Intra-MIC MPI Communication using MVAPICH2: Early Experience

Sreeram Potluri*    Karen Tomko+    Devendar Bureddy*

Dhabaleswar K. Panda*

*Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University

+Ohio SuperComputer Center
Outline

• Motivation

• Problem Statement

• Experience with MVAPICH2 on KNF

• Conclusion

• Future Work
Multi-core Era

- Multi-core architectures played a key role in achieving Petascale computing
- Addressed ILP wall, Power wall, Memory wall (through NUMA)
- Same issues as we move towards Exascale computing, only more substantial
- Consensus that heterogeneous architectures and hybrid computing will be part of the solution
Motivation

- Intel unveiled the Many Integrated Core (MIC) architecture
- Knights Ferry (KNF) and Knights Corner (KNC)
- Targeted towards High Performance Computing (HPC)
- Many low-power processor cores with hardware threads and wider vector units
- Based on x86 architecture
- Applications and libraries developed for multi-core architectures can run with minor or no modification
- However, will they deliver optimal performance out of the box?
- How much effort is required to tune them for the MIC architecture?
Programming Model

• MPI is the most popular programming model in the HPC domain

• Hybrid models being explored for heterogeneous architectures
  • MPI + OpenMP
  • MPI + CILK
  • MPI + OpenCL/CUDA

• MIC offers offload and native modes

• A plausible model – MPI processes with OpenMP/CILK for finer
  grained parallelism (symmetric and many-core hosted modes)

• Performance of MPI continues to be important – Intra-MIC, MIC-Host,
  MIC-MIC
MVAPICHH/MVAPICHH2 Software

- High Performance MPI Library for IB, 10GigE/iWARP and RoCE
  - MVAPICHH (MPI-1) and MVAPICHH2 (MPI-2.2), available since 2002
  - Used by more than 1,880 organizations (HPC centers, Industries and Universities) in 66 countries
  - More than 105,000 downloads from OSU site directly
  - Empowering many TOP500 clusters
    - 5th ranked 73,278-core cluster (Tsubame 2.0) at Tokyo Institute of Technology
    - 7th ranked 111,104-core cluster (Pleiades) at NASA
    - 25th ranked 62,976-core cluster (Ranger) at TACC
    - 39th ranked 22,656-core cluster (Lonestar) at TACC
  - Partner in the upcoming U.S. NSF-TACC Stampede (10-15 PFlop) System
  - Available with software stacks of many IB, HSE and server vendors, and Linux Distros (RedHat and SuSE)
  - http://mvapich.cse.ohio-state.edu
Outline

• Motivation

• Problem Statement

• Experience with MVAPICH2 on KNF

• Conclusion

• Future Work
Knights Ferry

- Placement of cores and memory hierarchy
Existing Intra-Node Designs in MVAPICH2

• Uses different protocols and designs based on message size
  • Short messages
    • Pair-wise shared-memory buffers between processes
    • Eager protocol
  • Large messages
    • Each process maintains a common pool of fixed size buffers
    • Rendezvous protocol
• Performance of these designs depends on various parameters
  • Total number of buffers, size of each buffer and more . . .
  • Vary across different platforms
Problem Statement

• Can the MVAPICH2 library run “out of the box” on a KNF and how will it perform?
• How does tuning improve the performance of MVAPICH2 on a KNF?
• Will designs using low level experimental interface benefit MVAPICH2?
• Performance analysis:
  • Impact of Affinity
  • Point-to-point communication
  • Multi-pair communication
  • Collective communication
Outline

- Introduction
- Problem Statement
- Experience with MVAPICH2 on KNF
- Conclusion
- Future Work
Experimental Setup

• Host
  • Dual socket node with Intel Westmere six-core processors
  • Running at 3.33 GHz and 24GB of memory
  • Linux kernel 2.6.32

• KNF co-processor - connected via PCIe 2.0
  • DO 1.20GHz card with 32 cores
  • Alpha 9 Intel MIC software stack with an additional pre-alpha patch
MVAPICH2 and Benchmarks

- Variations of MVAPICH2 1.8a2
  - Default – Out of the box version
  - Optimized V1 – Shared memory designs tuned for KNF
  - Optimized V2 – Design using Intel’s lower level API

- OSU Micro Benchmarks (OMB) 3.5
- Intel Micro Benchmarks (IMB) 3.2
Impact of Affinity
Impact of Affinity
(Latency: lower is better)

Normalized Latency vs Message Size (Bytes)

- 2
- 5
- 33
- 65
- 97
- 125
- no-affinity
Point-to-Point Performance
Latency
(lower is better)

Normalized Latency vs Message Size (Bytes)

Default
Optimized V1
Optimized V2

Normalized Latency vs Message Size (Bytes)

Normalized Latency vs Message Size (Bytes)

70%
80%
Bandwidth
(higher is better)

- Default
- Optimized V1
- Optimized V2

Normalized BW vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

Message Size (Bytes)

- 1
- 4
- 16
- 64
- 256
- 1K

Normalized BW vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

Message Size (Bytes)

- 2K
- 8K
- 32K

Normalized BW vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

Message Size (Bytes)

- 64K
- 256K
- 1M
- 4M

Optimized V2 improves performance significantly compared to Default and Optimized V1.
Bi-directional Bandwidth
(higher is better)

Normalized BW vs Message Size (Bytes)

- Default
- Optimized V1
- Optimized V2

Comparison:
- Optimized V1 is 13% better than Default.
- Optimized V2 is 12% better than Optimized V1.
- Optimized V2 is 18X better than Default.
- Optimized V2 is 4X better than Optimized V1.

Message Size (Bytes): 1, 4, 16, 64, 256, 1K, 2K, 8K, 32K, 64K, 256K, 1M, 4M

Normalized BW: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

19
Multi-pair Latency

- 16 and 32 processes
- Rank \( r \) communicates with Rank \( (r + n/2)\%n \)
Multi-pair Latency – 16 Procs
(lower is better)

- Default
- Optimized V1
- Optimized V2

Normalized Latency vs. Message Size (Bytes)

- 0 to 1.0
- 0 to 1.0
- 0 to 1.0

Message Size (Bytes):
- 0, 2, 8, 32, 128, 512
- 2K, 8K, 32K
- 64K, 256K, 1M, 4M

Normalized Latency

% Improvement:
- 55%
- 72%
Multi-pair Latency – 32 Procs
(lower is better)

- Default
- Optimized V1
- Optimized V2

Normalized Latency vs. Message Size (Bytes)

- 64K
- 256K
- 1M
- 4M

31%
57%
Collective Communication

- 32 processes
- Similar trends with 4, 8 and 16 processes
Collective Communication – Broadcast
(lower is better)

Normalized Latency vs. Message Size (Bytes)

- Default
- Optimized V1
- Optimized V2

- 17% improvement
- 20% improvement
- 26% improvement
Collective Communication – Scatter

(lower is better)

- Default
- Optimized V1
- Optimized V2

Normalized Latency vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

- 0
- 2
- 8
- 32
- 128
- 512

Normalized Latency vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8

- 2K
- 8K
- 32K

Normalized Latency vs Message Size (Bytes)

- 0.0
- 0.2
- 0.4
- 0.6
- 0.8
- 1.0

- 64K
- 256K
- 1M

- 72%
- 80%
Outline

• Introduction
• Problem Statement
• Experience with MVAPICH2 on KNF
• Conclusion
• Future Work
Conclusion

• Early experience with MVAPICH2 on KNF
• Tuning is imperative to achieve good performance
  • Up to 70% reduction in latency
  • Up to 4X improvement in bandwidth and bi-bandwidth
• Using lower level API benefits large and asynchronous messaging
  • Up to 80% improvement in latency
  • Up to 9.5X improvement in bandwidth
  • Up to 18X improvement in bi-directional bandwidth
Future Work

• Does the selection of collective algorithms change for the new architecture?
• How do these enhancements impact application performance?

• Enhancing MVAPICH2 to support MIC-Host and MIC-MIC communication
• An integrated MVAPICH2 solution
  • Intra-MIC
  • MIC-Host
  • MIC-MIC (intra-node and inter-node)
Thanks to

Timothy C. Prince
Paul J. Besl
Linda L. Kenworthy
Intel Corporation
Thank You!

{potluri, bureddy, panda} @cse.ohio-state.edu
ktomko@osc.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page
http://mvapich.cse.ohio-state.edu/