Intel® cluster tools on NERSC systems

For Distributed Performance

Mark Lubin, TCE, SSG/Intel

March 9th, 2017
Intel® Parallel Studio XE 2017

**HPC Cluster**

MPI Messages

**Vectorized & Threaded Node**

**Cluster Edition**
- Multi-fabric MPI library
- MPI error checking and tuning (ITAC and MPS)

**Professional Edition**
- Threading design & prototyping
- Parallel performance tuning
- Memory & thread correctness

**Composer Edition**
- Intel® C++ and Fortran compilers
- Parallel models (e.g., OpenMP*)
- Optimized libraries
*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
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* -- default currently on Cori
- ofa – Open Fabric Alliance (OFED* verbs)
- tmi – Tag Matching Interface
- tcp – Ethernet/Sockets
- ofi – OpenFabrics Interfaces* -- we advise to use OFI

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
Enhanced OFI support

More info: https://ofiwg.github.io/libfabric/
Resources

- A BKM for Working with libfabric* on a Cluster System when using Intel® MPI Library

- Howard Pritchard (LANL) repositories on OFI
  - https://github.com/hppritcha
  - https://github.com/ofi-cray/libfabric-cray/wiki
Using Intel MPI 2017 with OFI libfabric at NERSC

• Load the Intel 17 and Intel MPI 2017 modules, e.g.
  • module load intel
  • module load impi

• set environment variable to tell Intel MPI where to find the libfabric shared library, e.g.
  • % export I_MPI_OFI_LIBRARY=/global/homes/h/hpp/libfabric-v1.0.4rc1_install/lib/libfabric.so

• Specify the location of the SLURM PMI library. For Cori the setting is
  • % export I_MPI_PMI_LIBRARY=/usr/lib64/slurmpmi/libpmi.so

• tell Intel MPI to use OFI libfabric and GNI OFI provider
  • % export I_MPI_FABRICS=ofi
  • % export I_MPI_OFI_PROVIDER=gni
Tuning Methods (for MPI)

Library Tuning (algorithms, fabric parameters)
- Mpitune (next slide)

Application Tuning (load balance, MPI/threaded/serial performance)
- Intel® Trace Analyzer and Collector (ITAC)
- Intel® MPI Performance Snapshot (MPS)
- 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® Cluster Tools on NERSC clusters – all you need to know

• Intel® Advisor, Intel® Vtune and Intel® Inspector at NERSC:
  • https://www.nersc.gov/users/software/performance-and-debugging-tools/advisor/
  • https://www.nersc.gov/users/software/performance-and-debugging-tools/vtune/
  • https://www.nersc.gov/users/software/performance-and-debugging-tools/inspector/

• Intel® Tools ONLY WORK on Lustre file system at NERSC- please use $SCRATCH!

• Running Intel® Parallel Studio XE Analysis Tools on Clusters with Slurm*/srun
  • Next slide in details
Attaching analysis tools, Intel® VTune Amplifier XE, Intel® Inspector XE or Intel® Advisor XE, to srun

- However, the following, in case of Vtune, will attach the tool to each MPI rank:
  
  - $ srun amplxe-cl –c hotspots –r my_result_1 -- ./my_application

- If the user is only interested in analyzing a subset of MPI ranks or shared memory nodes, they can leverage the multiple program configuration from srun.

- Create config file
  
  - cat > mpmd_vtune.cfg << EOF
  - 
  - 0-98  ./my_application
  - 99  amplxe-cl –c hotspots –r my_result_2 -- ./my_application
  - 100-255  ./my_application
  - EOF
  
  - Run
  
  - srun --multi-prog  ./mpmd_vtune.cfg
Intel® Trace Analyzer and Collector (ITAC) and MPI Performance Snapshot (MPS) on NERSC systems

• Intel® Cluster Tools in a Cray* environment
  • $ module load itac

• ITAC - to collect the trace – just preload ITAC library
  • $export LD_PRELOAD=/path_to_ITAC_installation/intel64/slib/libVT.so

• MPS: preload library and collect statistics
  • $export LD_PRELOAD=/path_to_MPS_installation/intel64/slib/libmps_nopapi.so
  • $export I_MPI_STATS=20
  • $export I_MPI_STATS_COMPACT=1

• Run application
  • $srun –n 8 -c 8 ./app_name
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 for load balancing
- Identify communication hotspots

Features

- Event-based approach
- Low overhead
- Excellent scalability
- Powerful aggregation and filtering functions
- Performance Assistance and Imbalance Tuning
- NEW in 9.1: MPI Performance Snapshot

Source Code
- Compiler
- Objects
- Linker
- Binary
- Runtime
- Output

Intel® Trace Collector
- Trace File (.stf)

API and -tcollect

Intel® Trace Analyzer

-trace
# Strengths of Event-based Tracing

<table>
<thead>
<tr>
<th>Predict</th>
<th>Detailed MPI program behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record</td>
<td>Exact sequence of program states – keep timing consistent</td>
</tr>
<tr>
<td>Collect</td>
<td>Collect information about exchange of messages: at what times and in which order</td>
</tr>
</tbody>
</table>

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?  
Largest MPI consumers  
Resource usage  
Is your application CPU-bound?  
Next Steps
MPI-3.0 Support

Support for major MPI-3.0 features

- Non-blocking collectives
- Fast RMA
- Large counts
Intel® Trace Analyzer and Collector

Compare the event timelines of two communication profiles
- Blue = computation
- Red = communication

Chart showing how the MPI processes interact
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
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
Ideal Interconnect Simulator (Idealizer)

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

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
Building Blocks: Elementary Messages

Early Send / Late Receive

P1
MPI_Isend
P2
MPI_Recv

Late Send / Early Receive

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

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 Performance Snapshot (MPS)
MPI Performance Snapshot (MPS)

- New tool available with Intel® Trace Analyzer and Collector
- Enables developer to quickly collect performance summary of large jobs
  - Tested to 37,000 ranks
- Lightweight, scalable collection
- Report on application imbalances between MPI, OpenMP*, and serial time
- Report hardware counters via Intel® VTune™ Amplifier XE or PAPI
- Separates statistical analysis from event analysis
Why MPI Performance Snapshot (MPS)?

- **Advantages**
  - Get an initial profile of the application very quickly
  - Performance variation at scale can be detected and triaged quickly
  - Provides development recommendations to developers based on analysis
    - Intel® Trace Analyzer and Collector or Intel® VTune™ Amplifier XE for deeper analysis
  - Easy to use out of the box functionality

- **Benefits**
  - Difficult performance issues are easier to spot
  - Application performance guidance is obtained easily
  - Experienced & non-experienced developers can adopt quickly
## Complementary MPI Analysis Tools

<table>
<thead>
<tr>
<th></th>
<th><strong>Intel® Trace Analyzer and Collector</strong></th>
<th><strong>MPI Performance Snapshot</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>0 – 4K ranks</td>
<td>1K – 100K ranks (37K tested)</td>
</tr>
<tr>
<td><strong>Collection Details</strong></td>
<td>High (events, source hooks)</td>
<td>Low (aggregation)</td>
</tr>
<tr>
<td><strong>Collection Size</strong></td>
<td>Huge (~17 GB for 1K ranks)</td>
<td>Much less (~0.8 GB for 1K ranks; ~4.5 GB for 37K)</td>
</tr>
<tr>
<td><strong>Event-based Analysis</strong></td>
<td>Yes</td>
<td>EBS or PAPI events</td>
</tr>
<tr>
<td><strong>Statistics Analysis</strong></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Quick Processing</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Small &amp; flexible</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Collector</strong></td>
<td>Intel® Trace Collector</td>
<td>Intel® MPI Library built-in statistics and MPI/OpenMP* imbalance</td>
</tr>
</tbody>
</table>
## HTML Reporting

### MPI Performance Snapshot Summary

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>WallClock time:</td>
<td>21.78 sec</td>
<td></td>
</tr>
<tr>
<td>MPI Time:</td>
<td>17.28 sec</td>
<td>80.10%</td>
</tr>
<tr>
<td>OpenMP Time:</td>
<td>19.46 sec</td>
<td>90.19%</td>
</tr>
<tr>
<td>Serial Time:</td>
<td>0.00 sec</td>
<td>0.00%</td>
</tr>
<tr>
<td>MPI Imbalance:</td>
<td>7.44 sec</td>
<td>34.47%</td>
</tr>
<tr>
<td>OpenMP Imbalance:</td>
<td>17.05 sec</td>
<td>79.03%</td>
</tr>
<tr>
<td>Computation Time:</td>
<td>4.29 sec</td>
<td>19.90%</td>
</tr>
</tbody>
</table>

**Description:**

- **WallClock time:** The total application runtime. The time is elapsed time for the slowest process. This metric includes the MPI Time and the Computation time below.
- **MPI Time:** Time spent inside the MPI library. High values are usually bad. This value is **HIGH**. The application is **Communication-bound**. More details...
- **MPI Imbalance:** Mean unproductive wait time per process spent in the MPI library when a process is waiting for data. This time is part of the MPI Time above. High values are usually bad. This value is **HIGH**. The application workload is **NOT well balanced** between MPI ranks. More details...
- **Computation Time:** Mean time per process spent in the application code. This is the sum of the OpenMP Time and the Serial time. High values are usually good. This value is **LOW**.
- **OpenMP Time:** Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded. This value is **HIGH**.
- **OpenMP Imbalance:** Mean unproductive wait time per process spent in OpenMP parallel regions (usually at synchronization barriers). High values are usually bad. This value is **HIGH**. The application's OpenMP work sharing is **NOT well load-balanced**. More details...
- **Serial Time:** Mean application time per process spent outside OpenMP parallel regions. High values may be good or bad depending on the application algorithm. This value is **NEGLOGILE**. This application is **well parallelized** via OpenMP directives.

**Application:** build/heat_demos

**Number of ranks:** 17

**Used statistics:** app_stat_20160310-035458.txt, stats_20160310-035458.txt

**Creation date:** 2016-03-10 03:55:21
High Capacity MPI Profiler

Combination of lightweight collector (EBS/PAPI events + OMP itt_notify metrics + MPI wait time metrics) and internal MPI statistics

Metrics collected:

- MPI time vs. application time
- Sum of time spent in each MPI function
- MPI message size and transfer data (total and per rank)
- HW counters from EBS or PAPI (e.g. FP, vectorized DP, memory access instructions)
- Memory usage stats (total and per rank)
- MPI/OpenMP/Serial imbalance

Shows OpenMP/MPI imbalance and HW counters all in one result

Easy to use ‘entry point’ for starting analysis

Currently only available on Linux* as command line tool
MPS Usage
4 quick steps to getting started

Install Intel® MPI Library and Intel® Trace Analyzer and Collector

Setup your environment

$ source /opt/intel/itac/9.1/bin/mpsvars.sh --vtune

Run with the MPI Performance Snapshot enabled

$ mpirun -mps -n 1024 ./exe

Analyze your results

$ mps ./stats_<timestamp>.txt ./app_stat_<timestamp>.txt
MPS Output

Summary

Files and folders:

- stats.txt
  - MPI statistics
- app_stat.txt
  - MPS collector statistics
- _mps/results.<node>/
  - VTune results

---

### GENERAL STATISTICS

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total time</td>
<td>448.391 sec (All ranks)</td>
</tr>
<tr>
<td>MPI</td>
<td>40.73%</td>
</tr>
<tr>
<td>NON_MPI</td>
<td>59.27%</td>
</tr>
<tr>
<td>WallClock</td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>89.594 sec (rank 1)</td>
</tr>
<tr>
<td>MAX</td>
<td>89.975 sec (rank 4)</td>
</tr>
</tbody>
</table>

### MEMORY USAGE STATISTICS

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All ranks</td>
<td>226.969 MB</td>
</tr>
<tr>
<td>MIN</td>
<td>24.172 MB (rank 2)</td>
</tr>
<tr>
<td>MAX</td>
<td>96.465 MB (rank 0)</td>
</tr>
</tbody>
</table>

### MPI IMBALANCE STATISTICS

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Imbalance</td>
<td>31.798 sec</td>
</tr>
<tr>
<td>MIN</td>
<td>2.219 sec</td>
</tr>
<tr>
<td>MAX</td>
<td>9.157 sec</td>
</tr>
<tr>
<td></td>
<td>7.092% (All ranks)</td>
</tr>
<tr>
<td></td>
<td>2.467% (rank 4)</td>
</tr>
<tr>
<td></td>
<td>10.219% (rank 0)</td>
</tr>
</tbody>
</table>
MPS HTML Report Breakdown – MPI Time

MPI Time – Time spent in MPI calls

MPI Imbalance – MPI time spent waiting

Lower is better

If MPI Time or MPI Imbalance are high, use Intel® Trace Analyzer and Collector to investigate and optimize MPI usage.
MPS HTML Report Breakdown – OpenMP Time

OpenMP Time – Computation time spent in OpenMP parallel regions – higher is better

OpenMP Imbalance – OpenMP Time spent waiting – lower is better

If OpenMP Imbalance is high – recommend using Intel® VTune™ Amplifier XE

If OpenMP Time is low – Intel® Advisor to find opportunities to add more threading

<table>
<thead>
<tr>
<th>OpenMP Time: 19.46 sec</th>
<th>90.19%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time per process spent in the OpenMP parallel regions. High values are usually good and indicate that the application is well-threaded. This value is HIGH.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OpenMP Imbalance: 17.05 sec</th>
<th>79.03%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean unproductive wait time per process spent in OpenMP parallel regions (normally at synchronization barriers). High values are usually bad. This value is HIGH. The application's OpenMP work sharing is NOT well load-balanced. More details...</td>
<td></td>
</tr>
</tbody>
</table>
Useful MPS Reports – MPI Function Summary

```
[~/projects/Cardiac_demo-master]$ mps app_stat_20160310-035458.txt stats_20160310-035458.txt -f
Reading: app_stat_20160310-035458.txt
Reading: stats_20160310-035458.txt
Done.

Function summary for all ranks

<table>
<thead>
<tr>
<th>Function</th>
<th>Time(sec)</th>
<th>Time(%)</th>
<th>Volume(MB)</th>
<th>Volume(%)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait</td>
<td>259.42</td>
<td>88.30</td>
<td>0.00</td>
<td>0.00</td>
<td>2560144</td>
</tr>
<tr>
<td>Test</td>
<td>15.59</td>
<td>5.31</td>
<td>0.00</td>
<td>0.00</td>
<td>21843828</td>
</tr>
<tr>
<td>Barrier</td>
<td>10.31</td>
<td>3.51</td>
<td>0.00</td>
<td>0.00</td>
<td>1040</td>
</tr>
<tr>
<td>Send</td>
<td>2.96</td>
<td>1.01</td>
<td>4568.14</td>
<td>49.97</td>
<td>4000252</td>
</tr>
<tr>
<td>Init</td>
<td>2.80</td>
<td>0.95</td>
<td>0.00</td>
<td>0.00</td>
<td>17</td>
</tr>
<tr>
<td>Allreduce</td>
<td>1.14</td>
<td>0.39</td>
<td>0.01</td>
<td>0.00</td>
<td>1584</td>
</tr>
</tbody>
</table>
```

[skipped 9 lines]

| TOTAL     | 293.80    | 100.00  | 9142.59    | 100.00    | 32408964 |
### Useful MPS Reports – MPI Time per Rank

```bash
[~/projects/Cardiac_demo-master]$ mps app_stat_20160310-035458.txt stats_20160310-035458.txt \-t
Reading: app_stat_20160310-035458.txt
Reading: stats_20160310-035458.txt
Done.
```

<table>
<thead>
<tr>
<th>Rank</th>
<th>LifeTime (sec)</th>
<th>MPI Time (sec)</th>
<th>MPI Time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0015</td>
<td>21.50</td>
<td>19.52</td>
<td>90.81</td>
</tr>
<tr>
<td>0013</td>
<td>21.50</td>
<td>19.31</td>
<td>89.83</td>
</tr>
<tr>
<td>0007</td>
<td>21.50</td>
<td>19.29</td>
<td>89.71</td>
</tr>
<tr>
<td>0009</td>
<td>21.50</td>
<td>19.28</td>
<td>89.69</td>
</tr>
<tr>
<td>0004</td>
<td>21.50</td>
<td>19.23</td>
<td>89.43</td>
</tr>
<tr>
<td>0012</td>
<td>21.50</td>
<td>19.20</td>
<td>89.33</td>
</tr>
<tr>
<td>0011</td>
<td>21.50</td>
<td>19.17</td>
<td>89.14</td>
</tr>
<tr>
<td>0014</td>
<td>21.50</td>
<td>19.12</td>
<td>88.96</td>
</tr>
<tr>
<td>0010</td>
<td>21.50</td>
<td>19.12</td>
<td>88.94</td>
</tr>
<tr>
<td>0005</td>
<td>21.50</td>
<td>19.08</td>
<td>88.75</td>
</tr>
<tr>
<td>0006</td>
<td>21.50</td>
<td>19.06</td>
<td>88.67</td>
</tr>
<tr>
<td>0008</td>
<td>21.50</td>
<td>18.74</td>
<td>87.15</td>
</tr>
<tr>
<td>0000</td>
<td>21.78</td>
<td>18.25</td>
<td>83.80</td>
</tr>
<tr>
<td>0016</td>
<td>21.78</td>
<td>15.93</td>
<td>73.13</td>
</tr>
<tr>
<td>0003</td>
<td>21.76</td>
<td>13.55</td>
<td>62.30</td>
</tr>
<tr>
<td>0002</td>
<td>21.76</td>
<td>13.00</td>
<td>59.77</td>
</tr>
<tr>
<td>0001</td>
<td>21.76</td>
<td>2.94</td>
<td>13.52</td>
</tr>
</tbody>
</table>

**TOTAL**: 366.81, 293.80, 80.10

**AVG**: 21.58, 17.28, 80.10
Using Intel® VTune™ Amplifier XE on MPI programs

Run VTune underneath MPI

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

![Top OpenMP Processes by MPI Communication Spin Time](image)

- Process names link to OpenMP metrics
- Detailed OpenMP metrics per MPI ranks
- Ranks sorted by MPI Communication Spins – ranks on the critical path are on the top
- Per-rank OpenMP Potential Gain and Serial Time metrics
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:

```bash
$ 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 0x7fbfffc10.
[0] WARNING: Control over active buffer was transferred to MPI at:
[0] WARNING: MPI_Isend(buf=0x7fbfffc10, 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=0x7fbfffc10, 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:

  $ 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.
Usage of Debugger

error detected, process stopped at breakpoint

access to the parameters
full access to application source code and data

Copyright © 2016, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.
Intel® Inspector XE

Dynamic Analysis

Launch Intel® Inspector XE
- Use mpirun
- List your app as a parameter

Results organized by MPI rank

Review results
- Graphical user interface
- Command line report

Find errors earlier when they are less expensive to fix

Static Analysis

Source analyzed for errors (similar to a build)

Review results
- Graphical user interface
Using Intel® Inspector XE with MPI

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

```bash
$ 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 4.1

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