Networking R&D at Sandia: To Red Storm and Beyond

Ron Brightwell
Scalable Computing Systems Department
Center for Computation, Computers, Information and Math
Sandia National Laboratories

Pacific Northwest National Lab
January 29, 2007
Outline

• Sandia massively parallel systems
• Evolution of Portals
• Networking requirements and Red Storm
• Petascale networking requirements
• Ongoing research and development activities
• Application network resource usage patterns
Sandia Systems

1990

nCUBE2
- Sandia’s first large MPP
- Achieved Gflops performance on applications

1993

Paragon
- Tens of users
- First periods processing MPP
- World record performance
- Routine 3D simulations
- SUNMOS lightweight kernel

1997

ASCI Red
- Production MPP
- Hundreds of users
- Red & Black partitions
- Improved interconnect
- High-fidelity coupled 3-D physics
- Puma/Cougar lightweight kernel

1999

Cplant
- Commodity-based supercomputer
- Hundreds of users
- Enhanced simulation capacity
- Linux-based OS licensed for commercialization

Red Storm
- 41 Tflops
- Custom interconnect
- Purpose built RAS
- Highly balanced and scalable
- Catamount lightweight kernel

2004
2007 Red Storm - Upgraded

- 12,980 2.4 GHz dual-core AMD Opteron CPUs
  - 124.6 TF/s peak
- SeaStar 2.1 network
  - 2.1 GB/s one-way bandwidth
- 2 GB DDR 333 Memory
- #2 on November 2006 Top 500 list
- Catamount LWK with virtual node mode support
Evolution of Portals
Portals 0

- **SUNMOS (Sandia/UNM OS)**
  - Modeled on Vertex (the OS for the nCUBE)
  - Dynamic allocation for incoming messages
- **Experiments**
  - Multiple paths
  - Pre-posted receives
  - Use of a co-processor
- **nCUBE 2 and Intel Paragon**
  - Direct access to network FIFO’s
  - Message co-processor (Paragon)
Portals 1.0

- Moved all message reception structures to user space
- Kinds of portals
  - Kernel managed portals
  - Single block portals
- Never implemented
- Published 😊
Portals 2.0

- Puma/Cougar lightweight kernel
- Separate matching from memory descriptors
- Variety of memory descriptors
  - Kernel managed (dynamic)
  - Single block
  - Independent block
  - Combined block
- Intel TeraFLOPS (ASCI Red)
  - Direct access to message FIFO’s
  - Message co-processor
Issues with Portals 2.0

- No API
  - Data structures in user space
  - Protection boundaries have to be crossed to access data structures
  - Data structures have to be copied, manipulated, and copied back
  - Requires interrupts
- Address validation/translation on the fly
  - Incoming messages trigger address validation
  - Doesn’t fit the Linux model of validating addresses on a system call for the currently running process
Portals 3.x

- Operational API
- Unified memory descriptors
- Commodity processors and networks
  - Alphas, IA-32, IA-64, etc.
  - Linux OS with modules
  - Myrinet, Quadrics, etc.
  - DMA access to memory
- Fundamental change
  - NIC doesn’t have logical address maps
  - NIC access to memory needs to be carefully managed
Portals 3.3 Features

• Best effort, in-order delivery
• One-sided operations
  – Put, Get, Atomic swap
• Supports zero-copy
• Supports OS-bypass
• Supports application offload
  – No polling or threads to move data
  – No host CPU overhead
• Well-defined transport failure semantics
• Unexpected operations are discarded
• Receive-side access control
• Runtime-system independent
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 MPI “unexpected” messages
- Supports multiple protocols within a process
  - Needed for compute nodes where everything is a message
Portals Characteristics

- Minimal library space
  - Nothing depends on message size
  - All objects can be confirmed when created
- Designed for library writers
  - Not for application developers
  - Low-level API
    - We're happy to drop requests
    - Structures are complicated
    - Some functions (`PtlMDUpdate()`) are not obvious
- Designed to reflect underlying hardware
  - NICs
  - Packets and failure
- Provide the right amount of protection
What Portals Does

• Separates communication space from computation space
  – Moderately dynamic
    • During descriptor construction
    • Any part of an application's memory can be used for communication
  – Simplifies coherence issues
    • Important for PCI implementations as well
• Handles important protocol processing
  – MPI long message strategy
    • Force rendezvous at receiver
    • Post and forget
  – Supports parallel servers
What Portals Doesn't Do

- Dynamic integration of computation and communication space
  - May be needed for things like UPC
  - Race conditions
  - Memory consistency models

- Poor support for collectives
  - Each process must actively participate in collective operation
  - Would prefer to have a “contribute and forget” capability
  - Reduce variance in time for collective operations
Networking Requirements and Red Storm
Scalable MPP Interconnect Requirements

• High Bandwidth
  – Balance is the key to system scalability
  – Red Storm required 1.5 GB/s per direction
• Reliability
  – Uncorrected bit errors have to be extremely rare
  – High bandwidth combined with a large system requires at least $10^{-21}$ bit error rate
• High Message Rate
  – at least 1 million MPI messages per second
• Connectionless
  – A connection oriented model does not scale to 25,000 cores
Meeting the Requirements

• HyperTransport for bandwidth and integration
  – Highest bandwidth interface available
  – Lowest latency interface available
  – Eliminates need for additional chipset
• Hardware support for reliability
  – 16 bit link level CRC with link retries
  – End-to-end CRC in DMA engines
• 500 MHz PPC 440 for message rate
  – Still the fastest processor on a NIC today
  – Responsible for all aspects of NIC management
• Portals for API
  – Strong match to MPI semantics
  – Enables independent progress and offload
RDMA is the Wrong Model for MPI at Scale

• Complexity of scalable connection management
• Lack of message matching ability leads to
  – No progress without extra threads
  – Receive CPU overhead on every transfer
  – Severely limited ability to overlap computation with communication
• Lack of scalable support for unexpected messages leads to
  – Extra flow control in MPI library to manage unexpected message buffers
  – Inefficient use of application memory
Portals Was Designed for MPI at Scale

- Connectionless
- Supports MPI matching semantics
  - Allows for offloading MPI matching to NIC
  - Very low CPU overhead for both small and large messages
- Maximizes overlap of computation and communication
- Provides scalable and efficient support for buffering unexpected messages
  - Does not require extra flow control in MPI
A NIC to Support Portals

![Diagram of a NIC system with components like Host Memory, Opteron CPU, Memory Controller, SeaStar, TX DMA Engine, RX DMA Engine, HyperTransport, Local SRAM, PowerPC 440, Serial, Processor Local Bus, and RAS Controller.]
SeaStar Network Interface

- Independent send/recv DMA engines between Opteron memory and network
  - No PIO capability
- Message-based
  - DMA engines handle packetization
  - Message is native interaction between nodes
- Attempts to minimize host overhead
  - PPC drives DMA engines
  - PPC can handle MPI matching
- Supports reception of multiple simultaneous messages
SeaStar Embedded PowerPC

- Message transmission
  - PPC keeps the transmit DMA engine filled
- Message demultiplexing
  - MPI matching
  - Native IP packets
- End-to-end reliability protocol (in Sandia developed code)
  - Transmit side: retransmit when we learn a message has been dropped
  - Receive side: send ACK or NACK
Delivering Reliability

• Error detection and correction on memory
  – SECDED on scratch RAM with scrubbing
  – SEC on routing lookup tables
  – Parity protection on DMA tables
• CRC protection on the data
  – Link protocol – 16-bit CRC with automatic retry
  – DMA engines compute NIC-to-NIC 32-bit message CRC
  – Independent CRC on header and data
Integrated Router to Deliver Bandwidth

• Link to host uses HyperTransport
  – 2 GB/s per direction
  – 3.6 GB/s aggregate bidirectional because DMA requests consume some bandwidth

• Six high-speed network links per ASIC
  – Enable 3D torus to improve bisection bandwidth
    • Requires additional virtual channels and buffering
    • Requires support for longer cables
  – Twelve 3.2 Gb/s SERDES in each dimension
    • 3.84 GB/s of raw bandwidth
    • Over 2.5 GB/s of sustainable bandwidth

• 500 MHz router operation to sustain full speed crossbar
## Red Storm is a Highly Balanced System

<table>
<thead>
<tr>
<th>Machine</th>
<th>Peak Node (GFLOPS)</th>
<th>Peak BW (GB/s)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM BG/L</td>
<td>5.6</td>
<td>0.35</td>
<td>0.0625</td>
</tr>
<tr>
<td>Cray Red Storm’07</td>
<td>9.6</td>
<td>4.8</td>
<td>0.5000</td>
</tr>
<tr>
<td>IBM Purple</td>
<td>48</td>
<td>8</td>
<td>0.1700</td>
</tr>
<tr>
<td>SGI Columbia</td>
<td>24</td>
<td>6.4</td>
<td>0.2700</td>
</tr>
<tr>
<td>Dell Thunderbird</td>
<td>14.4</td>
<td>2</td>
<td>0.1300</td>
</tr>
<tr>
<td>Cray Red Storm’04</td>
<td>4</td>
<td>4.8</td>
<td>1.2000</td>
</tr>
<tr>
<td>NEC Earth Simulator</td>
<td>64</td>
<td>12.3</td>
<td>0.1920</td>
</tr>
<tr>
<td>Mare Nostrum</td>
<td>17.6</td>
<td>0.5</td>
<td>0.0280</td>
</tr>
<tr>
<td>Thunder</td>
<td>22.4</td>
<td>1</td>
<td>0.0400</td>
</tr>
</tbody>
</table>
Portals 3.3 for SeaStar

- Based on Sandia reference implementation
  - Network Abstraction Layer (NAL) for portability
- Needed single version of SeaStar firmware to support
  - User-level and kernel-level API
  - NIC-space and kernel-space library
- Cray developed bridge layer to allow multiple simultaneous NALs
  - qkbridge for Catamount applications
  - ukbridge for Linux user-level applications
  - kbridge for Linux kernel-level applications
Multiple SeaStar NALs

- Portals processing in kernel-space
  - Interrupt-driven
    - Kernel inspects every message header
    - Programs DMA engines to deliver data
  - Cray supported mode
  - “Generic” mode
- Portals processing on SeaStar
  - No interrupts
    - SeaStar handles all message processing
  - “Accelerated” mode
  - Takes advantage of Catamount OS memory structure
  - Sandia-developed mode
  - Available in future Cray software release
PMB PingPong Latency
PMB PingPong Bandwidth

![Graph showing Bandwidth vs. Message Size](image-url)
PMB Sendrecv Bandwidth

![Graph showing Bandwidth vs. Message Size for different models: SS1 Generic, SS1 Accelerated, SS2 Generic, SS2 Accelerated. The x-axis represents message size in bytes, ranging from 1 to 10^7, and the y-axis represents bandwidth in MB/s, ranging from 0 to 3500. Each model's performance is represented by a line with different markers and colors.](image)
OSU Streaming Bandwidth
SS1 - CPU Availability – Send
SS1 - CPU Availability – Receive

Default rendezvous mode for long MPI messages significantly decreases availability.

MPICH_PTLS_EAGER_LONG
SS2 - HPCC Baseline RandomAccess

Accelerated mode increases small message throughput for dual-core nodes.
Short-Term Portals Activities

• “Accelerated” Portals evaluation
• Portals collective library
  – Collective operations built on top of Portals
• Non-blocking collective functions
  – Collective operations integrated into Portals
  – SeaStar can support offloading collective operations
  – Barrier proof-of-concept is done and working
• Portals 4.0
  – Laundry list of issues with Portals 3.3 is too big
  – Unnecessary symmetry (PTL_EVENT_SEND_START)
  – Unneeded operations (arbitrary list insertion)
  – Missing functionality
Next-Generation Network Requirements

• Performance
  – 20 GB/s per direction
  – 500 ns one-way nearest neighbor MPI latency
  – 20 ns per hop of a 3D mesh
  – 15 million MPI messages per second

• Reliability
  – $10^{-23}$ bit error rate
  – Link-level flow control is a must

• Functional
  – MPI
  – Non-coherent global load/store (PGAS)
  – Network-level collective communication
Sandia Networking Research Projects

• Portals and MPI in hardware
  – Associative list processing units (Underwood)
• Network simulation
  – Structured Simulation Toolkit (Rodrigues)
  – MPI network simulator (Riesen)
• Application network resource usage (Brightwell)
Application Network Resource Usage
Research Motivation

- Intelligent/programmable network interfaces have been shown to be beneficial for
  - Protocol offload (TOE, RTS/CTS, etc.)
  - Application offload (MPI matching, collectives, etc.)
- Typical network interface resources
  - Slower CPU relative to host CPU(s)
  - Smaller memory relative to host
- Research questions:
  - What level of resources is needed?
  - What is the impact of limited resources?
  - Are current resource management strategies appropriate?
Practical Motivation

• Red Storm Seastar NIC
  – 500 MHz embedded PowerPC
  – 384 KB on-board scratch RAM
  – Other (possibly) scary things
  – Portals 3.3 programming interface

• Practical questions:
  – Will important Sandia applications work at scale?
  – What demands do Sandia applications place on network resources?
  – Will applications need to adapt to Red Storm? If so, how?
Goals

• Better understanding of how real applications use network resources
• Explore whether this type of analysis can help
  – Characterize performance/scalability of applications
  – Identify potential application performance/scalability problems
  – Determine the amount of required network resources
  – Evaluate different resource management strategies
MPI Queue Abstractions

• Posted receive queue
  – List of pending receives that the user has enqueued using `MPI_Irecv()` or `MPI_Recv()`
  – An incoming message traverses the posted receive queue looking for a matching posted receive

• Unexpected message queue
  – Also called the early arrival queue
  – List of received messages (or partial messages) for which there is no matching posted receive
  – Posting a receive involves atomically searching the unexpected message queue
Network Resources Needed by MPI

• Processor
  – Traversing posted receive queue
    • Every time a message arrives
  – Traversing unexpected message queue (possibly)
    • Every time a receive is posted

• Memory
  – Posted receive queue entries
  – Unexpected message queue entries (possibly)
Approach

• Instrumented MPICH/GM to track
  – Unexpected messages (short/long)
  – Expected messages (short/long)
  – Posted and unexpected queue data
    • Number of times searched
    • Number of entries searched
    • Maximum number of queue entries
    • Maximum number of queue entries search
• Implemented counters as global variables
• Used MPI profiling interface to write data
• Data averaged over four runs
• Two processes per node
Long Expected Messages
Long Unexpected Messages
Short Expected Messages

![Graph showing the expected messages breakdown vs. number of processors.](image)
Short Unexpected Messages
Posted Queue - Max Length

All Ranks

No Rank 0
Posted Queue - Max Search

All Ranks

No Rank 0
Unexpected Queue - Max Length

All Ranks

No Rank 0
Unexpected Queue - Max Search

All Ranks

No Rank 0
Average Posted Queue Search

![Graph showing average posted queue search entries against number of processors.]

- CTH - AMR: Green line with square markers.
- CTH - 2Gas: Red line with diamond markers.
- ITS: Blue line with square markers.
- LAMMPS: Pink line with diamond markers.
Average Unexpected Queue Search

![Graph showing the relationship between the number of processors and the number of entries for various programs. The graph indicates that as the number of processors increases, the number of entries also increases significantly.](image)
Point-to-point Versus Collective

• Offloading of collective operations has strong support in the community
• Need some idea whether resources significantly different for collective operations
• Implemented separate queues for point-to-point and collective operations
Long Expected

Point-to-Point

Collective
Long Unexpected

Point-to-Point

Collective
Short Expected

Point-to-Point

Collective
Short Unexpected

**Point-to-Point**

**Collective**
Max Posted Queue Length

Point-to-Point

Collective
Max Posted Queue Searched

Point-to-Point

Collective
Max Unexpected Queue Length

Point-to-Point

Collective
Max Unexpected Queue Searched

Point-to-Point

Collective
Average Unexpected Queue Searched

Point-to-Point

Collective
Summary

- Usage of network resources varies dramatically across applications and NAS parallel benchmark suite.
- Significant variability in parameters for a single application:
  - Rank 0 seems to always be a hog.
- Linear growth of queues and queue traversals for point-to-point and collectives is potentially bad for scaling.
- May require greater amount of NIC resources or restructuring of applications.
Ongoing Work

• Gather data on Red Storm for accelerated and generic Portals
• Design better benchmarks that test network parameters under more realistic workloads
• Understand the performance and scalability implications of resource management policies and implementations
• Better tools for data gathering and analysis
Acknowledgments

- Cray
  - Bob Alverson
  - Portals development team
- Sandia
  - Sue Kelly, Sue Goudy, Jim Laros, Kevin Pedretti, Rolf Riesen, Keith Underwood
  - On-site Cray support team
- University of New Mexico
  - Barney Maccabe
- OS Research
  - Trammell Hudson