Low latency, high bandwidth communication. Infiniband and RDMA programming

Knut Omang
Ifi/Oracle
2 Nov, 2015

Bandwidth vs latency

There is an old network saying:
“Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed – you can't bribe God.”
Motivation

- Traditional network stacks: Explicit message passing + overhead
  - Copying, checksumming, buffering, interrupts,...
  - How should interfaces look to promote low latency?
  - How can hardware help?
- In an SMP (Symmetric MultiProcessor): Communication in shared memory.
  - But now often: CC-NUMA (Cache coherent non-Uniform Memory architecture)
  - Deep cache hierarchies..
- “Can we use the SMP model even across the network?”
- “If not, how do we make our APIs to facilitate low latency?”
- Problem originates in HPC, but applies to all parallel apps!

Overview

- Bandwidth and latency..
- Some background and history
  - earlier technologies, perspectives
  - past challenges (successes and griefs..)
- Infiniband
  - infrastructure
  - the HCA (Host Channel Adapter)
  - Linux software model (OFED)
Low latency communication beyond the SMP

A lot of shared memory thinking
- Desire to make it easy to program
- Well known model from SMPs, reuse programs?

Approaches:
- Creative software solutions (Lazy release consistency)
- CC-NUMA (Cache Coherent non-uniform memory architecture)
- COMA (Cache Only Memory Architecture)

Problems:
- Only modest gains on legacy parallel apps..

NUMA (Non-Uniform Memory Architecture)
Low latency communication beyond CC-NUMA

- SCI (Scalable Coherent Interface)
- Myrinet - proprietary interconnect from Myricom
- Several other proprietary

- Special hardware, custom software
- Reducing latency, increasing bandwidth
- “Zero copy”
- Use CPU instructions to write directly to remote memory
  - skip system calls (overhead of kernel traps)

SCI (Scalable Coherent Interface (IEEE standard 1596-1992))

- Ring based, hardware supported network (data center range)
- Support for shared memory between computers
- Optional Cache Coherency protocol
- Dolphin Interconnect Solutions (off spin from Norsk Data)
  - I/O interconnect boards (SBus → PCI)
  - Used without cache coherency
  - Designed for shared memory, but very NUMA!
  - Shared address space!
Scali

- “Affordable supercomputing”
- Norwegian startup
- Off-spin from Kongsberg Group (defence computing)
  - Radar images processing...
- Specialized hardware → “off-the-shelf” + PCs
- Modified versions of Dolphin adapters
  - Own software, including driver (written by some phD. Students... ;-) )

SCI adapters from Dolphin ICS/Scali

- PCI 133MB/s
- B-Link 400MB/s
- LC-2
- LC-2
- 4x 500MB/s SCI links
SCI adapters from Dolphin ICS

- Error detection (checksumming etc) in hardware
- Ability to write into remote memory
- Ability to read from remote memory
- Non-strict ordering
  - explicit barrier operation - expensive!
- Shared memory locations very asymmetric!
  - close to one side, far from other side!
  - In effect, only usable for message passing model!
  - How to implement efficiently..

SCI: 2D-Torus (64 nodes, ca. 1999)

Bi-section
bandwidth: 14Gbyte/s
Longest
Latency: 1.85 \( \mu \)sec
SCI: 3D-Torus, 4-ary 3-cube (64 nodes)

- Bi-section bandwidth: 24Gbyte/s
- Longest Latency: 2.3 μsec

University of Paderborn: Flagship installations

PSC2
- 12 x 8 Torus
- 192 Processors
- 450MHz
- 86.4GFlops

PSC1
- 8 x 4 Torus
- 64 Processors
- 300MHz
- 19.2GFlops
Dolphin/Scali and SCI - some lessons to learn

- Small missing hardware features/weaknesses can result in huge software overhead/complexity!
  - Ordering
  - Observability
  - Error handling
  - Quality (heating, cables, cable design)
- Uncertainty is a big thing for startups..
- Dangerous to be small and compete on price!
- Custom, bleeding edge means expensive..
- Timing...

Message passing in shared memory

- The traditional SMP approach
  - Synchronized data structures:
    - critical sections: Serialize!
    - Shared locations protected by locks
- Problem: Asymmetry:
  - Process local to the memory had much faster access
  - Very slow access for everyone else - perf. killer...
  - Fairness..
- Not only between machines, also an increasing SMP problem!
  - remember false sharing?
Remember false sharing?

Solution: Avoid the sharing part

- Memory write is (can be made) asynchronous
- Memory read instructions (normally) synchronous
- Avoid reading from remote
- Avoid sharing data - separate “ownership”
VIA: Virtual Interface Architecture

- Attempt to standardize communication software for low latency interconnects
- Tailored for “Zero copy”
- Formalized message queue abstractions
- Kernel bypass
- Evolved into Infiniband, iWARP, RoCE
  - RDMA (Remote DMA)
Infiniband (IB)

- Standard v.1.0 ready in 2000
- Intended to replace PCI, Ethernet, HPC interconnects and Fibre Channel...
- Designed to allow direct user level communication (without kernel calls)
- Today: Commonly used as interconnect
  - within data centers
  - for supercomputers
- IPoIB (IP over Infiniband)
- EoIB (Ethernet over Infiniband)
- Fibre channel over RDMA
- NFS over RDMA
- (Infiniband) RDMA over Converged Ethernet (RoCE)
- ...

Infiniband architecture
Infiniband: Defined by software and hardware

- **HCA: Host Channel Adapter (= IB network adapter)**
  - Defines set of operations to support
  - does not define if hardware or software implementation
  - Standard only defines semantics, requirements, not syntax/implementation
  - Operation at higher abstraction level than traditional network interface
  - But: Very detailed APIs – very long hello world program...

- **4 transport types**
  - UD, RC, UC (RD) U/R =Un/Reliable, D/C = Datagram, Connected mode

- **Linux: OFED (Ofen Fabrics Enterprise Distribution)**
  - Implements "RDMA support" in the kernel and from user mode
  - Common infrastructure shared by all RDMA supporting devices
  - Infiniband and other implementations

- **Standard counts ~2500 pages...**

---

Infiniband concepts

- **Queue pair (QP)**
  - Work Queues (WQ)
    - A send queue (SQ) and a receive queue (RQ)
  - State (RESET – INIT - RTR – RTS – ERROR (++))

- **Completion queue (CQ)**

- **Event queues (EQ)**
  - Out-of-band signalling – exception handling

- **Memory region (MR)**
  - Memory prepared for Infiniband access
  - Local and remote keys for access

- **Protection domain (PD)**
  - Ensures that only authorized entities may access an Infiniband resource

- **To communicate: Post Work Request into a work queue**
  - Wait for completions to arrive on associated completion queue
Infiniband communication example

- Create a protection domain
- Create completion queue(s)
- Create a QP
  - initialize it and set it in INIT state
- Register user memory buffers
- Post receives
  - modify QP state to RTR (Ready To Receive)
- Set up QP with a remote counterpart (connected mode)
  - modify to RTS (Ready to send)
- Post SEND work request (use WR ID to know what to do with a completion)
  - Wait for completion(s)
  - Work request ID used to dispatch
Infiniband communication example

Types of communication operations

- **POST send/receive**
  - Remote side is active – typical message passing

- **RDMA Write/READ**
  - One sided – remote process may not be involved

- **Atomic operations (HCA atomic or host atomic (PCIe 3.0))**
  - Allows atomic read-modify-write cycles on remote side

- **Multicast**

- **Poll or interrupt (event) based completions**
Addressing remote nodes

- **GUID (Globally Unique IDentifier)**
  - similar to ethernet addresses – vendor assigns nto each device
- **LID (Local Identifier)** – dynamic local address within subnet
  - Assigned by subnet manager
- **Ports**
  - Each HCA may have a number of ports
  - Ports may be connected to different nets and have equal or different LIDs
- **VL (Virtual Lane)s**
  - Each port supports communication in multiple lanes
  - Lanes have separate flow control
- **Paths (Primary and Alternate)**
  - For failover
- **QP index at remote node**

Summary

- Raw network latency impacted by software (and hardware) overhead
- Poll vs interrupt - low latency vs high CPU usage
- Traditional network stacks:
  - Buffering and copying, traps to kernel mode
- Low latency/high bandwidth networks: Proprietary, then SCI
  - system level interconnect used for clustering
- VIA → Infiniband: standardization of APIs tailored for low latency/high bandwidth
  - Queue pairs: Data structures for direct comm. from user to user
  - Reduce need for kernel intervention: Direct access from user space
- APIs standardized in Linux as RDMA programming.