InfiniBand - Towards Extreme Scale Support for MPI

Rich Graham
Solutions Architect – Mellanox
Overview

- Challenges for scaling MPI to Exascale
- Proposed InfiniBand enhancements
  - Dynamically Connected Transport (DC)
  - Cross-Channel synchronization
  - Non-contiguous data transfer
Challenges for Scaling MPI to Exascale

- Scalable Communication support
  - Collective communication
  - Point-to-point communication

- Support for truly asynchronous communication
  - Communication Computation overlap
  - Effective support for MPI RMA operations

- System noise issues
- Minimize data motion
- Ease access to the network
Dynamically Connected Transport
A Scalable Transport
New Transport

- Current status:
  - RC
    - High Performance: Supports RDMA and Atomic Operations
    - **Scalability limitations**: One connection per destination
  - UD
    - Scalable: One QP services multiple destinations
    - **Limited communication support**: No support for RDMA and Atomic Operations, unreliable

- Need scalable transport that also supports RDMA and Atomic operations ➔ DC – The best of both worlds
  - **High Performance**: Supports RDMA and Atomic Operations, Reliable
  - **Scalable**: One QP services multiple destinations
IB Reliable Transports Model

- QoS/Multipathing: 2 to 8 times the above
- Resource sharing (XRC/RD) causes processes to impact each-other

Example $n=4K$ $p=16$
On the Way to Exascale – Scalability Challenge

Host Memory Consumption (MB)

- 8 nodes
- 2K nodes
- 10K nodes
- 100K nodes

InfiniHost, RC 2002
InfiniHost-III, SRQ 2005
ConnectX, XRC 2008

© 2013 Mellanox Technologies
Dynamic Connectivity

- Each DC Queue can be used to reach any remote process
- No resources' sharing between processes
  - process controls how many (and can adapt to load)
  - process controls usage model (e.g. SQ allocation policy)
  - no inter-process dependencies
- Resource footprint
  - Function of HCA capability
  - Independent of system size
- Fast Setup Time

\[ cs \times cr \times \left( \frac{cs + cr}{2} \right) \]

*cs* – concurrency of the sender
*cr* – concurrency of the responder
ConnectIB – Exascale Scalability

Host Memory Consumption (MB)

- InfiniHost, RC 2002
- InfiniHost-III, SRQ 2005
- ConnectX, XRC 2008
- Connect-IB, DCT 2012

- 8 nodes
- 2K nodes
- 10K nodes
- 100K nodes

© 2013 Mellanox Technologies
- Mellanox Confidential -
Dynamically Connected Transport

- **Key objects**
  - DC Initiator: Initiates data transfer
  - DC Target: Handles incoming data
Reliable Connection Transport Mode
Dynamically Connected Transport Mode
Cross-Channel Synchronization (CORE-Direct)
High Level Objectives

- Provide synchronization mechanism between QP’s
- Provide mechanisms for communication dependencies to be managed at the network level
- Support asynchronous progress of multi-staged communication protocols
Motivating Example

- Collective communications optimization
  - Communication pattern involving multiple processes
  - Optimized collectives involve a communicator-wide data-dependent communication pattern, e.g., communication initiation is dependent on prior completion of other communication operations
  - Data needs to be manipulated at intermediate stages of a collective operation (reduction operations)
  - Collective operations limit application scalability
    - Performance, scalability, and system noise
Scalability of Collective Operations

**Ideal Algorithm**

- Process Rank: 1, 2, 3, 4
- Time: 0, 1, 2, 3, 6
- Work
- Reduction
- Communication
- Use Result

**Impact of System Noise**

- Process Rank: 1, 2, 3, 4
- Time: 0, 1, 2, 3, 6
- Work
- Reduction
- Communication
- Use Result
- Noise

Red X marks the points where the algorithm fails due to system noise.
Scalability of Collective Operations - II

Offloaded Algorithm

Nonblocking Algorithm

- Communication processing
Network Managed Multi-stage Communication

- **Key Ideas**
  - Create a local description of the communication pattern
  - Pass the description to the communication subsystem
  - Manage the communication operations on the network, freeing the CPU to do meaningful computation
  - Poll for full-operation completion

- **Current Assumptions**
  - Data delivery is detected by new Completion Queue Events
  - Use Completion Queue to identify the data source
  - Completion order is used to associate data with a specific operation
  - Use RDMA with the immediate to generate Completion Queue events
Key New Features

- New QP trait - Managed QP: WQE on such a QP must be enabled by WQEs from other QP’s

- Synchronization primitives:
  - Wait work queue entry: waits until specified completion queue (QP) reaches specified producer index value
  - Enable tasks: WQE on one QP can “enable” a WQE on a second QP

- Submit lists of task to multiple QP’s in single post - sufficient to describe chained operations (such as collective communication)

- Can setup a special completion queue to monitor list completion (request CQE from the relevant task)
Setting up CORE-\textit{Direct} QP’s

- Create QP with the ability to use the CORE-Direct primitives
- Decide if managed QP’s will be used, if so, need to create QP that will take the enable tasks. Most likely, this is a centralized resource handling both the enable and the wait tasks
- Decide on a Completion Queue strategy
- Setup all needed QP’s
Initiating CORE-**Direct** Communication

- **Task list is created**
  - Target QP for task
  - Operation send/wait/enable
  - For wait, the number of completions to wait for
    - Number of completions is specified relative to the beginning of the task list
    - Number of completions can be positive, zero, or negative (wait on previously posted tasks)
  - For enable, the number of send tasks on the target QP is specified
    - Number of tasks to enable
Posting of Tasks

QP-0 verbs 
Send Task 

ibv task

ibv task

ibv task

ibv task

NULL

QP-0 verbs 
Receive Task 

ibv task

ibv task

ibv task

ibv task

NULL

QP-1 verbs 
Send task

ibv task

ibv task

NULL

QP-2 verbs 
Send task

ibv task

ibv task

NULL

NULL
CORE-Direct Task-List Completion

- Can specify which task will generate completion, in “Collective” completion queue
- Single CQ signals full list (collective) completion
- CPU is not needed for progress
Example – Four Process Recursive Doubling

Step 1

Step 2
Four Process Barrier Example: Using Managed Queues – Rank 0

- Managed QP
  - QP Proc 2
    - Send to 2
      - NULL

- Managed QP
  - QP Proc 3
    - Send to 3
      - NULL

- Regular QP
  - Master QP
    - Enable send 2 (1)
      - Wait on QC 2 (1)
      - Enable send 3 (1)
      - Wait on QC 3 (1)
      - Signal Completion
      - NULL
Four Process Barrier Example: No Managed Queues – Rank 0

- QP Proc 2
  - Send to 2
    - NULL

- QP Proc 3
  - Wait on QC 2 (1)
  - Send to 3
  - Wait on QC 3 (1)
  - NULL

Signal Completion
User-Mode Memory Registration
Key Features

- Support combining contiguous registered memory regions into a single memory region. H/W treats them as a single contiguous region (and handles the non-contiguous regions).

- For a given memory region, supports non-contiguous access to memory, using a regular structure representation – base pointer, element length, stride, repeat count.
  - Can combine these from multiple different memory keys.

- Memory descriptors are created by posting WQE’s to fill in the memory key.

- Supports local and remote non-contiguous memory access.
  - Eliminates the need for some memory copies.
Combining Contiguous Memory Regions
Non-Contiguous Memory Access – Regular Access
Non-Contiguous Memory Access – Regular Access
Thank You