INTEL® PARALLEL STUDIO XE 2017
CLUSTER EDITION OVERVIEW
For Distributed Performance
Intel® Parallel Studio XE 2017 development suite
Empowering Faster Code Faster

Delivering HPC Development Solutions

- Over 20 years
- Industry Collaboration on Standards
- Developed with Performance & Scaling with Intel hardware

Meeting the Challenges
- Boosting Performance
- Increasing Scalability
- Increasing Productivity
## Intel® Parallel Studio XE

### Performance Libraries

- **Intel® Math Kernel Library**
  Optimized Routines for Science, Engineering, and Financial

- **Intel® Data Analytics Acceleration Library**
  Optimized for Data Analytics & Machine Learning

### Profiling, Analysis, and Architecture

- **Intel® Inspector**
  Memory and Threading Checking

- **Intel® VTune™ Amplifier**
  Performance Profiler

### Cluster Tools

- **Intel® Cluster Checker**
  Cluster Diagnostic Expert System

- **Intel® Trace Analyzer and Collector**
  MPI Profiler

### Intel® MPI Library

- **Intel® Integrated Performance Primitives**
  Image, Signal, and Compression Routines

- **Intel® Threading Building Blocks**
  Task-Based Parallel C++ Template Library

### Intel® C/C++ and Fortran Compilers

- **Intel® Distribution for Python**
  Performance Scripting
Intel® MPI Library Overview

Optimized MPI application performance
- Application-specific tuning
- Automatic tuning
- New! - Support for Intel® Xeon Phi™ processor (code-named Knights Landing)
- New: Support for Intel® Omni-Path Architecture Fabric

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

Faster MPI communication
- Optimized collectives

Sustainable scalability up to 340K cores
- Native InfiniBand* interface support allows for lower latencies, higher bandwidth, and reduced memory requirements

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

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

```plaintext
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
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
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
```
TUNING MPI APPLICATION PERFORMANCE
Tuning Methods

Library Tuning (algorithms, fabric parameters)

- mpitune

Application Tuning (load balance, MPI/threaded/serial performance)

- Intel® Trace Analyzer and Collector
- Application Performance Snapshot
- Intel® VTune™ Amplifier XE
Library Tuning: mpitune

Use the automatic tuning facility to tune the Intel® MPI Library for your cluster or application (done once, may take a long time)

Modes (see mpitune –h for options)

- Cluster-wide tuning
  
  mpitune ...

- Application-specific tuning
  
  mpitune –application "mpirun –n 32 ./exe" ...

Creates options settings which are used with the –tune flag

  mpirun –tune ...
INTEL® TRACE ANALYZER AND COLLECTOR (ITAC)
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. Must be done at link time.</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>
Summary page shows computation vs. communication breakdown

Is your application MPI-bound?

Resource usage

Largest MPI consumers

Is your application CPU-bound?

Next Steps
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
# Call Tree and Call Graph

Function statistics including calling hierarchy

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

---

<table>
<thead>
<tr>
<th>Group All Processes</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Time</td>
<td>Total</td>
<td>#Calls</td>
<td>Tcall</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.001 s</td>
<td>0.002 s</td>
<td>37</td>
<td>0.002 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.000 s</td>
<td>0.000 s</td>
<td>1</td>
<td>0.000 s</td>
</tr>
<tr>
<td><strong>STF_TraceRedZone</strong></td>
<td>0.009 s</td>
<td>0.016 s</td>
<td>33</td>
<td>0.016 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.010 s</td>
<td>0.018 s</td>
<td>30</td>
<td>0.018 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.003 s</td>
<td>0.005 s</td>
<td>22</td>
<td>0.005 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.016 s</td>
<td>0.019 s</td>
<td>29</td>
<td>0.019 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.033 s</td>
<td>0.033 s</td>
<td>61</td>
<td>0.033 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.000 s</td>
<td>0.000 s</td>
<td>11</td>
<td>0.000 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.000 s</td>
<td>0.000 s</td>
<td>9</td>
<td>0.000 s</td>
</tr>
<tr>
<td><strong>STF_WorkStackHistory</strong></td>
<td>0.000 s</td>
<td>0.000 s</td>
<td>6</td>
<td>0.000 s</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.
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”

Actual trace

Idealized Trace

Easy way to identify application bottlenecks
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive

zero duration

P1

MPI_Isend

P2

MPI_Recv

P1

MPI_Isend

P2

MPI_Recv
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Building Blocks: Elementary Messages

**Early Send / Late Receive**

- P1: `MPI_Isend`
- P2: `MPI_Recv`

**Late Send / Early Receive**

- P1: `MPI_Isend`
- P2: `MPI_Recv`

Load imbalance
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)
Using Intel® VTune™ Amplifier XE on MPI programs

Run VTune underneath MPI

NEW! – VTune can run multiple instances per node

- Results are grouped into one result per node
  - <result folder>.<node name>

- Within result, ranks indicate rank number

```
$ mpirun <mpi args> amplxe-cl <vtune args> -- <application and args>
```
Easier Multi-Rank Analysis of MPI + OpenMP

Tune hybrid parallelism using ITAC + VTune Amplifier

Tune OpenMP performance of high impact ranks in VTune Amplifier

Ranks sorted by MPI Communication Spins – ranks on the critical path are on the top

Process names link to OpenMP metrics

Detailed OpenMP metrics per MPI ranks

Optimization Notice
Copyright © 2017, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
CHECKING MPI APPLICATION CORRECTNESS
MPI Correctness Checking

Solves two problems:

- Finding programming mistakes in the application 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, ...
How Correctness Checking Works

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 0x7fbfffe10.
[0] WARNING: Control over active buffer was transferred to MPI at:
[0] WARNING: MPI_Isend(*buf=0x7fbfffe10, 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=0x7fbfffe10, 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 trace file:

- Enable VT_CHECK_TRACING environment variable:

  ```bash
  $ mpirun --check_mpi --genv VT_CHECK_TRACING on -n 4 ./a.out
  ```
Viewing Source Code

**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.
Dynamic Analysis
Launch Intel® Inspector
- Use mpirun
- List your app as a parameter

Results organized by MPI rank

Review results
- Graphical user interface
- Command line report

Static Analysis
Source analyzed for errors (similar to a build)

Review results
- Graphical user interface

Find errors earlier when they are less expensive to fix
Using Intel® Inspector with MPI

Use the command-line tool under the MPI run script to gather report data

$ mpirun -n 4 inspxe-cl -r my_result -collect mi1 -- ./test

Argument Sets can be used for more control

- Only collect data on certain ranks
- Different collections or options on different ranks

A unique results directory is created for each analyzed MPI rank
Launch the GUI and view the results for each rank
BENCHMARKING MPI AND CLUSTER PERFORMANCE
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.
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® Xeon Phi™ Coprocessor Developer Community

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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.

Copyright © 2016, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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