, ...
All checks are done at runtime in MPI wrappers

Detected problems are reported on stderr immediately in textual format

A debugger can be used to investigate the problem at the moment when it is found
Categories of Checks

- **Local checks: isolated to single process**
  - Unexpected process termination
  - Buffer handling
  - Request and data type management
  - Parameter errors found by MPI

- **Global checks: all processes**
  - Global checks for collectives and p2p ops
    - Data type mismatches
    - Corrupted data transmission
    - Pending messages
    - Deadlocks (hard & potential)
  - Global checks for collectives – one report per operation
    - Operation, size, reduction operation, root mismatch
    - Parameter error
    - Mismatched MPI_Comm_free()
Severity of Checks

Levels of severity:

• *Warnings*: application can continue
• *Error*: application can continue but almost certainly not as intended
• *Fatal error*: application must be aborted

Some checks may find both warnings and errors

• Example: CALL FAILED check due to invalid parameter
• Invalid parameter in MPI_Send() => msg cannot be sent => *error*
• Invalid parameter in MPI_Request_free() => resource leak => *warning*
Correctness Checking on Command Line

Command line option via –check_mpi flag for Intel MPI Library:

```
$ mpirun --check_mpi -n 2 overlap
[...]
[0] WARNING: LOCAL:MEMORY:OVERLAP: warning
[0] WARNING: New send buffer overlaps with currently active send buffer at address 0x7fbfffec10.
[0] WARNING: Control over active buffer was transferred to MPI at:
[0] WARNING: MPI_Isend(*buf=0x7fbfffec10, count=4, datatype=MPI_INT, dest=0, tag=103, comm=COMM_SELF [0], *request=0x508980)
[0] WARNING: overlap.c:104
[0] WARNING: Control over new buffer is about to be transferred to MPI at:
[0] WARNING: MPI_Isend(*buf=0x7fbfffec10, count=4, datatype=MPI_INT, dest=0, tag=104, comm=COMM_SELF [0], *request=0x508984)
[0] WARNING: overlap.c:105
```
Correctness Checking in GUI

Enable correctness checking info to be added to the tracefile:

- Enable VT_CHECK_TRACING environment variable:
  
  ```
  $ mpirun --check_mpi --genv VT_CHECK_TRACING on --n 4 ./a.out
  ```
Warnings indicate potential problems that could cause unexpected behavior (e.g., incomplete message requests, overwriting a send/receive buffer, potential deadlock, etc.).

Errors indicate problems that violate the MPI standard or definitely cause behavior not intended by the programmer (e.g., incomplete collectives, API errors, corrupting a send/receive buffer, deadlock, etc.).
Debugger Integration

Debugger must be in control of application before error is found

A breakpoint must be set in MessageCheckingBreakpoint()

Documentation contains instructions for automating this process for TotalView*, gdb, and idb.
Trace of a Simple MPI Program

Demo
Related Tools

Intel® MPI Benchmarks
Intel® Cluster Checker
Intel® MPI Benchmarks

- Standard benchmarks with OSI-compatible CPL license
  - Enables testing of interconnects, systems, and MPI implementations
  - Comprehensive set of MPI kernels that provide performance measurements for:
    - Point-to-point message-passing
    - Global data movement and computation routines
    - One-sided communications
    - File I/O
    - Supports MPI-1.x, MPI-2.x, and MPI-3.x standards

- What’s New:
  - Introduction of new benchmarks
    - Measure cumulative bandwidth and message rate values

The Intel® MPI Benchmarks provide a simple and easy way to measure MPI performance on your cluster.
Use an Extensive Diagnostic Toolset for High Performance Compute Clusters—Intel® Cluster Checker (for Linux*)

- **Ensure Cluster Systems Health**
  - Expert system approach providing cluster systems expertise - verifies system health: find issues, offers suggested actions
  - Provides extensible framework, API for integrated support
  - Check 100+ characteristics that may affect operation & performance – improve uptime & productivity

- **New in 2019 Update 5 Release: Output & Features**
  - Improve Usability & Capabilities
    - New default test with faster execution
    - New predefined user/admin specific tests and in-depth analysis
    - Improved summary output on nodes and issue, details in log files
    - Troubleshooting tests on prerequisites for Intel® MPI Library
    - Support for the latest Intel processors (Intel® Xeon® Platinum 9200 Processor Family)
    - BIOS checking capability for administrators, using ‘syscfg’ utility

For application developers, cluster architects & users, & system administrators
Online Resources

Intel® MPI Library product page
  • www.intel.com/go/mpi

Intel® Trace Analyzer and Collector product page
  • www.intel.com/go/traceanalyzer

Intel® Clusters and HPC Technology forums

Intel® MPI Library Tuning Files

Intel® Cluster Checker
Backup
Environment Propagation

- Use \(-[g]env[*]\) to control environment propagation
  - Adding g propagates to all ranks, otherwise only to ranks in current argument set
- \(-\text{env} \ <\text{variable}> \ <\text{value}>\) Set <variable> to <value>
- \(-\text{envuser}\) All user environment variables, with a few exceptions (Default)
- \(-\text{envall}\) All environment variables
- \(-\text{envnone}\) No environment variables
- \(-\text{envlist} \ <\text{variable list}>\) Only the listed variables
Autotuner Detail
# Intel® MPI Library Tuning Approaches

## Table:

<table>
<thead>
<tr>
<th></th>
<th>mpitune</th>
<th>mpitune / fast tuner</th>
<th>autotuner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro benchmark tuning</td>
<td>![Green]</td>
<td>![Green]</td>
<td>![Red]</td>
</tr>
<tr>
<td>Application tuning</td>
<td>![Red]</td>
<td>![Orange]</td>
<td>![Green]</td>
</tr>
<tr>
<td>Easy of use</td>
<td>![Red]</td>
<td>![Green]</td>
<td>![Green]</td>
</tr>
<tr>
<td>Cluster time</td>
<td>![Red]</td>
<td>![Orange]</td>
<td>![Green]</td>
</tr>
<tr>
<td>Adoption to environment</td>
<td>![Red]</td>
<td>![Green]</td>
<td>![Green]</td>
</tr>
</tbody>
</table>
No extra calls. Pure **application driven** tuning

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

```bash
$ 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):

```bash
$ 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, \ldots, \text{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™ Profiler’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™ Profiler’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™ Profiler’s Application Performance Snapshot Update 4
Intel® VTune™ Profiler’s Application Performance Snapshot (APS) per communicator analysis

4. Get the results:

<table>
<thead>
<tr>
<th>Communicators used in the application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicator Size</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>461686018431582688 4</td>
</tr>
<tr>
<td>461686018431582208 4</td>
</tr>
</tbody>
</table>

One Intel Software & Architecture (OISA)
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.
HANDLING HETEROGENEOUS JOBS
Global Options vs. Local Options

- Global Options are applied to all ranks
  - -ppn, -genv, ...
- Local Options are applied to a subset of ranks
  - -n, -host, -env, ...
- **WARNING**: Some options can be set as local options via environment variable, but must be consistent across job
  - Collective algorithms
  - Fabric selection and parameters
Configuration Files and Argument Sets

- Arguments Sets are used on the command line
- Configuration Files are pulled from the file specified by `--configfile <configfile>`
- Global arguments appear first (first line, or at beginning of first argument set)
- Local arguments for each argument set next
- Separated by `:` on command line (don’t separate globals), new line in configfile
- Can be used to run heterogeneous binaries, different arguments for each binary, different environment variables, etc.
- All ranks combined in order specified into one job
Command Line Argument Set

$ mpirun -genv OMP_NUM_THREADS 4 -n 6 -host node1 ./exe1: -n 4 -host node2 ./exe2: -n 6 -host node4 ./exe4

- Host 1 runs “exe1” on “node1” using 6 MPI tasks and 4 threads per MPI task
- No limit to number of different host or executables
- For high numbers of hosts a configuration file is more convenient...
Configuration File

- Configuration file allows flexibility and automation
- Notice commented out line – simple to change host assignment

```
$ cat theconfigfile
-genv OMP_NUM_THREADS 4
-n 6–host node1 ./exe1
-n 4–host node2 ./exe2
# -n 4 –host dead_node3 ./exe3
-n 6–host node4 ./exe4
```

- Launching job is straightforward

```
$ mpirun –configfile theconfigfile
```