Communication in a HPC cluster with MIC

Abhijit Vageeswara
Outline

- Intel MIC and their role in HPC
- Data Movement on Infiniband clusters with Intel MIC
- MVAPICH-PRISM
- Performance Evaluation
- Conclusion
Many Integrated Cores – MIC

- Run on standard, existing programming tools and methods
- Extremely power efficient
- Can be used in an Offload mode where application processes running on the host can offload compute to the coprocessor using compiler directives
- Also offers a Symmetric mode where application processes can directly run on both the coprocessor and the host
- Coprocessor-only mode is a subset of Symmetric mode, with all MPI processes being confined to the MIC architecture alone
Outline

• Intel MIC and their role in HPC
• Data Movement on Infiniband clusters with Intel MIC
• MVAPICH-PRISM
• Performance Evaluation
• Conclusion
Data Movement on Infiniband clusters with Intel MIC

- In addition to the expected communication in a HPC cluster, many additional levels of communication exist in MIC clusters
  - Between processes on the same MIC
  - Between processes on MIC and processes on host, within the same node
  - Between processes on MIC and processes on host, across nodes
  - Between processes on MICs, across nodes
Disparity in bandwidth when HCA is reading data from the MIC and when writing data to the MIC.
Challenges

- Design a high performance MPI library that can transparently by-pass inefficient communication paths on heterogeneous MIC clusters?
- Optimize communication operations involving MIC processes by efficiently utilizing resources on the host to stage the data transfers?
- Minimize the involvement of the host processors in implementing the data transfer operations
Symmetric Communication Interface Framework (SCIF)

- Sockets-like API for communication between processes on MIC and host within the same system
- SCIF API provides both send-receive semantics, as well as Remote Memory Access (RMA) semantics

Coprocessor Communication Link (CCL)

- Enables MIC to use IB directly and enables processes on the MIC to talk with the HCA
- It’s an IB proxy through which all privileged operations are staged through
- Resides on the host and make requests on behalf of the process running on the MIC
- Data movement calls from the process on the MIC can be made in a direct manner to the HCA using PCIe peer-to-peer copies
IB-SCIF

- Intel’s Manycore Platform Software (MPSS) implementation of IB verbs over SCIF API
- This allows processes to use verbs API over a virtual HCA as underlying operations are handled using SCIF
- Beneficial for MPI processes that reside in the MIC alone
Outline

- Intel MIC and their role in HPC
- Data Movement on Infiniband clusters with Intel MIC
- **MVAPICH-PRISM**
- Performance Evaluation
- Conclusion
Using CCL, MPI communication between processes running on a MIC and processes running on a remote Host or MIC. MPI transfer can be implemented as an IB transfer initiated by the mic-process. Provides a low latency path for small message communication. Limited by peer-to-peer bandwidth.
Passive Proxy

- Proxy is only involved in setting up staging buffers on the host and is not directly involved in any communication and resides on each node
- Achieve this by taking advantage of the RMA capabilities offered by SCIF and IB
- MIC processes establish a SCIF connection and an IB connection to the proxy running on the local host and requests buffer allocation
- Every process establishes an IB connection with the passive proxies running on all remote nodes in an on-demand fashion
Passive Proxy

When a mic-process has to send a message to a process on a remote MIC or host:

- It pulls a buffer from its local host buffer pool and writes the data into it, using an SCIF RMA
- It then sends that virtual address of the host buffer and the corresponding IB rkey to the remote process through Direct-IB
- The remote process can issue an IB RDMA operation to read data from the remote host
- Finally, remote process sends a finish message, when the read completes and the buffer is released back to the pool, at the sender process
Passive Proxy

- Important for the communication runtime to make simultaneous use of the PCIe and IB channels to achieve maximum performance. Achieve this by pipelining SCIF and IB operations.
Active Proxy

- A processor core is dedicated as the communication proxy to actively route messages from Xeon Phi through the host.
- Through this, we take advantage of the best communication channels possible.
- Two types
  - One Hop
  - Two Hop
One Hop Active Proxy

- The dedicated processor core allows the proxy to initiate and progress communication that is staged through the host.
- It also takes care of allocating and managing the staged buffers on the host.
- Similar to passive proxy in establishing connections but, proxy takes care of reading data from the local mic-process using SCIF and then writing the data into the remote process buffers using IB RDMA write.
- These two transfers are pipelined to utilize the PCIe and IB channels concurrently.
One Hop Active Proxy

- Since SCIF transfers are initiated from host, it delivers higher performance for medium messages.
- The overall performance is still bound by the bandwidth offered by the IB channel, as is the case with passive proxy.
Two Hop Active Proxy

- It is aimed at taking advantage of the high bandwidth channels between MIC and the host.
- For a transfer between a mic-process and a remote mic-process, the data is first staged to the local host then to the remote host and finally to the remote mic-process.
- The transfers are pipelined to minimize the impact of the additional hop.
- Still, additional hop can result in added latency and pipelining overheads.
Comparison between the four approaches
Outline

- Intel MIC and their role in HPC
- Data Movement on Infiniband clusters with Intel MIC
- MVAPICH-PRISM
- Performance Evaluation
- Conclusion
Inter node MIC to MIC Communication

Latency - Small Messages

Latency - Large Messages

Bandwidth

Bi-directional Bandwidth
Inter node MIC to Host Communication
All to all communication with 16 MIC coprocessors, up to 256 processes
Outline

- Intel MIC and their role in HPC
- Data Movement on Infiniband clusters with Intel MIC
- MVAPICH-PRISM
- Performance Evaluation
- Conclusion
Conclusion

- Provides an alternative methodology to communicate in MIC clusters alleviating bottlenecks
- Different approaches are optimal depending on message size, communication pattern, system resource
- Shows a huge improvement in MIC to MIC, MIC to host and collective communications
UPC on MIC: Early Experiences with Native and Symmetric Modes
Outline

- **UPC on MIC**
- Performance Evaluation
- Conclusion
UPC on MIC

- Highly multithreaded environment and the need for high programming productivity makes PGAS implementations such as UPC a natural fit for the MIC.

(a) Mapping Global Shared Memory Region on MIC for _native_ mode

(b) Connect MIC and host in _symmetric_ mode

UPC on MIC: _native_ and _symmetric_ modes
UPC on MIC

- UPC thread refers to an instance of execution for UPC applications
- Each UPC thread can be mapped to either an
  - OS process
  - OS thread
Process-based runtime

- OS processes can only share certain memory regions
- Intra-node memory access between two UPC threads realized through one of the two shared memory based schemes
  - Copy-via-shared-memory
  - Shared-memory-mapping
Copy-via-shared-memory

- Shared memory performs as intermediate buffer
  - Overhead from extra copy
  - Extra memory for intermediate buffer
  - Extra synchronization between source and destination processes

Shared-memory-mapping

- Local part of the shared memory region, which belongs to one process mapped into the address space of all the other processes
  - Huge amount of memory footprint in the kernel space.
Thread-based runtime

- All the UPC threads on the same node are mapped to OS threads spawned by a single OS process
- Memory access now done by system memory copy functions directly from the source buffer to destination buffer
- Potentially bad network performance due to sharing of network connections when hardware allows one connection per process
Multi-endpoint support with “leader-to-all” connection mode

- MIC-leader registers a shared memory region local to the whole MIC co-processor. Same is done in host.
- The two leaders exchange their rkeys of the pinned-down memory.
- The two rkeys are distributed to all the threads on MIC and host by the MIC-leader and HOST-leader.
- The number of connections between MIC and host is reduced to $NMIC + NHOST$. 
Multi-endpoint support with "leader-to-all" connection mode
Outline

- UPC
- UPC on MIC
- Performance Evaluation
- Conclusion
Performance Evaluation

Native Mode Point-to-Point Evaluations

Single Pair Latency

Multi Pair Latency
Native Mode Collective Evaluations

- **Bcast**
- **Gather**
- **Exchange**
Symmetric Mode Point-to-Point Evaluations
Outline

- UPC
- UPC on MIC
- Performance Evaluation
- Conclusion
Conclusion

- Discussed issues with running UPC applications on a MIC system under native and symmetric programming modes.

- Determined multi-threaded UPC runtime with the new proposed “leader-to-all” connection mode yields best performance.

- Found out several significant problems for UPC running on many-core system like MIC, such as the communication bottleneck between MIC and host, and the unbalanced physical memory and computation power issues.
Thank You

Questions?