MPI Programming on the Intel® Xeon Phi™ Coprocessor

Dr.-Ing. Michael Klemm
Software and Services Group
Intel Corporation
(michael.klemm@intel.com)
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 www.intel.com/software/products.

Copyright © 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, Phi, 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.

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

Notice revision #20110804
Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing
Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing
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.
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.
Preserve Your Development Investment
Common Tools and Programming Models for Parallelism

Develop Using Parallel Models that Support Heterogeneous Computing
Intel® MPI Library Overview

- Intel is a 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
Levels of communication speed

• Current clusters are not homogenous regarding communication speed:
  – Inter node (Infiniband, Ethernet, etc)
  – Intra node
    o Inter sockets (Quick Path Interconnect)
    o Intra socket

• Two additional levels to come with Intel® Xeon Phi™ coprocessor:
  – Host-coprocessor communication
  – Inter coprocessor communication
Intel® MPI Library Architecture & Staging

Application

MPI-2.2

MPICH2* upper layer

ADI3*

CH3* device layer

Nemesis*

CH3*

Netmod*

shm

tcp

Pre-Alpha

mpi(2)

user SCIF†

kernel SCIF

Alpha

dapl, ofa

Beta/Gold

OFED verbs/core

HCA‡ driver

†: Symmetric Communications Interface
‡: Host Channel Adapter

Intel® Xeon Phi™ Architecture
Software & Services Group, Developer Relations Division
Copyright© 2012, Intel Corporation. All rights reserved. *Other brands and names are the property of their respective owners.
Selecting network fabrics

- Intel® MPI selects automatically the best available network fabric it can find.
  - Use I_MPI_FABRICS to select a different communication device explicitly
- The best fabric is usually based on Infiniband (dapl, ofa) for inter node communication and shared memory for intra node.
- Available for Intel(R) Xeon Phi(TM):
  - shm, tcp, ofa, dapl
  - Availability checked in the order shm:dapl, shm:ofa, shm:tcp (intra:inter)
- Set I_MPI_SSHM_SCIF=1 to enable shm fabric between host and Intel® Xeon Phi™
Agenda

• Overview
• Programming Models
• Hybrid Computing
• Load Balancing
Coprocessor only Programming Model

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

How to use Intel® Xeon Phi™ coprocessors:

1. Build Intel® Xeon Phi™ binary using Intel® compiler.
2. Upload the binary to the Intel® Xeon Phi™.
3. Run instances of the MPI application on Intel® Xeon Phi™ 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
  
  ```
  # sudo micput 172.31.1.1 ./test_hello.MIC /tmp/test_hello.MIC
  ```
  – Remark: If NFS available no explicit uploads required (just copies)!

- Launch the application on the coprocessor from host
  
  ```
  # mpiexec -n 2 -wdir /tmp -host 172.31.1.1 /tmp/test_hello.MIC
  ```

- Alternatively: login to the coprocessor and execute the already uploaded mpiexec.hydra 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

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

Upload the Intel® Xeon Phi™ binary to the Intel® Xeon Phi™ Architecture.

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

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

- Upload the Intel® Xeon Phi™ executable

  # sudo micput 172.31.1.1 ./test_hello.MIC /tmp/test_hello.MIC

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

  # mpiexec -n 2 -host <hostname> ./test_hello : -wdir /tmp -n 2 -host 172.31.1.1 /tmp/test_hello.MIC
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™

Build Intel® 64 executable with included offload by using the Intel® 64 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 test.c

- Latest compiler compiles by default for offloading if offload construct is detected!
  - Switch off by `-no-offload` flag
  - Previous compilers needed `-offload-build` flag

- Execute on host(s) as usual
  
  # mpiexec -n 2 ./test

- MPI processes will offload code for acceleration
Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing
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

Ease of use / code maintainability

Programmer control

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

- Intel® MPI is strong in mapping control
- Sophisticated default or user controlled
  - `I_MPI_PIN_PROCESSOR_LIST` for pure MPI
  - For hybrid codes (takes precedence):
    - `I_MPI_PIN_DOMAIN = <size>[:<layout>]`
      - `<size>` =
        - `omp` Adjust to OMP_NUM_THREADS
        - `auto` #CPUs/#MPIprocs
        - `<n>` Number
      - `<layout>` =
        - `platform` According to BIOS numbering
        - `compact` Close to each other
        - `scatter` Far away from each other
- Naturally extends to hybrid codes on Intel® Xeon Phi™

*Although locality issues apply as well, multicore threading runtimes are by far more expressive, richer, and with lower overhead.*
Intel® MPI Support of Hybrid Codes

- Define **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 **KMP_AFFINITY** (or in the code)
Intel® MPI Environment Support

• The execution command mpiexec of Intel® MPI reads argument sets from the command line:
  – Sections between "::" define an argument set
    (also lines in a configfile, but not yet available in Beta)
  – Host, number of nodes, but also environment can be set independently in each argument set
    
    # mpiexec –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.
Co-Processor only and Symmetric Support

• Full hybrid support on Intel® Xeon from Intel® MPI extends to Intel® Xeon Phi(TM)
• **KMP_AFFINITY=balanced** (only on coprocessor) in addition to scatter and compact

• Recommendations:
  – Explicitly control where MPI processes and threads run in a hybrid application (according to threading model and application)
  – Avoid splitting cores among MPI processes, i.e. **I_MPI_PIN_DOMAIN** should be a multiple of 4
  – Try different **KMP_AFFINITY** settings for your application
MPI+Offload Support

• Define thread affinity manually per single MPI process (pseudo syntax!):
  
  # export OMP_NUM_THREADS=4
  
  # mpiexec -env KMP_AFFINITY=[1-4] -n 1 -host myMIC ... :
  -env KMP_AFFINITY=[5-8] -n 1 -host myMIC ... :
  ...

Agenda

- Overview
- Programming Models
- Hybrid Computing
- Load Balancing
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
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
    o Example: uneven split of calculation grid for MPI processes on host vs. coprocessor(s)
  – Approach 3: ...

• Analyze load balance of application with ITAC
  – 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
What is the Ideal Interconnect Simulator (IIS)?

- Using a ITAC 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 can not 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

Zero duration

Load imbalance
Building Blocks: Collective Operations

Actual trace (Gigabit Ethernet)

Simulated trace (Ideal interconnect)

Same timescale in both figures

Legend:
257 = MPI_Alltoallv
506 = User_Code

Same MPI_Alltoallv

Gigabit Ethernet

Ideal interconnect
Application Imbalance Diagram: Total

- Faster network
- Change parallel decomposition
- Change algorithm
- "interconnect"
- "load imbalance"
- "calculation"

MPI
Application Imbalance Diagram: Breakdown

Interconnect

Load imbalance

MPI_Recv

MPI_Allreduce

MPI_Alltoallv
Questions?