MPI Programming on Intel® Xeon Phi™ coprocessor

Presenter: Zhou Shan SSG/DRD/CRT
Contributor: SSG/DRD SSG/DPD
Legal Disclaimers

- All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Knights Ferry, Knights Corner, Ivy Bridge, Romley, Sandy Bridge, Westmere, Nehalem, Harpertown, and certain other names are code names used to identify unreleased Intel products. Intel makes no warranty of trademark non-infringement, and use of these code names by third parties is at their own risk.
- Intel may make changes to specifications, product descriptions, and plans at any time, without notice.
- INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.
- Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.
- This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.
- The Intel® Xeon® Processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Intel, the Intel logo, Intel® Virtualization Technology, Intel® I/O Acceleration Technology, Intel® VTune™ Analyzer, Intel® Thread Checker™, Intel® Tools, Intel® Trace Analyzer and Collector and Intel® Xeon™ are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Hyper-Threading Technology requires a computer system with a processor supporting HT Technology and an HT Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. For more information including details on which processors support HT Technology, see here.
- “Intel® Turbo Boost Technology requires a PC with a processor with Intel Turbo Boost Technology capability. Intel Turbo Boost Technology performance varies depending on hardware, software and overall system configuration. Check with your PC manufacturer on whether your system delivers Intel Turbo Boost Technology. For more information, see http://www.intel.com/technology/turboboost.”
- Intel® Virtualization Technology requires a computer system with an enabled Intel® processor, BIOS, virtual machine monitor (VMM) and, for some uses, certain computer system software enabled for it. Functionality, performance or other benefits will vary depending on hardware and software configurations and may require a BIOS update. Software applications may not be compatible with all operating systems. Please check with your application vendor.
Objectives

• Introduce MPI programming models on Intel® Xeon Phi™ product

• Basic implementation and fabric support information on Intel® Xeon Phi™ product
Agenda

• Overview
• MPI Programming Models
• MPI Communication Mechanisms
• Intel® Trace Analyzer and Collector
• Load Balance
• Summary
Intel® MPI Library Overview

- 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® MIC Architecture Programming

**Programming Models**

- **Multi-Core**
- **Many-Core**
- **Multi-core + Many-Core**

For illustration only, potential future options subject to change without notice.

**Common with Intel® Xeon® processors**

- Programming Models
- C/C++, Fortran compilers
- Intel® SW developer tools and libraries including Intel® Math Kernel Library, Intel® Integrated Performance Primitives and Intel® Threading Building Blocks
- Coding and optimization techniques and SW tools
- Ecosystem support

**Eliminates Need for Dual Programming Architecture**
Agenda

• Overview
• **MPI Programming Models**
• MPI Communication Mechanisms
• Intel® Trace Analyzer and Collector
• Load Balance
• Summary
Programming Models-Overview

**MPI+Offload**
- MPI ranks on Xeon only
- Messages into/out of Xeon
- MIC as accelerator
- Simplest

**Co-processor-only**
- MPI ranks on several MIC cards only
- Messages into/out of MIC thru Xeon

**Symmetric**
- MPI ranks on several MIC cards and Xeons
- Messages to/from any core
MPI+Offload Programming Model

- MPI ranks on Xeon processor(s) only
- All messages into/out of Xeon processors
- Offload models used to accelerate MPI ranks
- Intel® Cilk™ Plus, Intel® TBB, OpenMP, Pthreads used within MIC

Suitable to start MIC enabling in HPC
- Seamless movement to heterogeneous computing
- Advantages of more cores and wider SIMD for certain applications
MPI+offload Programming Model

• Build the application with MPI+offload mode
  
  # mpiicc -o test test.c

• The code will be compiled by default for offload if offload construct is detected by compiler
  
  # switch off by adding "-no-offload" option when compiling

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

• MPI processes will offload code to MIC card(s)
Co-processor-only Programming Model

- MPI ranks on MIC co-processor(s) only
- All messages into/out of MIC thru Xeon
- Intel® Cilk™ Plus, Intel® TBB, OpenMP, Pthreads used directly within MPI processes
Co-processor-only Programming Model

• Build the application in Co-processor-only mode for MIC architecture
  
  # mpiicc -mmic -o test_MIC test.c

• Upload the MIC executable file to MIC card
  
  # sudo scp test_MIC mic0:/tmp/test_MIC

• Launch the application from host or login to the MIC card and launch it from the card by using mpiexec.hydra
  
  # mpiexec -n 2 -wdir /tmp -host mic0 /tmp/test_MIC
Symmetric Programming Model

- MPI ranks on MIC card(s) and Xeon Processor(s)
- Messages to/from any core, MIC card(s) or Host CPU(s)
- Intel® Cilk™ Plus, Intel® TBB, OpenMP, Pthreads used directly within MPI processes
Symmetric Programming Model

• Build the application for Xeon and MIC Co-processor separately
  
  # mpiicc -o test test.c
  # mpiicc -mmic -o test_MIC test.c

• Upload the MIC executable file to MIC card
  
  # sudo scp test_MIC mic0:/tmp/test_MIC

• Launch the application on the host and MIC card from host
  
  # mpiexec -n 2 -host <hostname> ./test
    : -wdir /tmp -n 2 -host mic0 /tmp/test_MIC
Agenda

• Overview
• MPI Programming Models
• **MPI Communication Mechanisms**
• Intel® Trace Analyzer and Collector
• Load Balance
• Summary
Communication Mechanisms

- **Multi-core cluster communication**:
  - Inter node, by Infiniband, Ethernet, etc
  - Intra node
    - Inter sockets (QPI)
    - Intra socket

- **Multi-core + Many-core cluster communication**
  - All of the communication mechanisms of multi-core cluster
  - Two additional communication
    - Communication between host and co-processor
    - Inter co-processors communication
**Intel® MPI Library Architecture & Staging**

- **MPI-2.2**
  - **MPICH2* upper layer**
  - **CH3* device layer**
  - **Nemesis***

**SCIF**: symmetric communications Interface
**HCA**: Host Channel Adapter
**OFA**: OpenFabrics Alliance
**DAPL**: Direct Access Programming Library

- **Pre-Alpha**
  - **shm**
  - **mmap(2)**

- **Alpha**
  - **TCP**
  - **User SCIF**
  - **Kernel SCIF**

- **Beta/Gold**
  - **DAPL, OFA**
  - **OFED verbs/core**
  - **HCA driver**

---

SCIF: symmetric communications Interface
HCA: Host Channel Adapter
OFA: OpenFabrics Alliance
DAPL: Direct Access Programming Library

Software & Services Group
Copyright © 2010, Intel Corporation. All rights reserved.
Network Fabrics for Heterogenous Cluster

- Default select the best available network fabric which can be found
- The best fabric is usually based on Infiniband (dapl, ofa) for inter node and shared memory (shm) for intra node
- Available fabrics for Intel® Xeon Phi™ coprocessor
  - shm, tcp, ofa, dapl
  - Availability checked in the order, shm:dapl, shm:ofa, shm:tcp (intra:inter)
- I_MPI_FABRICS can be used to select a different communication mechanism explicitly
- Set I_MPI_SSHM_SCIF=1 to enable shm fabric between host and MIC card
MPSS software to support MPI
OFED over SCIF

- OFA Software (OFED) is the industry standard code used for messaging on high-end HPC clusters (>60% of Top500)
  - Supports Intel MPI and all open source MPIs
  - Is in Linux and all the various Linux distros
- OFED for MIC uses the core OFA software modules from the Open Fabrics Alliance
- OFED/SCIF is a new hardware specific driver and library that plugs into the OFED core mid-layer
  - SCIF is the lowest level communications driver between the Xeon and the MIC cards
  - OFED/SCIF allows communication between the Xeon and the MIC or between MIC cards on the same system
MPSS software to support MPI
MIC- direct

- MIC-Direct allows MPI speed path operations direct access to a RDMA NIC, i.e., InfiniBand HCA, iWarp RNIC
  - MIC shares RDMA NIC with Host
  - No changes to MPIs are needed
- Communication setup operations are proxied to the Xeon, e.g, allocate QPs, register memory, map HCA registers
Pure MPI Programming

• MPI is the portable cluster solution
• Enlarge Cluster Scale and improve performance by
  - Increasing core number per node
  - Increasing node number
• Performance improved need more MPI processes
• Potential limitations
  - Memory requirement per MPI process, would exceeds the node memory
  - Limited scalability due to exhausted interconnects
  - Load imbalance
Hybrid programming - overview

- Combine MPI programming model with many multi-threading models
- Overcome MPI limitations by using multi-threads
  - Potential memory gains in threaded code
  - Better scalability, less MPI communication
  - Better load balance strategies
- Maximize performance by exploitation of hardware
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

Unified programming to support Intel® Xeon and Intel® Xeon Phi™ coprocessor
MPI Process Mapping

- By mapping MPI processes would cause impact on performance, especially for hybrid mode
- Intel® MPI provides many environment variable for programmer to control process pinning
  - For pure MPI: `I_MPI_PIN_PROCESSOR_LIST`
  - For hybrid code,
    - split logical processors into subsets,
    - pin omp threads inside domain with affinity set
      `I_MPI_PIN_DOMAIN=<size>[::<layout>]`
      - `<size>` =
        - `omp` Adjust to OMP_NUM_THREADS
        - `auto` #CPU/#MPI procs
        - `<n>` Number
      - `<layout>` =
        - `platform` According to BIOS number
        - `compact` close to each other
        - `scatter` far away from each other
- Also extends this support to MIC
MPI + Offload Support

• Avoid the interference between the offload from different MPI processes, such as
  - Don’t offload to the same card for different MPI process

- Using MPI+openMP and set thread affinity manually for every MPI process
  
  #export OMP_NUM_THREADS=4
  #mpiexec -env KMP_AFFINITY=[1,4] -n 1 -host mic0
  -env KMP_AFFINITY=[5,6] -n 1 -host mic0

  .......
Agenda

- Overview
- MPI Programming Models
- MPI Communication Mechanisms
- Intel® Trace Analyzer and Collector
- Load Balance
- Summary
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 on time and processors
  – Comparison of multiple profiles
  – Powerful aggregation and filtering functions
  – Fail-safe MPI tracing
  – Provides API to instrument user code
  – MPI correctness checking
  – Idealizer
Build Application with ITAC support

Co-processor-only mode
• Build MIC executable with `-trace`
  
  ```shell
  # mpiicc -mmic -trace -o test_MIC test.c
  ```
• Upload it to the MIC
• Run this binary remotely from the host
• Copy trace file (if no NFS mounted) from the card to a host system

Symmetric mode
• Build Intel®64 and MIC executables with `-trace`
  
  ```shell
  # mpiicc -mmic -trace -o test_MIC test.c
  
  # mpiicc -trace -o test test.c
  ```
• Upload the MIC executable to the card:
• Run the application from the host so that rank 0 will generate trace file

Offload mode
• Build Intel®64 executables with `-trace`
  
  ```shell
  # mpiicc -trace -o test test.c
  ```
• Run the application from the host to generate trace file
Run Application with ITAC support

Co-processor-only mode
- Run with -trace flag to create a trace file
  
  ```bash
  # mpiexec -trace -n 2 -wdir /tmp -host mic0 /tmp/test_MIC
  ```

Symmetric mode
- Run with -trace flag to create a trace file
  
  ```bash
  # mpiexec -trace -n 2 -host michost ./test:
  # -wdir /tmp -n 2 -host mic0 /tmp/test_MIC
  ```

Offload mode
- Run with -trace flag to create a trace file
  
  ```bash
  # mpiexec -trace -n 2 -o ./test
  ```

Set VT_LOGFILE_FORMAT=stfsingle to create a singletrace
ITAC Analysis

• Start the ITAC analysis GUI with the trace file (or load it)
  # traceanalyzer test.single.stf
• Start the analysis, usually by inspection of the Flat Profile (default chart), the Event Timeline, and the Message Profile
  - Select “Charts->Event Timeline”
  - Select “Charts->Message Profile”
  - Zoom into the Event Timeline
    Click into it, keep pressed, move to the right, and release the mouse
    See menu Navigate to get back
  - Right click the “Group MPI->Ungroup MPI”.
ITAC Profile Example

Compare the event timelines of two communication profiles

Blue = computation
Red = communication

Chart showing how the MPI processes interact
Intel® Trace Analyzer and Collector

Timeline of optimized application run

Timeline of initial application run

Comparison of function and process profile data

Works on systems from 2 processes to more than a thousand processes

Shorter RED bars mean less MPI traffic and increased performance

Network usage profile data for MPI messages
Scale Performance
Tune Hybrid Cluster MPI and Thread Performance

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

Tune single node threading
- Visualize thread behavior
- Evaluate thread load balancing
- Find thread sync. bottlenecks

Intel®
Trace Analyzer and Collector

Intel®
VTune™ Amplifier XE
Agenda

• Overview
• MPI Programming Models
• MPI Communication Mechanisms
• Intel® Trace Analyzer and Collector
• Load Balance
• Summary
Load Balance

- MPI in symmetric mode is like running on a heterogeneous cluster. Original load balanced codes may get imbalanced. Because
  - Host and coprocessor computation performance are different
  - Host and coprocessor internal communication speed is different
- There is 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 such as 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

- Collapsed data per node and coprocessor card
  - Host: 16 MPI ranks * 1 OMP thread
  - MIC: 8 MPI ranks * 28 OMP threads

Too high load on Host = too low load on coprocessor
Improving Load Balance (cont.)

- Collapsed data per node and coprocessor card
  - Host: 16 MPI ranks * 1 OMP thread
  - MIC: 24 MPI ranks * 8 OMP threads

Too low load on Host = too high load on coprocessor
Improving Load Balance (cont.)

- Collapsed data per node and coprocessor card
  - Host: 16 MPI ranks * 1 OMP thread
  - MIC: 16 MPI ranks * 12 OMP threads

Perfect balance
Host load = Coprocessor load
Hands On Lab
Thanks!

More detailed info can be got from
http://software.intel.com/mic-developer