Intel® Cluster Studio XE 2013
for Distributed Performance

Gergana Slavova
gergana.s.slavova@intel.com
Technical Computing, Analyzers, and Runtimes
Intel Software & Services Group
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.

Performance tests and ratings are measured using specific computer systems and/or components and
reflect the approximate performance of Intel products as measured by those tests. Any difference in
system hardware or software design or configuration may affect actual performance. Buyers should
consult other sources of information to evaluate the performance of systems or components they are
considering purchasing. For more information on performance tests and on the performance of Intel
products, reference

Copyright © 2012 , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core,
VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries. *Other names
and brands may be claimed as the property of others.

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
Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013

More Cores
- Multicore Many-core
- Xeon
- Xeon Phi
- 50+ cores

Wider Vectors
- 128 Bits
- 256 Bits
- 512 Bits

Scaling Performance Efficiently
- Serial Performance
- Task & Data Parallel Performance
- Distributed Performance

Serial, Threaded & Cluster Application Development Suites
- Industry-leading performance from advanced compilers
- Comprehensive libraries
- Parallel programming models
- Insightful analysis tools

intel.com/software/products
Intel® Cluster Studio XE
Scale Forward, Scale Faster – for HPC Clusters

Scale Performance – Perform on More Nodes
- MPI Latency - Intel® MPI Library - Up to 2.6X as fast as alternative MPI libraries
- Compiler Performance – Industry leading Intel® C/C++ & Fortran compilers

Scale Forward – multicore now, many-core ready
- Intel® MPI Library scales beyond 120k processes
- Parallel Programming Models – Commercially supported Intel® versions of open source Threading Building Blocks 4.0 and Intel® Cilk™ Plus 1.1, MPI, OpenMP 3.1, Coarray Fortran
- Focused to preserve programming investments for multicore on future many-core machines

Scale Efficiently – Tune & Debug on More Nodes
- Thread & Memory Correctness Checking – Intel® Inspector XE now MPI enabled across many nodes
- Rapid Node Level Performance Profiling – Intel VTune Amplifier XE can identify hotspots faster and on thousands of nodes

Optimization Notice
Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
in tel.com/software/products
Intel® MPI Library Overview

- Optimized MPI application performance
  - Application-specific tuning
  - Automatic tuning
- Lower latency and multi-vendor interoperability
  - Industry leading latency
  - Performance optimized support for the latest OFED capabilities through DAPL 2.0
- Faster MPI communication
  - Optimized collectives
- Simplify and accelerate clusters
  - “Intel® Cluster Ready”
- Sustainable scalability beyond 120K 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
Technical Performance Benchmark Report

MPI Latency: 96 Processes / 8 Nodes on Intel processor running Linux* 64

Intel® MPI Library vs. alternative MPI libraries

Industry Leading Performance with Intel® MPI Library 4.1
Relative (Geomean) MPI Latency Benchmarks on Linux* 64 (Higher is Better)
96 Processes on 8 nodes (InfiniBand + shared memory)

Up to 2.6X as fast as on 8 nodes

Configuration Info - Sw Versions: Intel® C/C++ version 13.0, Intel® MPI Library 4.1, Platform MPI B.2.1, MVAPICH2 1.8, Open MPI 1.6.1, Intel® MPI Benchmarks 3.2.4; Hardware: Intel® Xeon® CPU DP X5680 @ 3.33GHz, RAM 24GB; Interconnect: InfiniBand, ConnectX adapters; QOR; Operating System: SLES 11.1; Notes: 96 Processes on 8 nodes (InfiniBand + shared memory). All listed MPI libraries were built with the Intel® C++ Compiler 12.1 Update 10 for Linux*.

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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

Optimization Notice: Intel® compiler's 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.
MPI Latency: 96 Processes / 8 Nodes on Intel processor running Windows* 64
Intel® MPI Library vs. alternative MPI libraries

Industry Leading Performance with Intel® MPI Library 4.1
Relative (Geomean) MPI Latency Benchmarks on Windows* 64 (Higher is Better)
96 Processes on 8 nodes (InfiniBand + shared memory)

Up to 3.2X as fast as on 8 nodes

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 these 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. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

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

Configuration Info - SW Versions: Intel® C/C++ version 13.0, Intel® MPI Library 4.1, Platform MPI 8.2.1, MS MPI 3.4, Intel® MPI Benchmarks 3.2.4; Hardware: Intel® Xeon® CPU DP X5680@3.33Ghz, RAM 24GB; Interconnect: InfiniBand, ConnectX adapters; QDR; Operating System: Windows Server 2008 R2 x64 HPC Edition; Notes: 96 Processes on 8 nodes (InfiniBand + shared memory). All listed MPI libraries were built with the Intel® C++ Compiler 12.1 Update 10 for Windows*.

Optimization Notice
Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.
Intel® MPI Library Overview

Streamlined product setup
• Installation under root or ordinary user ID
• mpivars.(c)sh scripts for easy path setting

Simplified process management
• mpiexec -perhost and -nolocal options
• mpirun script that automates usage of the Hydra process manager
• System-, user-, and session-specific configuration files

Environment variables for runtime control over
• Process pinning
• Optimized collective operations
• Device-specific protocol thresholds
• Collective algorithm thresholds
• Enhanced memory registration cache
• Many others ...
Compile and Link Commands

Using Intel compilers

mpiicc, mpiicpc, mpiifort, ...

Using Gnu compilers (same underlying Intel MPI library)

mpicc, mpicxx, mpif77, ...

Ease of use

• Commands find the Intel® MPI Library include files automatically
• Commands link the Intel® MPI libraries automatically

Commands use compilers from PATH (or selected through options); compilers not hard-wired!

Example:

• Compile using the Intel Fortran compiler

  $ mpiifort -o testf test.f
Execution Commands

All-inclusive

```
mpirun -f hostfile -n #processes executable
```

- Most common usage scenario
  - Convenient
  - Uses new Hydra process manager by default
  - Better for jobs in batch system
    “In-session” mode: mpirun acquires the list of nodes from the batch system

Example:

- Run the test program

```
$ mpirun -f hosts.file -n 2 ./testc
Hello world: rank 0 of 2 running on node1
Hello world: rank 1 of 2 running on node1
```
Process Placement

1. Simple process placement (consecutive assignment of MPI ranks to round robin selection of nodes)
   
   mpirun [-perhost #ppn] -n #procs executable
   - Place #ppn processes per node until the total number #procs of processes is reached

2. Exact process placement using Argument Sets:
   
   mpirun -n #p1 -hosts node1 exe1 : -n #p2 -hosts node2 exe2
   - Argument Set (separated by ":") is valid for the specified node:
     Place #p1 processes of exe1 on node1
     Place #p2 processes of exe2 on node2, ...
     (usually: exe1 = exe2 = ...)

3. Exact process placement with a config file
   
   $ cat theconfigfile
   -n #p1 -hosts node1 exe1
   -n #p2 -hosts node2 exe2
   #-n #p3 -hosts dead_node3 exe3
   -n #p4 -hosts node4 exe4
   mpirun -configfile theconfigfile
   - One argument set per line in a file
     - Comment unused lines with "#"
Intel® MPI Library Fabric Selection

Environment variable $I\_MPI\_FABRICS$ selects the interconnect device at runtime

$I\_MPI\_FABRICS$ values:
- shm (shared memory only)
- dapl (DAPL fabrics)
- tcp (sockets)
- tmi
- ofa

$shm:dapl$ fabrics is default

Example
- Check selected device
  
  ```bash
  $ mpirun -f hosts.file -genv I_MPI_DEBUG 2 -n 2 ./testc  
  ..... will use default fabric shm:dapl (RDMA-enabled device + shared memory)
  ```

- Change selected device
  
  ```bash
  $ mpirun -f hosts.file -genv I_MPI_DEBUG 2 -genv I_MPI_FABRICS shm:ofa -n 2 ./testc  
  ..... will use fabric shm:ofa (OFED verbs + shared memory)
  ```
Performance 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)

Example (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® MPI Library support for the Intel® Xeon Phi™ Coprocessor
**MPI+Offload**

MPI ranks on Intel® Xeon® processors (only)

All messages into/out of processors

Offload models used to accelerate MPI ranks

Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® MIC Architecture

Homogenous network of hybrid nodes:
MPI+Offload

How to run

Compile your code with the offload directives

$ mpiifort -openmp test.f -o test.offload

Create your hosts file (Xeon only)

$ cat hosts
node0
node1

Run your application (Xeon only)

$ mpirun -f hosts -n 2 ./test.offload
Many-core Hosted (Native)

MPI ranks on Intel® Xeon Phi™ coprocessors (only)

All messages into/out of Intel® Xeon Phi™ coprocessors

Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads used directly within MPI processes

Programmed as homogenous network of many-core CPUs:
Many-core Hosted (Native)  
*How to run*

Compile your code for Intel® Xeon Phi™ Coprocessor

```
$ mpiifort -mmic test.f -o test.mic
```

Copy the MIC-enabled executable to the coprocessor

```
$ scp test.mic mic0:/home/user/
$ scp test.mic mic1:/home/user/
```

Create your hosts file (MIC only)

```
$ cat hosts
mic0
mic1
```

Let the library know you’re planning on running on MIC

```
$ export I_MPI_MIC=1
```

Run your application (from the Xeon)

```
$ mpirun -f hosts -n 4 /home/user/test.mic
```
Symmetric

MPI ranks on Intel® Xeon Phi™ coprocessors and Intel® Xeon® processors

Messages to/from any core

Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes

Programmed as heterogeneous network of homogeneous nodes:
Symmetric

How to run

Compile your code for the Intel® Xeon node

```bash
$ mpiifort test.f -o /home/user/test
```

And for Intel® Xeon Phi™ Coprocessor

```bash
$ mpiifort -mmic test.f -o test.mic
```

Copy the MIC-enabled executable to the coprocessor (rename during copy)

```bash
$ scp test.mic mic0:/home/user/test
$ scp test.mic mic1:/home/user/test
```

Create your hosts file (Xeon+MIC)

```bash
$ cat hosts
node0
mic0
mic1
```

Let the library know you’re planning on running on MIC

```bash
$ export I_MPI_MIC=1
```

Run your application (from the Xeon)

```bash
$ mpirun -f hosts -n 4 /home/user/test
```
Utilize the POSTFIX env variable

**Supported on NFS-shared cards**

Compile your code for the Intel® Xeon node

```
$ mpiifort test.f -o test
```

And for Intel® Xeon Phi™ Coprocessor

```
$ mpiifort -mmic test.f -o test.mic
```

Create your hosts file (Xeon+MIC)

```
$ cat hosts
node0
mic0
mic1
```

Instead of copying the executable, simply tell the Intel MPI Library to add a postfix to the executable when running on Intel® Xeon Phi™

```
$ export I_MPI_MIC_POSTFIX=.mic
```

Let the library know you’re planning on running on MIC

```
$ export I_MPI_MIC=1
```

Run your application (from the Xeon)

```
$ mpirun -f hosts -n 4 test
```

# When the test executable is run on a Xeon Phi™ host, the mpirun script will automatically add the .mic postfix
Utilize the PREFIX env variable

Supported on NFS-shared cards

Compile your code for the Intel® Xeon node

\$ mpiifort test.f -o test

And for Intel® Xeon Phi™ Coprocessor (place the executable in a separate directory)

\$ mpiifort -mmic test.f -o ./MIC/test

Create your hosts file (Xeon+MIC)

\$ cat hosts
node0
mic0
mic1

Tell the Intel MPI Library that all Intel® Xeon Phi™ executable are located in a separate path

\$ export I_MPI_MIC_PREFIX=./MIC/

Let the library know you’re planning on running on MIC

\$ export I_MPI_MIC=1

Run your application (from the Xeon)

\$ mpirun -f hosts -n 4 test

# When the test executable is run on a Xeon Phi™ host, the mpirun script will automatically add the ./MIC/ prefix
Create a configuration file for ease-of-use

Instead of editing a long and tedious command line:

```
$ mpirun -genv I_MPI_DEBUG 3 \\
-host mic0 -n 2 -env OMP_NUM_THREADS 4 /home/user/exec.a : \\
-host mic1 -n 4 -env OMP_NUM_THREADS 2 /home/user/exec.b
```

Create a configuration file (which you can edit as needed)

```
$ cat conf_file
-genv I_MPI_DEBUG 3  # global settings, applied to all executables (exec.a & exec.b)
# settings applicable to exec.a only
-host mic0 -n 2 -env OMP_NUM_THREADS 4 /home/user/exec.a
# settings applicable to exec.b only
-host mic1 -n 2 -env OMP_NUM_THREADS 2 /home/user/exec.b
```

And always run the same command line

```
$ mpirun -configfile conf_file
```
Intel® MPI Library 4.1 Update 1

What’s New on Linux*

Improved MPI application performance and scalability

• Better Scalability at OFA fabric via new connection manager
• Improved support for NUMA applications and advanced process pinning controls
• Addition of a DAPL* auto-provider functionality for selecting best fabric at startup

Extended support for the Intel® Xeon Phi™ Coprocessor architecture for improved bandwidth and latency through:

• Coprocessor Communication Link (CCL) support
• Native port of the Tag Matching Interface (TMI) over the Qlogic* PSM fabric
• Extending support for Checkpoint/Restart (BLCR*) on the Intel® Xeon Phi™ coprocessor

Backwards compatibility with existing Intel® MPI Library 4.x applications

New GUI-based installer

*Other brands and names are the property of their respective owners.
Intel® MPI Library 4.1 Update 1
What’s New on Windows*

Highly Scalable Hydra Process Manager
• Now available on Windows* OS

Support for Microsoft*’s Network Direct
• Enabling low-latency RDMA devices on Windows*-based clusters

Intel® Xeon Phi™ Coprocessor support
• Added support for offloading work from a Windows* host
Tune Hybrid Cluster MPI and Thread Performance

Intel® Trace Analyzer and Collector

- Tune cross-node MPI
  - Visualize MPI behavior
  - Evaluate MPI load balancing
  - Find communication hotspots

Intel® VTune™ Amplifier XE

- Tune single node threading
  - Visualize thread behavior
  - Evaluate thread load balancing
  - Find thread sync. bottlenecks
Introduction – What is Tracing?

Record program execution

• Program events such as function enter/exit, communication

1:1 protocol of the actual program execution

• Sampling gathers statistical information

Accurate data

Easily get loads of data
Intel® Trace Analyzer and Collector helps the developer:

- Visualize and understand parallel application behavior
- Evaluate profiling statistics and load balancing
- Identify communication hotspots
Event based approach

Event = time stamp + thread ID + description

Function entry/exit

Messages

Collective operations

Counter samples
Strengths of Event-based Tracing

Predict detailed program behavior

Record exact sequence of program states – keep timing consistent

Collect information about exchange of messages: at what times and in which order

An event-based approach is able to detect temporal dependencies!
Key Features

Low Overhead

Catch all MPI events

Powerful configuration mechanism
• Filters, settings, features

Automatic source-code references

Instrumentation
• Rich API
• Binary instrumentation (itcpin)
• Compiler based (-tcollect)

Fail-safe version

Comparison of multiple profiles

Idealizer

MPI Correctness Checking
How to use Intel® Trace Analyzer and Collector

Step 1: Run your binary and create a tracefile
run the binary for a representative amount of time (to reduce initialization influences) on representative data (no corner cases)

```
$ mpirun -trace -n 2 ./test
```

- Alternative: Generate an instrumented binary via re-linking
  
  ```
  $ mpiicc -trace test.c -o test.inst
  $ mpirun -n 2 ./test.inst
  ```

Step 2: To view the generated trace file, start the GUI:

```
traceanalyzer &
```
Support for the Intel® Xeon Phi™ coprocessor

The tracing libraries have been fully ported to Xeon Phi™

Make sure the necessary libraries are accessible to the card via:
- NFS-sharing the /opt/intel & $HOME directories (preferred)
- Manually copying the files:
  $ scp /opt/intel/itac/<version>/mic/slib/libVT.so mic0:/lib64

Now run as shown previously:
  $ mpirun -trace -n 2 ./test

Make sure you have all trace files in the same directory:
- If your card is NFS-shared, all trace files will be accessible in the directory where the executable is located (preferred)
- If your card is not NFS-shared, you might have partial trace files created on the Xeon Phi™ coprocessor that you need to copy:
  $ scp mic0:/home/<user>/test.stf.* node0:~

Finally view using the GUI:
  $ traceanalyzer test.single.stf &
Intel® Trace Analyzer and Collector

Chart showing how the MPI processes interact

Compare the event timelines of two communication profiles

Blue = computation
Red = communication
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 process/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/processes 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>TSelf</td>
<td>TSelf /</td>
<td>TTotal</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Group Threads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRECON</td>
<td>678.797,445 s</td>
<td>678.797,445 s</td>
<td>48.536</td>
</tr>
<tr>
<td>OMP_SYNC</td>
<td>580.473,944 s</td>
<td>580.473,944 s</td>
<td>286.320</td>
</tr>
<tr>
<td>MATMUL</td>
<td>410.463,131 s</td>
<td>410.463,131 s</td>
<td>48.260</td>
</tr>
<tr>
<td>SOLVER</td>
<td>328.400,819 s</td>
<td>328.400,819 s</td>
<td>218.943</td>
</tr>
<tr>
<td>User_Code</td>
<td>148.746,145 s</td>
<td>148.746,145 s</td>
<td>129.917</td>
</tr>
<tr>
<td>MPI_Comm</td>
<td>94.227,914 s</td>
<td>94.227,914 s</td>
<td>37.346</td>
</tr>
<tr>
<td>ASSEMBLY</td>
<td>42.822,701 s</td>
<td>42.822,701 s</td>
<td>32.346</td>
</tr>
<tr>
<td>MPI_BARRIER</td>
<td>24.271,402 s</td>
<td>24.271,402 s</td>
<td>49.312</td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>23.807,464 s</td>
<td>23.807,464 s</td>
<td>37.184</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>17.607,615 s</td>
<td>17.607,615 s</td>
<td>49.472</td>
</tr>
<tr>
<td>MPI_Comm_dup</td>
<td>11.759,694 s</td>
<td>11.759,694 s</td>
<td>32.546</td>
</tr>
<tr>
<td>MPI_Init</td>
<td>7.893,689 s</td>
<td>7.893,689 s</td>
<td>145.324</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>4.629,197 s</td>
<td>4.629,197 s</td>
<td>145.324</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>0.000,209 s</td>
<td>0.000,209 s</td>
<td>32.546</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>0.000,233 s</td>
<td>0.000,233 s</td>
<td>32.546</td>
</tr>
</tbody>
</table>

<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>TSelf</td>
<td>TSelf /</td>
<td>TTotal</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>Group Threads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI_Comm_dup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process 0</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 1</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 2</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 3</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 4</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 5</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 6</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 7</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 8</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 9</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 10</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 11</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 12</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 13</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 14</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 15</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 16</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
<tr>
<td>Process 17</td>
<td>0.013,388 s</td>
<td>0.013,388 s</td>
<td>1.546</td>
</tr>
</tbody>
</table>

Optimization Notice
Copyright© 2013, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.

intel.com/software/products
Call-Tree and Call-Graph

- Function statistics including calling hierarchy
  - Tree: call-stack
  - Graph: calling dependencies
Communication Profiles

- Statistics about point-to-point or collective communication
- Generic matrix supports grouping by several 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 & 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
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
8 MPI procs x 28 OpenMP threads

Too high load on Host = too low load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x
1 OpenMP thread

Coprocessor
24 MPI procs x
8 OpenMP threads

Too low load on Host = too high load on coprocessor
Improving Load Balance: Real World Case

Collapsed data per node and coprocessor card

Host
16 MPI procs x 1 OpenMP thread

Coprocessor
16 MPI procs x 12 OpenMP threads

Perfect balance
Host load = Coprocessor load
Ideal Interconnect Simulator (Idealizer)

Helps to figure out application's imbalance simulating its behavior in the "ideal communication environment"

Real trace

Ideal trace

Hot Spot

Easy way to identify application bottlenecks
Application Imbalance diagram

Intuitive diagram for simplified application analysis

Basic building block: breakdown of a single run time into 3 colors

Combined information in one location:
- Load Imbalance
- MPI overall time
- MPI Interconnect time
- Different Breakdowns etc

physical MPI (library, interconnect)
imbalance (idling in MPI; application intrinsic)
‘pure’ calculation

Simplified application analysis helps to identify performance issues
Application Imbalance diagram

Expanded View

Breakdown mode

Application Imbalance Diagram (Breakdown Mode)
Intel® Trace Analyzer and Collector 8.1 Update 3

What’s New

Fresh look-and-feel to the Intel® Trace Analyzer Graphical Interface

- New toolbars, icons, and dialogs for more streamlined analysis flow
- Addition of Welcome Page and easy access to past projects

Support of Dynamic Profiling Tool Command

- *MPI_PControl* supported

Support for MPI 2.x Standard

New GUI-based installer on Linux*
Tune MPI Apps Single Node Threading
Intel® VTune™ Amplifier XE Performance Profiler

Launch Intel® VTune™ Amplifier XE
• Use mpirun
• List your app as a parameter

Results organized by MPI rank

Review results
• Graphical user interface
• Command line report

Tune for Scalable Multicore Performance
Using the Intel® VTune™ Amplifier XE with MPI

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

$$\text{mpirun -n 4 amplxe-cl --result-dir ampl_results -collect hotspots -- .}/example.exe$$

A results directory is created for each MPI rank

- Can use arg sets to filter on a subset of ranks

Launch the GUI and view the results for each particular rank

$$\text{amplxe-gui ampl_results.<rank#>}$$
Intel® Cluster Studio XE correctness tools find errors early in the design cycle

Where are my application’s…

**Memory Errors**
- Invalid Accesses
- Memory Leaks
- Uninitialized Memory Accesses

**Threading Errors**
- Races
- Deadlocks
- Cross Stack References

**Security Errors**
- Buffer overflows and underflows
- Incorrect pointer usage
- Over 250 error types...

- MPI aware, cluster friendly
- Both dynamic and static analysis
- Multiple tools – common GUI
- Windows* & Linux*

“Having such a tool this early in the development stage frees the validation from trivial bug reports and gives our engineers the opportunity to code more efficiently from the very beginning of the product cycle.”

Jean Kypreos
Advanced Video Processing Team Manager
Envivio

Multi-threading problems are hard to reproduce, difficult to debug and expensive to fix!
MPI Correctness Checking: automatically checks MPI correctness

Solves two problems:
1. Finding programming mistakes in MPI application which need to be fixed by the application developer.
2. Detecting errors in the execution environment.

Two aspects:

1. *error detection* – done *automatically* by the tool

2. *error analysis* – manually by the user based on
   - information provided about an error
   - knowledge of source code, system, ...
MPI Correctness Checking

*How it 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.
MPI Correctness Checking

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()
MPI Correctness Checking

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**
MPI Correctness Checking: Usage (Part I)

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
```
MPI Correctness Checking: Usage (Part II)

Enable correctness checking info to be added to the trace file:
- Enable VT_CHECK_TRACING env 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.).
MPI Correctness Checking

*Debugger Integration*

Debugger must be in control of application before error is found

A breakpoint must be set in `MessageCheckingBreakpoint()`

Can be done automatically by configuring the debugger, instructions for TotalView, gdb and idb contained in documentation.
MPI Correctness Checking

Usage of Debugger

error detected, process stopped at breakpoint

access to MPI parameters in wrapper

full access to application source code and data
Intel® Cluster Studio XE Correctness Tools
Analyze MPI Apps For Memory, Threading and Security Errors

**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

**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 the Intel® Inspector XE with MPI

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

```bash
$ mpirun -n 4 inspxe-cl --result-dir insp_results -collect mil -- ./insp_example.exe
```

A results directory is created for each MPI rank

- Can use arg sets to filter on a subset of ranks

Launch the GUI and view the results for each particular rank

```bash
$ inspxe-gui insp_results.<rank#>
```
Intel® MPI Benchmarks 3.2.4

Overview and What's New

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

Enhancements:

Support for the Intel® Xeon Phi™ Coprocessor

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