A NIC-offload Implementation of Portals for Quadrics QsNet

Kevin Pedretti & Ron Brightwell
Scalable Computing Systems Department
Sandia National Laboratories
Albuquerque, NM
{ktpedre, rbbrigh}@sandia.gov

Outline

- Background
- Different approaches: Portals vs. QsNet
- Design and performance of Portals on QsNet
- Conclusion and future directions
Motivation

- The Ideal ----->
  - Balanced
  - Linux
  - Light Weight Kernel
  - Purpose Built

- Would like to approximate with COTS components
  - Need a fast Portals implementation

- Gain more NIC offload experience

Portals – What is it?

- Sandia/UNM developed API for data movement
  - Portals 2.x had no functional interface (ASCI Red)
  - Portals 3.0 introduced a functional interface (Cplant)
  - Portals 3.3 being used for Red Storm

- Embedded matching (like Tports, MX)
- Designed for offloading
- Designed with light weight kernel in mind
- External users: Lustre
Portals Data Movement

Matching Put

Optional Acknowledgement

Matching Get

Port Table

Match List

Memory Descriptor

Event Queue

Memory Regions

Library Space

Application Space

tag=B

Quadrics QsNet

- Elan NIC
  - Programmable processor
  - Memory management unit
  - 64 MB SDRAM
  - RDMA engine
- Supplied with extensive software stack
  - IP stack
  - Kernel comms
  - MPICH, SHMEM
  - Tports (in Libelan)
  - Libelan, Libelan3
QsNet RDMA Operation

1. Source Host
   - Host Memory
   - Source Buffer
   - Event Block

2. User Process

3. QsNet NIC

4. Source Event
   - DMA Descriptor
   - Src Address
   - Length
   - Dest Address
   - Src Event
   - Dest Event

5. QsNet Queued RDMA Operation

6. Destination Host
   - Host Memory
   - Destination Buffer

7. User Process

8. QsNet NIC

9. Queue Event

QsNet Queued RDMA Operation

1. Source Host
   - Host Memory
   - Source Buffer
   - Event Block

2. User Process

3. QsNet NIC

4. Source Event
   - DMA Descriptor
   - Src Address
   - Length
   - Slot Offset
   - Src Event
   - Dest Queue Desc

5. Queue Event
   - Queue Slot 1
   - Queue Slot 2
   - Queue Slot 3

6. Destination Host
   - Host Memory
   - QsNet NIC

7. User Process
Multi-Protocol Support

- Portals: Portals table abstraction
  - Each protocol allocates a set of Portals table entries
    - Example: MPI entries 1-5, Lustre entries 5-8
  - Match entries and memory descriptors then attached
  - Drawback: possible overkill
- QsNet: NIC thread per protocol
  - Hardware primitives provide minimal functionality
  - NIC threads used to implement more complex semantics
  - Drawbacks: complicated hardware, limited portability

User-Level Network Protection

- Portals: Receiver-side checks
  - All network end-points can communicate by default
    - End-points addressed by (nid, pid)
    - File systems and runtime systems need this
  - Access control lists setup to filter unwanted messages
- QsNet: Capabilities
  - Only processes in a parallel application can communicate
    - End-points addressed by rank
  - Drawback: relies on runtime to distribute capabilities
Network Failures

- Portals: Exposes network failures
  - Completion events can specify operation failure
  - File systems and runtime systems need this

- QsNet: Failures not exposed
  - Runtime handles failures (kills application)

Portals vs. Other APIs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Portals</th>
<th>Tports</th>
<th>GM</th>
<th>MX</th>
<th>SHMEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded Matching</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent processes</td>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Connectionless</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliable</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Ordered</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>No alignment restrictions</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Independent MPI progress</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Allows overlap</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Asynchronous completion</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>One sided operations</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Exposes failures</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Allows reviews to be canceled</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Collective operations</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Embeds unexpected message handling</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>
Design Options

Small Message Protocol
Large Message Protocol

Ping-Pong Latency
Ping-Pong Bandwidth

![Graph showing throughput versus message size for different APIs.]

Ping-Pong Host Level Timing

<table>
<thead>
<tr>
<th>Post receive for pong</th>
<th>Send ping</th>
<th>Wait for send to complete</th>
<th>Wait for pong to arrive</th>
<th>Total:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>6.4</td>
<td>2.2</td>
<td>24.7</td>
<td>36.1</td>
</tr>
<tr>
<td>Tports</td>
<td>1.7</td>
<td>1.0</td>
<td>0.1</td>
<td>11.3</td>
</tr>
</tbody>
</table>

(microseconds)

- Post receive takes two API calls (ME Attach, MD Attach)
  - Implemented separately (2x PCI traffic)
- Portals events are large (128 bytes)
  - API copies events by value
Ping-pong NIC Thread Timing

<table>
<thead>
<tr>
<th></th>
<th>Portals</th>
<th>Tports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Take lock</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Inspect header</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>Match</td>
<td>1.4</td>
<td>0.3</td>
</tr>
<tr>
<td>Hosekeeping</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>Write completion event</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Unlink matched buffer</td>
<td>1.3</td>
<td>0.1</td>
</tr>
<tr>
<td>Drop lock</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>4.9</strong></td>
<td><strong>1.5</strong></td>
</tr>
</tbody>
</table>

(microseconds)

- Portals matching more complex
- Supporting conditional unlinking adds overhead
  - Tports buffers are always unlinked, allows optimization

Conclusion

- Portals and QsNet take different approaches
  - Portals “all-in-one”, QsNet many specialized
  - Portals useful for file system, runtime
- Portals is implementable on QsNet
  - Not a perfect match
  - Rich semantics of Portals costs performance
Next Steps

• QsNetII (Elan4) looks promising
  – PCI-X based
  – Faster thread processor (2x)
  – Possibly write completion events into queue
• 64 node QsNetII development cluster in house
• Possible Portals changes

Backup Slides
Portals Receive Processing

Main Portals API Functions

- **Match Entries (MEs)**
  - PtlMEAttach, PtlMEInsert, PtlMEUnlink

- **Memory Descriptors (MDs)**
  - PtlMDAttach, PtlMDBind, PtlMDUnlink

- **Event Queues (EQs)**
  - PtlEQAlloc, PtlEQFree

- **Data Movement**
  - PtlPut, PtlGet
Main Portals API Structures

typedef struct {
  void *
  ptl_size_t
  int
  unsigned int
  unsigned int
  void *
  ptl_handle_eq_t eq_handle;
} ptl_md_t;

typedef struct {
  ptl_event_kind_t type;
  ptl_process_id_t initiator;
  ptl_uid_t uid;
  ptl_jid_t jid;
  ptl_pt_index_t pt_index;
  ptl_match_bits_t match_bits;
  ptl_size_t rlength;
  ptl_size_t mlength;
  ptl_size_t offset;
  ptl_handle_md_t md_handle;
  ptl_md_t md;
  ptl_hdr_data_t hdr_data;
  ptl_seq_t link;
  ptl_ni_fail_t ni_fail_type;
  ptl_seq_t sequence;
} ptl_event_t;

int PtlMEAttach(ni, portal, match_id,
  match_bits,ignore_bits,
  unlink, position, &me_handle);

int PtlMDAttach(me_handle, md, unlink_op, &md_handle);

QsNet RDMA Descriptor

Typedef volatile struct e3_dma {
  E3_DmaType         dma_u;
  E3_uint32          dma_size;
  E3.Addr            dma_source;
  E3.Addr            dma_dest;
  E3.Addr            dma_destEvent;
  E3_CookieVproc     dma_destCookieProc;
  E3.Addr            dma_src_Event;
  E3_CookieVproc     dma_srcCookieProc;
} E3_DMA;

elan3_initevent_blk(sdramaddr_t);
elan3_waitevent(sdramaddr_t);
elan3_putdma(E3_DMA);
elan3_getdma(E3_DMA);
Matching Semantics

- Portals: Embedded Complex Matching Semantics
  - Match entry list traversal general enough for all network services
  - Complex memory descriptors
    - Can be persistent (e.g., threshold = infinity)
    - Offset management (local vs. remote offset management)
  - Drawback: possible overkill

- QsNet: NIC Threads Perform Matching
  - Tports has simple matching semantics
  - Drawbacks: hardware complexity, portability

Portals Header

<table>
<thead>
<tr>
<th>flags</th>
<th>portal</th>
<th>sender</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>length</td>
<td></td>
</tr>
<tr>
<td></td>
<td>match bits</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hdr data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ack event address</td>
<td></td>
</tr>
</tbody>
</table>

very small message data or long protocol initiator address

4 bytes

32 bytes

64 bytes
MPI over Portals

**Standard Short Send**

- `PtIMDBind()`
- `PtIPut()`
- `PTL_EVENT_SEND`

- `PtIMEAttach()`
- `PtIMDAattach()`
- `PTL_EVENT_PUT`

**Synchronous Short Send**

- `PtIMDBind()`
- `PtIPut()`
- `PTL_EVENT_SEND`

- `PtIMEAttach()`
- `PtIMDAattach()`
- `PTL_EVENT_PUT`

- `PTL_EVENT_ACK`

**MPI Over Portals (cont.)**

**Long Send Pre-Posted**

- `PtIMEAttach()`
- `PtIMDAattach()`
- `PtIPut()`
- `PTL_EVENT_SEND`

- `PTL_EVENT_ACK`

**Long Send Unexpected**

- `PtIMEAttach()`
- `PtIMDAattach()`
- `PtIPut()`
- `PTL_EVENT_SEND`

- `PTL_EVENT_PUT`

- `PtIGet()`

- `PTL_EVENT_REPLY`