Implementation and Performance of Portals 3.3 on the Cray XT3

Ron Brightwell    Trammell Hudson    Kevin Pedretti
Rolf Riesen       Keith Underwood
Sandia National Laboratories
Scalable Computing Systems Department

IEEE International Conference on Cluster Computing
September 28, 2005
Outline

• Portals history
• Portals objects
• Portals implementation
• Portals for Cray SeaStar
• Performance
• Conclusions
• Future Work
Portals Timeline

• Portals 0.0 - 1991
  – SUNMOS (Sandia/UNM OS)
  – nCUBE, Intel Paragon
  – Direct access to network FIFOs
  – Message co-processor
• Portals 1.0 - 1993
  – Data structures in user-space
  – Kernel-managed and user-managed memory descriptors
  – Published but never implemented
• Portals 2.0 - 1994
  – Puma/Cougar
  – Message selection (match lists)
  – Four types of memory descriptors (three implemented)
• Portals 3.0 - 1998
  – Cplant/Linux
  – Functional API
  – Target intelligent/programmable network interfaces
Portals 3.3 Features

- Best effort, in-order delivery
- Well-defined transport failure semantics
- Based on expected messages
- One-sided operations
  - Put, Get, Atomic swap
- Zero-copy
- OS-bypass
- Application offload
  - No polling or threads to move data
  - No host CPU overhead
- Runtime system independent
Put Operation

Initiator

Target

Data Transmission

Portal Translation

Optional Acknowledgement
Get Operation

[Diagram showing the flow of a Get Operation with arrows labeled: Request, Portal, Translation, and Data Transmission between Initiator and Target]
Portal Translation

Portal Table

Operational Boundary

Match List
Memory Descriptors
Event Queue
Memory Regions

Portal Space
Access Control Table
Application Space
Match Entry Contents

- Source node id
- Source process id
- 64 match bits
- 64 ignore bits
Memory Descriptor

- Start address
  - Optionally supports gather/scatter list
- Length in bytes
- Threshold
  - Number of operations allowed
- Max size
  - Low-water mark
- Options
  - Put/get
  - Receiver/sender managed offset
  - Truncate
  - Ack/no ack
  - Ignore start/end events
- 64 bits of user data
- Event queue handle
- Auto-unlink option
Event Queue

- Circular queue that records operations on MDs
- Types of events
  - Get (PTL_EVENT_GET_{START,END})
    - MD has received a get request
  - Put (PTL_EVENT_PUT_{START,END})
    - MD has received a put request
  - Reply (PTL_EVENT_REPLY_{START,END})
    - MD has received a reply to a get request
  - Send (PTL_EVENT_SEND_{START,END})
    - Put request has been processed
  - Ack (PTL_EVENT_ACK)
    - MD has received an ack to a put request
Event Scenarios

- initiator
  - send start
  - send end
  - ack
  - reply start
  - reply end

- target
  - put
    - put start
    - put end
  - get
    - get start
    - get end
Event Entry Contents

• Event type
• Initiator of event (nid,pid)
• Portal table index
• Match bits
• Requested length
• Manipulated length
• Offset
• 64 bits of out-of-band data
What Makes Portals Different?

- Connectionless RDMA with matching
- Provides elementary building blocks for supporting higher-level protocols well
  - MPI, RPC, Lustre, etc.
- Allows structures to be placed in user-space, kernel-space, or NIC-space
- Receiver-managed offset allows for efficient and scalable buffering of “unexpected” messages
- Supports multiple protocols within a process
  - Needed for compute nodes where everything is a message
Portals Reference Implementation Design

API Space

API

Network

Library Space

Library

Abstraction

Layer

Transport
Myrinet Kernel Implementation
Cray SeaStar NIC/Router

- 16 1.6 Gb/s HyperTransport to Opteron
- 500 MHz embedded PowerPC 440
- 384 KB on-board scratch RAM
- Seven-port router
- Six 12-channel 3.2 Gb/s high-speed serial links
Cray Portals Bridge

• Needed single version of NIC firmware that supports all combinations of
  – User-level and kernel-level API
  – NIC-space and kernel-space library
• Cray added bridge layer to reference implementation to allow NAL to interface multiple API NALs and multiple library NALs
  – qkbridge for Catamount applications
  – ukbridge for Linux user-level applications
  – kbridge for Linux kernel-level applications
SeaStar NAL

- Portals library currently in kernel-space
  - Interrupt-driven
  - “generic”
- Portals library moving to NIC-space
  - No interrupts
  - “accelerated”
Micro-Benchmarks

- ptlperf
  - Ping-pong latency and bandwidth (uni- and bi-directional)
  - Single, persistent ME, MD, EQ
  - Best-case performance for Portals
- mpilatency
  - Standard best-case ping-pong latency and bandwidth
- NetPIPE 3.6.2
  - Ping-pong latency and bandwidth (uni- and bi-directional)
  - Streaming bandwidth
  - Implemented a Portals module
MPI Implementations

- **MPICH 1.2.6 with Portals 3.3 device**
  - Originally developed for Cplant Linux clusters
  - Uses a copy block for very short messages
    - Avoids waiting for wire-level acknowledgment
    - Avoids overhead of creating a memory descriptor
- **MPICH2 0.97 with Portals 3.3 device**
  - Cray supported version for XT3
Disclaimer

• Results are from a developer snapshot of a Sandia code base from last week
• This software may or may not make it to other Cray XT3 systems
• Accelerated implementation has undergone minimal tuning
A graph showing latency (microseconds) versus message size (bytes) for different MPI implementations: mpich2-gen, mpich-1.2.6-gen, portals-gen, mpich2-accel, mpich-1.2.6 accel, and portals-accel.
NetPIPE Latency

![Graph showing latency vs. message size for different MPI implementations. The x-axis represents message size in bytes, ranging from 1 to 1000, and the y-axis represents latency in microseconds, ranging from 0 to 14. Different lines represent different MPI implementations: mpich2-gen, mpich2-accel, mpich-1.2.6-gen, portals-gen, mpich-1.2.6 accel, portals-accel. Each line shows varying latency across different message sizes.](image-url)
NetPIPE Bandwidth (Default)
NetPIPE Bandwidth (Preposted)
NetPIPE Bandwidth Comparison

![Graph showing bandwidth comparison](image-url)
Conclusions

• Portals 3.3 is the lowest-level network programming interface on the Cray XT3
• Cray bridge abstraction allows single instance of firmware to support multiple API and Library paths
• Accelerated NIC-space implementation achieves ~3.8 µs latency
• Generic kernel-space implementation achieves ~4.7 µs latency
• Asymptotic bandwidth is ~1.1 GB/s
Future Work

- Tune accelerated implementation
- Work with Cray to get short message optimization into MPICH2
- More optimizations for MPI
  - Use persistent memory descriptors for send side
    - Avoid creating an MD for all sends
  - Rendezvous protocol for benchmarking
- More and better benchmarks
  - CPU utilization/overhead
  - Collective operations
- NIC-based collective operations
- Next-generation Portals API
Questions?