Message Passing Interface (MPI) on Intel® Xeon Phi™ coprocessor

Special considerations for MPI on Intel® Xeon Phi™ and using the Intel® Trace Analyzer and Collector

Gergana Slavova

gergana.s.slavova@intel.com
Technical Consulting Engineer
Intel Software and Services Group
Outline

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Outline

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Enabling & Advancing Parallelism
High Performance Parallel Programming

Intel tools, libraries and parallel models extend to multicore, many-core and heterogeneous computing.

Use One Software Architecture Today. Scale Forward Tomorrow.
Intel® MPI Library Overview

Intel is the leading vendor of MPI implementations and tools

Optimized MPI application performance
  • Application-specific tuning
  • Automatic tuning

Lower latency
  • Industry leading latency

Interconnect Independence & Runtime Selection
  • Multi-vendor interoperability
  • Performance optimized support for the latest OFED capabilities through DAPL 2.0

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

Intel® MPI library eliminates the need to develop, maintain, and test applications running on multiple fabrics.
Intel® Xeon Phi™ Coprocessor Becomes a Network Node

Intel® Xeon® Processor

Intel® Xeon Phi™ Coprocessor

Virtual Network Connection

Intel® Xeon® Processor

Intel® Xeon Phi™ Coprocessor

Virtual Network Connection

Intel® Xeon Phi™ Architecture + Linux enables IP addressability
Levels of communication speed

Current clusters are not homogenous regarding communication speed:

• Inter node (InfiniBand*, Ethernet, etc)
• Intra node
  - Inter sockets (Quick Path Interconnect)
  - Intra socket

Two additional levels to come with Intel® Xeon Phi™ coprocessor:

• Host-coprocessor communication
• Inter coprocessor communication
Selecting network fabrics

The Intel® MPI Library automatically selects the best available network fabric it can find

• Use \texttt{I\_MPI\_FABRICS} to select a different communication device explicitly

Available for Intel® Xeon Phi™ coprocessor:

• \texttt{shm}, \texttt{tcp}, \texttt{ofa}, \texttt{dapl}

• Availability checked in the order \texttt{shm:dapl}, \texttt{shm:ofa}, \texttt{shm:tcp} (intra:inter)
Outline

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Installation

Download latest Intel® MPI Library plus a license from Intel Registration Center (included in Intel® Cluster Studio XE)

l_mpi_p_4.1.1.036.tgz
l_itac_p_8.1.2.033.tgz

Unpack the tar file, and execute the installation script:

# tar zxf l_mpi_p_4.1.1.036.tgz
# cd l_mpi_p_4.1.1.036
# ./install.sh
- Follow the installation instructions

Root or user installation possible!

Resulting directory structure has intel64 and coprocessor sub-directories:

/opt/intel/impi/4.1.1.036/intel64/{bin,etc,include,lib}
/opt/intel/impi/4.1.1.036/mic/{bin,etc,include,lib}
- Only one user environment setup required, serves both architectures!
Prerequisites

Make sure the Intel® Xeon Phi™ card is accessible through the network (meaning, unique IP address)

Upload the Intel® MPI libraries onto the card(s)

```bash
# scp /opt/intel/impi/4.1.1.036/mic/bin/* node0-mic0:/bin
# scp /opt/intel/impi/4.1.1.036/mic/lib/* node0-mic0:/lib64
```

- Execute as root or user with sudo rights

- If you don’t have access to “/“ directory on the card, simply copy them to your home directory

- Alternatively, install the libraries in an NFS-shared space where the card is able to access them
Prerequisites per User

Set the compiler environment

```bash
# source <compiler_installldir>/bin/compilervars.sh intel64
```
- Identical for Host and coprocessor

Set the Intel® MPI Library environment

```bash
# source /opt/intel/impi/4.1.1.036/bin64/mpivars.sh
```
- Identical for Host and coprocessor

`mpirun` needs ssh access to Intel® Xeon Phi™ coprocessor
Compiling and Linking for Intel® Xeon Phi™ Coprocessor

Compile MPI sources using the Intel® MPI Library scripts (C/C++)

• For Xeon (including any potential offload code)
  
  # mpiicc -o test test.c

• For native execution on the coprocessor add the “–mmic” flag, i.e. the usual compiler flag controls also the MPI compilation
  
  # mpiicc -mmic -o test test.c

Linker verbose mode “-v” shows

• Without “–mmic” linkage with intel64 libraries:
  
  ld ... -L/opt/intel/impi/4.1.1.036/intel64/lib ...

• With “–mmic” linkage with coprocessor libraries:
  
  ld ... -L/opt/intel/impi/4.1.1.036/mic/lib ...
Outline

Overview

Installation of Intel® MPI

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
**Coprocesor only Programming Model**

- MPI ranks on Intel® Xeon Phi™ coprocessor (only)
- All messages into/out of the coprocessors
- Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads used directly within MPI processes

Build Intel Xeon Phi coprocessor binary using the Intel® compiler

Upload the binary to the Intel Xeon Phi coprocessor

Run instances of the MPI application on Intel Xeon Phi coprocessor nodes
Coprocessor only Programming Model

- MPI ranks on the Intel® Xeon Phi™ coprocessor(s) only
- MPI messages into/out of the coprocessor(s)
- Threading possible

Build the application for the Intel® Xeon Phi™ Architecture

```
# mpiicc -mmic -o test_hello.MIC test.c
```

Upload the coprocessor executable

```
# scp ./test_hello.MIC node0-mic0:/tmp
```

Remark: If NFS available no explicit uploads required!

Launch the application on the coprocessor from host

```
# export I_MPI_MIC=enable
# cat mpi_hosts
node0-mic0
# mpirun -f mpi_hosts -n 2 /tmp/test_hello.MIC
```

Alternatively: login to the coprocessor and execute the already uploaded mpirun there!

Symmetric Programming Model

MPI ranks on Intel® Xeon Phi™ Architecture and host CPUs

Messages to/from any core

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

Heterogeneous network of homogeneous CPUs

Build binaries by using the resp. compilers targeting Intel 64 and Intel Xeon Phi Architecture

Upload the binary to the Intel Xeon Phi coprocessor

Run instances of the MPI application on different mixed nodes
Symmetric Programming Model

- MPI ranks on the coprocessor(s) and host CPU(s)
- MPI messages into/out of the coprocessor(s) and host CPU(s)
- Threading possible

Build the application for Intel®64 and the Intel® Xeon Phi™ Architecture separately

```bash
# mpiicc -o test_hello test.c
# mpiicc -mmic -o test_hello.MIC test.c
```

Upload the Intel® Xeon Phi™ coprocessor executable

```bash
# scp ./test_hello.MIC node0-mic0:~/test_hello
   Remark: If NFS available no explicit uploads required (look for tips on next slide)!
```

Launch the application on the host and the coprocessor from the host

```bash
# export I_MPI_MIC=enable
# cat mpi_hosts
node0
node0-mic0
# mpirun -f mpi_hosts -n 2 ~/test_hello
```
Utilize the POSTFIX env variable

Support for NFS-shared cards

Compile your code for the Intel® Xeon® processor node

```bash
# mpiicc -o test test.c
```

And for Intel® Xeon Phi™ Coprocessor

```bash
# mpiicc -mmic -o test.mic test.c
```

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

```bash
# export I_MPI_MIC_POSTFIX=.mic
```

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

```bash
# export I_MPI_MIC=1
```

Run your application (from the Intel Xeon processor host)

```bash
# mpirun -f mpi_hosts -n 4 test
```
- When the `test` executable is run on a Intel Xeon Phi coprocessor, the `mpirun` script will automatically add the `.mic` postfix
Utilize the **PREFIX** env variable

*Support for NFS-shared cards*

Compile your code for the Intel® Xeon® Architecture

```
# mpiicc -o test test.c
```

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

```
# mpiicc -mmic -o ./MIC/test test.c
```

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

```
# export I_MPI_MIC_PREFERENCE=./MIC/
```

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

```
# export I_MPI_MIC=1
```

Run your application (from the Intel Xeon processor host)

```
# mpirun -f mpi_hosts -n 4 test
```

- When the `test` executable is run on a Intel Xeon Phi coprocessor, the `mpirun` script will automatically add the `./MIC/` prefix
MPI+Offload Programming Model

MPI ranks on Intel® Xeon® processors (only)

All messages into/out of host CPUs

Offload models used to accelerate MPI ranks

Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® Xeon Phi™ coprocessor

Build Intel® 64 executable with included offload by using the Intel compiler

Run instances of the MPI application on the host, offloading code onto coprocessor

Advantages of more cores and wider SIMD for certain applications
**MPI+Offload Programming Model**

MPI ranks on the host CPUs only

MPI messages into/out of the host CPUs

Intel® Xeon Phi™ Architecture as an accelerator

Compile for MPI and internal offload

```
# mpiicc -o test_hello test.c
```

Latest compiler compiles by default for offloading if offload construct is detected!

- Switch off by `-no-offload` flag

Execute on host(s) as usual

```
# cat mpi_hosts
node0

# mpirun -f mpi_hosts -n 2 ~/test_hello
```

MPI processes will offload code for acceleration
Outline

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Traditional Cluster Computing

- MPI is »the« portable cluster solution
- Parallel programs use MPI over cores inside the nodes
  - Homogeneous programming model
  - "Easily" portable to different sizes of clusters
  - No threading issues like »False Sharing« (common cache line)
  - Maintenance costs only for one parallelization model
Traditional Cluster Computing (cont’d)

Hardware trends
- Increasing number of cores per node - plus cores on co-processors
- Increasing number of nodes per cluster

Consequence: Increasing number of MPI processes per application

Potential MPI limitations
- Memory consumption per MPI process, sum exceeds the node memory
- Limited scalability due to exhausted interconnects (e.g. MPI collectives)
- Load balancing is often challenging in MPI
Hybrid Computing

Combine MPI programming model with threading model

Overcome MPI limitations by adding threading:
  - Potential memory gains in threaded code
  - Better scalability (e.g. less MPI communication)
  - Threading offers smart load balancing strategies

Result: Maximize performance by exploitation of hardware (including coprocessors)
Options for Thread Parallelism

- Intel® Math Kernel Library
- OpenMP*
- Intel® Threading Building Blocks
  Intel® Cilk™ Plus
- Pthreads* and other threading libraries

Choice of unified programming to target Intel® Xeon® and Intel® Xeon Phi™ Architecture!
Intel® MPI Library Support of Hybrid Codes

Define \texttt{I\_MPI\_PIN\_DOMAIN} to split logical processors into non-overlapping subsets

Mapping rule: 1 MPI process per 1 domain

Pin OpenMP threads inside the domain with \texttt{KMP\_AFFINITY} (or in the code)
Intel® MPI Library Environment Support

The execution command `mpirun` of the Intel® MPI Library reads argument sets from the command line:

- Sections between " : " define an argument set (or you can define a config file)
- Host, number of nodes, but also environment can be set independently in each argument set

```bash
# mpirun -env I_MPI_PIN_DOMAIN 4 -host myXEON ...
: -env I_MPI_PIN_DOMAIN 16 -host myMIC
```

Adapt the important environment variables to the architecture

- `OMP_NUM_THREADS`, `KMP_AFFINITY` for OpenMP
- `CILK_NWORKERS` for Intel® Cilk™ Plus

* Although locality issues apply as well, multicore threading runtimes are by far more expressive, richer, and with lower overhead.
Outline

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Intel® Trace Analyzer and Collector

Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact
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
- Comparison of multiple profiles
- Powerful aggregation and filtering functions
- Fail-safe MPI tracing
- Provides API to instrument user code
- MPI correctness checking
- Idealizer
Intel® Trace Analyzer and Collector
Prerequisites

Upload the tracing library manually

```
# scp /opt/intel/itac/8.1.2.033/mic/slib/libVT.so \
   node0-mic0:/lib64/libVT.so
```

Set the Intel® Trace Analyzer and Collector environment (per user)

```
# source /opt/intel/itac/8.1.2.033/bin/itacvars.sh
```

- Identical for Host and coprocessor
Intel® Trace Analyzer and Collector Usage with Intel® Xeon Phi™ coprocessor

Recommended

Run with –trace flag (without linkage) to create a trace file

- MPI+Offload
  
  ```bash
  # mpirun -trace -n 2 ./test
  ```

- Coprocessor only and Symmetric
  
  ```bash
  # export I_MPI_MIC=enable
  
  # mpirun -trace -f mpi_hosts -n 2 ~/test_hello[.MIC]
  ```

Flag “-trace” will implicitly pre-load the libVT.so (which finally calls libmpi.so to execute the MPI call)
Intel® Trace Analyzer and Collector Usage with Intel® Xeon Phi™ coprocessor

Compilation Support

Compile and link with the “–trace” flag

# mpiicc -trace -o test_hello test.c
# mpiicc -trace -mmic -o test_hello.MIC test.c

- Linkage of tracing library

or, Compile with –tcollect flag

# mpiicc -tcollect -o test_hello test.c
# mpiicc -tcollect -mmic -o test_hello.MIC test.c

- Linkage of tracing library
- Will do full instrumentation of your code, i.e. All user functions will be visible in the trace file
- Maximal insight, but also maximal overhead

or, Use the tracing API to manually instrument your code

Run your Intel® MPI program as per usual, without “–trace” flag

# mpirun -f mpi_hosts -n 2 ~/test_hello[.MIC]
Intel® Trace Analyzer and Collector Analysis

Start the Intel® Trace Analyzer and Collector analysis GUI with the trace file as an argument (or load it through the menu later)

# traceanalyzer test_hello.single.stf
Outline

Overview

Installation of Intel® MPI Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Intel® Xeon Phi™ Coprocessor Becomes a Network Node

Intel® Xeon® Processor

Intel® Xeon® Processor

Intel® Xeon Phi™ Coprocessor

Intel® Xeon® Processor

Intel® Xeon Phi™ Coprocessor

Virtual Network Connection

Virtual Network Connection

Intel® Xeon Phi™ Architecture + Linux enables IP addressability
Load Balancing

Situation
• Host and coprocessor computation performance are different
• Host and coprocessor internal communication speed is different

MPI in symmetric mode is like running on a heterogenous cluster

Load balanced codes (on homogeneous cluster) may get imbalanced!

Solution? No general solution!
• Approach 1: Adapt MPI mapping of (hybrid) code to performance characteristics: \#m processes per host, \#n process per coprocessor(s)
• Approach 2: Change code internal mapping of workload to MPI processes
  – Example: uneven split of calculation grid for MPI processes on host vs. coprocessor(s)
• Approach 3: ...

Analyze load balance of application with Intel® Trace Analyzer and Collector
• Ideal Interconnect Simulator
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)

What is the Ideal Interconnect Simulator?

Using an Intel® Trace Analyzer and Collector trace of an MPI application, simulate it under ideal conditions

- Zero network latency
- Infinite network bandwidth
- Zero MPI buffer copy time
- Infinite MPI buffer size

Only limiting factors are concurrency rules, e.g.,

- A message cannot be received before it is sent
- An All-to-All collective may end only when the last thread starts
Ideal Interconnect Simulator (Idealizer)
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / Early Receive
Building Blocks: Elementary Messages

Early Send / Late Receive

Late Send / EarlyReceive
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
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Legend:
257 = MPI_Alltoallv
506 = User_Code

Same timescale in both figures
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
257 = MPI_Alltoallv
506 = User_Code
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
257 = MPI_Alltoallv
506 = User_Code

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

Overview

Installation of the Intel® MPI Library

Programming Models

Hybrid Computing

Intel® Trace Analyzer and Collector

Load Balancing

Debugging
Debugging an Intel® MPI Library Application

Use environment variables:

• **I_MPI_DEBUG** to set the debug level
• **I_MPI_DEBUG_OUTPUT** to specify a file for output re-direction
  – Use format strings like %r, %p or %h to add rank, pid or host name to the file name accordingly

Usage:

  ```
  # export I_MPI_DEBUG=<debug level>
  
  or:

  # mpirun -env I_MPI_DEBUG <debug level>
  -n <# of processes> ./a.out
  ```

Processor information utility in the Intel® MPI Library:

  ```
  # cpuinfo
  - Aggregates /proc/cpuinfo information
  ```
Debugging an Intel® MPI Library Application (cont’d)

Or use any of the variety of parallel debuggers we support

Compile the MPI source with symbols (compiler flag “-g”)

    # mpiicc -g test.c -o hello_test

Run your Intel® MPI Library application under a debugger:

• Using the Intel Debugger, simply start idb as the MPI executable and run the real application under debugger control:
    # mpirun -n 2 idb ./hello_test

• Using the GNU* Debugger
    # mpirun -gdb -n 2 ./hello_test

• Using the Totalview* debugger
    # mpirun -tv -n 2 ./hello_test
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
Summary

The ease of use of the Intel® MPI Library and related tools like the Intel® Trace Analyzer and Collector extends from the Intel® Xeon® architecture to the Intel® Xeon Phi™ architecture.
Thank You for Attending
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 © , 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.

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