Intel® MPI Library
## Intel® MPI Library

### Value Proposition

<table>
<thead>
<tr>
<th>What</th>
<th>Intel’s High Performance MPI Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why</td>
<td>Scale Performance – Tuned for Latest Intel Architectures</td>
</tr>
<tr>
<td></td>
<td>Scale Forward – Multicore and Manycore Ready</td>
</tr>
<tr>
<td></td>
<td>Scale Efficiently – Flexible Fabric Selection &amp; Compatibility</td>
</tr>
<tr>
<td>How</td>
<td>Standards Based – Built on Open Source MPICH Implementation</td>
</tr>
<tr>
<td></td>
<td>Sustained Scalability – Tuning for Low Latencies, High Bandwidth &amp; Increased Processes</td>
</tr>
<tr>
<td></td>
<td>Multi Fabric Support – Supports Popular High Performance Networking Fabrics</td>
</tr>
</tbody>
</table>
Intel® MPI Library Overview

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 adds necessary links to MPI libraries and passes options to underlying compiler

- Use mpiifort, mpiicpc, or mpiicc to force usage of the associated Intel compiler

- Use mpif77, mpicxx, mpicc, or others to allow user to specify compiler (I_MPI_F77, ... or -f77=, -cxx=, ...)
  - Useful for makefiles portable between MPI implementations

- All compilers are found via PATH
MPI Launcher

Robust launch command

```
mpirun <mpi args> executable <program args>
```

Options available for:

- Rank distribution and pinning
- Fabric selection and control
- Environment propagation
- And more
Understanding MPI and Launcher Behavior

I_MPI_DEBUG=<level>

Debug Levels (cumulative):

- 0 – Default, no debug information
- 1 – Verbose error diagnostics
- 2 – Fabric selection process
- 3 – Rank, PID, node mapping
- 4 – Process pinning
- 5 – Display Intel® MPI Library environment variables
- 6 – Collective operation algorithm controls

I_MPI_HYDRA_DEBUG=1 turns on Hydra debug output

- Keep in mind that this gives a LOT of output. Only turn on if needed
Process Placement

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
Fabric Selection

I_MPI_FABRICS=<intranode fabric>:<internode fabric> or <fabric>

Fabric options

- shm – Shared Memory (only valid for intranode)
- dapl – Direct Access Provider Library*
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces*

Default behavior goes through a list to find first working fabric combination

If you specify a fabric, fallback is disabled, I_MPI_FALLBACK=1 to re-enable
Environment Propagation

Use –[g]env[*] to control environment propagation

- Adding g propagates to all ranks, otherwise only to ranks in current argument set

- **env** `<variable>` `<value>` Set `<variable>` to `<value>`

- **envuser** All user environment variables, with a few exceptions (Default)

- **envall** All environment variables

- **envnone** No environment variables

- **envlist** `<variable list>` Only the listed variables
What’s New: Intel® MPI Library 2018

- Up to 11x faster job start-up performance.
- Up to 25% reduction in job finalization time.
- Added support for the latest Intel® Xeon® Scalable processor.
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 \(-\text{configfile} <\text{configfile}>\)

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
Examples

Configuration File

$ 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
$ mpirun –configfile theconfigfile

Argument Set

$ mpirun –genv OMP_NUM_THREADS 4 –n 6 –host node1 ./exe1
- -n 4 –host node2 ./exe2 : –n 6 –host node4 ./exe4
Intel® Trace Analyzer and Collector (ITAC)
### Value Proposition

<table>
<thead>
<tr>
<th>What</th>
<th>Intel’s High Performance MPI Communications Profiler &amp; Analyzer for Scalable HPC Development</th>
</tr>
</thead>
</table>
| Why  | • Scale Performance – Perform on More Nodes  
      • Scale Forward – Multicore and Manycore Ready  
      • Scale Efficiently – Tune & Debug on More Nodes |
| How  | • Visualize – Understand parallel application behavior  
      • Evaluate – Profiling statistics and load balancing  
      • Analyze – Automated analysis of common MPI issues  
      • Identify – Communication hotspots |
Intel® Trace Analyzer and Collector Overview

Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and 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!
## Multiple Methods for Data Collection

<table>
<thead>
<tr>
<th>Collection Mechanism</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run with --trace or preload trace collector library.</td>
<td>Automatically collects all MPI calls, requires no modification to source, compile, or link.</td>
<td>No user code collection.</td>
</tr>
<tr>
<td>Link with --trace.</td>
<td>Automatically collects all MPI calls.</td>
<td>No user code collection.</td>
</tr>
<tr>
<td>Must be done at link time.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compile with --tcollect.</td>
<td>Automatically instruments all function entries/exports.</td>
<td>Requires recompile of code.</td>
</tr>
<tr>
<td>Add API calls to source code.</td>
<td>Can selectively instrument desired code sections.</td>
<td>Requires code modification.</td>
</tr>
</tbody>
</table>
Intel® Trace Analyzer Summary Page

Summary: poisson.sendrecv.single.stf
Total time: 0.675 sec. Resources: 16 processes, 4 nodes.

Ratio
This section represents a ratio of all MPI calls to the rest of your code in the application.

- Serial Code - 0.506 sec  82.4%
- OpenMP - 0 sec  0%
- MPI calls - 0.118 sec  17.5%

Top MPI functions
This section lists the most active MPI functions from all MPI calls in the application.

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>% of Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Sendrecv</td>
<td>0.0441 sec (69.99%)</td>
<td></td>
</tr>
<tr>
<td>MPI_Allreduce</td>
<td>0.0413 sec (6.24%)</td>
<td></td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>0.0076 sec (1.16%)</td>
<td></td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0.0057 sec (0.84%)</td>
<td></td>
</tr>
<tr>
<td>MPI_InitHandler_create</td>
<td>0.0000 sec (0.00%)</td>
<td></td>
</tr>
</tbody>
</table>

Where to start with analysis
For deep analysis of the MPI-bound application click "Continue" to open the traceView and leverage the Intel® Trace Analyzer functionality.

- Performance Analyzer - to identify possible performance problems
- Instinctual Debugger - for detailed inspection of code
- Tagging/Profiling - for thorough customizable analysis

To optimize node-level performance use:

- Intel® VTune™ Amplifier
  - Algorithmic level tuning with hypo-performance and threading efficiency analysis
  - microarchitecture level tuning with general exploration and bandwidth analysis
  - Intel® Advisor
  - vectorization optimization and thread prototyping.

For more information, see documentation for the respective tool:
- Analyzing MPI applications with Intel® VTune™ Amplifier
- Analyzing MPI applications with Intel® Advisor

Show Summary Page when opening a tracefile
Views and Charts

Helps navigating through the trace data and keep orientation

Every View can contain 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)
Event Timeline

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><strong>Name</strong></td>
<td><strong>TSelf</strong></td>
<td><strong>TTot</strong></td>
<td><strong>#Calls</strong></td>
</tr>
<tr>
<td>Group All Threads</td>
<td>678,787,446.9</td>
<td>678,787,446.9</td>
<td>49,336</td>
</tr>
<tr>
<td>PRECON</td>
<td>580,473,344.9</td>
<td>580,473,344.9</td>
<td>206,320</td>
</tr>
<tr>
<td>OMP_SYNC</td>
<td>410,463,131.9</td>
<td>410,463,131.9</td>
<td>48,280</td>
</tr>
<tr>
<td>SOLVER</td>
<td>258,943,181.9</td>
<td>258,943,181.9</td>
<td>128,566</td>
</tr>
<tr>
<td>User_Code</td>
<td>148,746,154.9</td>
<td>148,746,154.9</td>
<td>128,566</td>
</tr>
<tr>
<td>MPB_Boost</td>
<td>94,227,914.9</td>
<td>94,227,914.9</td>
<td>37,348</td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>49,822,701.9</td>
<td>49,822,701.9</td>
<td>32,3,669</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>24,222,480.9</td>
<td>24,222,480.9</td>
<td>49,336</td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>23,907,485.9</td>
<td>23,907,485.9</td>
<td>128,566</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>17,507,615.9</td>
<td>17,507,615.9</td>
<td>128,566</td>
</tr>
<tr>
<td>MPI_Comm_dup</td>
<td>11,756,564.9</td>
<td>11,756,564.9</td>
<td>64,1,836</td>
</tr>
<tr>
<td>MPICycle</td>
<td>7,400,231.9</td>
<td>7,400,231.9</td>
<td>128,566</td>
</tr>
<tr>
<td>MPI_Wtime</td>
<td>4,805,187.9</td>
<td>4,805,187.9</td>
<td>128,566</td>
</tr>
<tr>
<td>MPITime</td>
<td>0,000,288.0</td>
<td>0,000,288.0</td>
<td>32,0,000</td>
</tr>
<tr>
<td>MPITime_neg</td>
<td>0,000,288.0</td>
<td>0,000,288.0</td>
<td>32,0,000</td>
</tr>
</tbody>
</table>

### Optimization Notice

Copyright © 2017, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.*
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
Zooming
Grouping and Aggregation

Allow analysis on different levels of detail by aggregating data upon group-definitions.

Functions and threads can be grouped hierarchically:
- Process Groups and Function Groups

Arbitrary nesting is supported:
- Functions/threads on the same level as groups
- User can define his/her own groups

Aggregation is part of View-definition:
- All charts in a View adapt to requested grouping
- All charts support aggregation
Aggregation Example
Tagging and Filtering

Help concentrating on relevant parts

Avoid getting lost in huge amounts of trace data

Define a set of interesting data

- E.g. all occurrences of function x
- E.g. all messages with tag y on communicator z

Combine several filters:
Intersection, Union, Complement

Apply it

- Tagging: Highlight messages
- Filtering: Suppress all non-matching events
Tagging Example
Filtering Example
Ideal Interconnect Simulator (Idealizer)

Helps to figure out application’s imbalance simulating its behavior in the “ideal communication environment”
MPI Performance Assistance

Automatic Performance Assistant
Detect common MPI performance issues
Automated tips on potential solutions

Automatically detect performance issues and their impact on runtime
MPI-3.0 Support

Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts

Non-blocking Allreduce (MPI_Iallreduce)
What’s New in Intel® Trace Analyzer and Collector 2018

Support for OpenSHMEM* applications.

Support for the latest Intel® Xeon® Scalable and Intel® Xeon Phi™ processors.