A Case for Making Network Interfaces Less Peripheral

Shubhendu S. Mukherjee and Mark D. Hill

Computer Sciences Department, University of Wisconsin-Madison
1210 West Dayton Street, Madison, WI 53706-1685
Email:{shubu,markhill}@cs.wisc.edu
URL:http://www.cs.wisc.edu/~{shubu,markhill}

Talk Abstract

Network-centric computing, such as world-wide web and database query processing, is making network performance more important. Unfortunately, the network performance delivered to applications is less good due to the current peripheral nature of host network interfaces (NIs). A host NI is a device that a processor uses to send and receive messages from the network. Many current computer systems treat processor accesses to an NI as peripheral I/O operations. For example, a processor's accesses to an NI is often routed through the operating system like processor-initiated, peripheral disk I/O operations. In contrast, most processors provide user applications with direct and protected access to main memory through high-speed virtual memory hardware.

In this talk we will argue that NIs should be treated as "standard high-performance equipment" like main memory, and not as a peripheral I/O device, such as a disk. Such treatment opens up at least eight opportunities to improve a processor's interaction with an NI. These opportunities are:

- using virtual memory hardware, and not operating system intervention, to virtualize the NI,
- placing the NI on the higher performance memory bus, and not on the slower I/O bus,
- caching messages in processor and NI caches, like regular cachable memory,
- allowing out-of-order accesses and speculative loads on a processor's accesses to an NI, like side-effect-free regular memory accesses,
- designing the application programming interface (or API) to the NI as memory-based queues, and not directly exposing the underlying data movement primitives as the API,
- using virtual memory as a huge buffer for network messages, instead of small amounts of dedicated memory on the NI,
- transferring messages between processor caches, NI cache, and main memory through cache block transfers, instead of DMA, and
- notifying processor of NI events through cache invalidations, instead of heavy-weight interrupts.

Most of these opportunities have been partially explored by others. A principal contribution of this talk is to organize the opportunities into a framework that exposes commonality and synergistic interactions.

A Case for Making Network Interfaces Less Peripheral

Shubhendu S. Mukherjee and Mark D. Hill

Computer Sciences Department
University of Wisconsin-Madison
Email:{shubu,markhill}@cs.wisc.edu
URL:http://www.cs.wisc.edu/~{shubu,markhill}

Background & Motivation

- What is Network Interface (NI) Access?
- Why care about NI Access?

Treat NI access as Memory Access

- Eight opportunities to improve performance

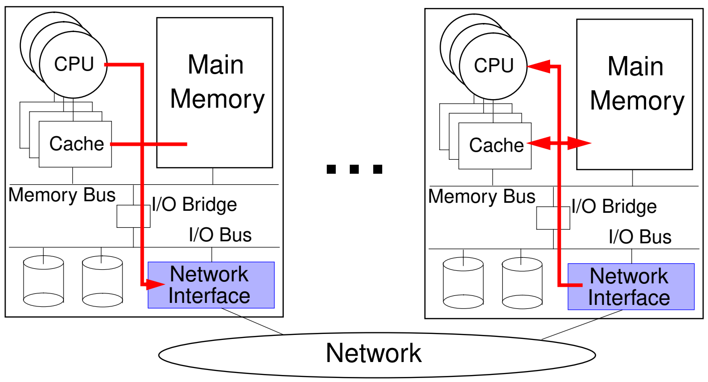
What is NI Access?

Depends on level of abstraction

- send & receive messages to & from network
- access network via NI
- read & write NI registers (status, control, & data)
  - both program-controlled I/O and DMA

NI Access = Read and Write NI Memory (primarily)

What is NI Access?



Path through software protocol

Processor access to NI memory

Path through network

Why Care About NI Access?

NI = standard high-performance equipment (like frame buffer, memory)

- almost every computer connected to network => needs NI
- high-performance network-centric computers
  - e.g., servers, parallel computer nodes

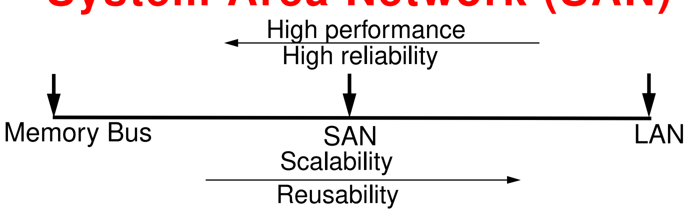
Fine-Grain Communication (messages less than few kilobytes)

- Network File Server [Kay and Pasquale, SIGCOMM, 1993]
- WWW Servers [Arlitt & Williamson, Sigmetrics, 1996]
- Database Query [Keeton, et al., Hot Interconnects III, 1995]
- Parallel scientific applications [Cypher, et al., ISCA, 1993]

High-Performance System Area Networks

- e.g., Myricom Myrinet, IBM Vulcan, SGI Spider, Fujitsu AP-Net

System Area Network (SAN)*



SANs connect

- Clients and servers in a secure environment (e.g., intranet)
- Parallel server nodes

\+ Shorter path through software protocols

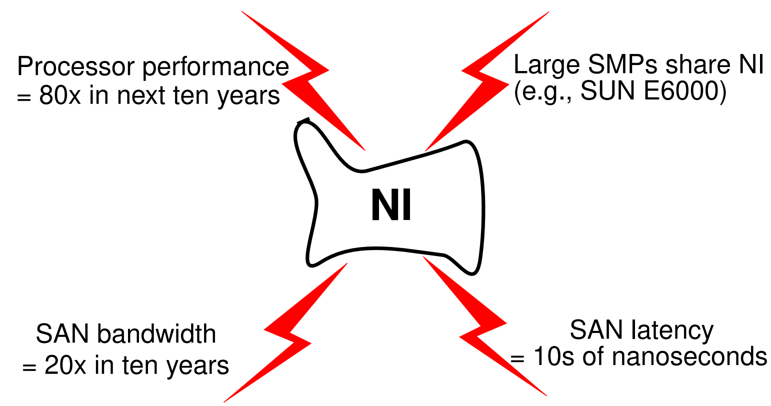
- e.g., most reliability guarantees move out of software

\+ Faster path through network

=> *Processor Accesses to NI Memory become critical bottleneck*

*. Gordon Bell coined the phrase "System Area Networks" [Communications of the ACM, March 1996]

Future NI Bottleneck



Treat NI Access as Memory Access

I/O Access: > 10 - 100 μs	**Memory Access: < 1 μs**
Through operating system (OS)	Direct, protected user access
Device on I/O bus	Main memory on memory bus
Uncached NI registers	Cache NI registers
No OOO[a] access & speculation	OOO access & speculation
API = data movement	Memory-based queues
Limited device memory	Plentiful
Ad hoc data movement	Cache block transfers
Notification thru' interrupts	Cache invalidations

a. OOO = Out-Of-Order load/store. Most modern microprocessors support OOO access and speculation.

Our contribution = organized opportunities in common framework

Steer Messages via Virtual Memory, Not via the OS

Messages through OS (e.g., through Unix sockets)

+ Easy protection
+ Easy address translation for message buffers
- High latency to access NI (e.g., high DMA initiation overhead)

Use virtual memory to access NI registers

- examples: TMC CM-5, Princeton SHRIMP, Arizona ADC, Cornell U-Net
- Protection: memory map NI memory
- Translation: at page granularity

+ Rapid access to NI

Move NI from I/O to Memory Bus

I/O bus

- high latency to access I/O bus device
  - traverse memory bus, I/O bridge, & I/O bus
  - I/O bus much slower than memory bus
- low bandwidth (e.g., PCI = 111 MB/s peak)

+ standard interface

Memory Bus (e.g., TMC CM-5)

+ low latency
+ high bandwidth (e.g., SUN Gigaplane = 2.6 GB/s sustained)
+ cache coherence protocol ...

- non-standard interface, but bridges possible
  - e.g., I/O bridge converts proprietary memory bus signals to I/O bus signals
  - directly attach I/O device to bridge (e.g., Intel's Accelerated Graphics Port)
  - other alternatives possible ...

Cache NI Registers in Processor & NI Caches

NI Registers: why uncached?

- no coherence on I/O buses
- side-effects (e.g., message send) ....(later)
- easy clear-on-read semantics

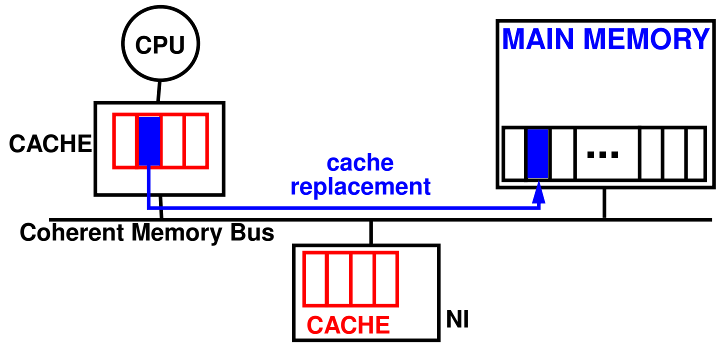
- high latency & low bandwidth

Cache NI Registers in processor caches [Mukherjee, et al, ISCA 1996]

- explicit handshake to reuse cached NI register
  - cost of handshake can be amortized

+ high performance ...

Cache NI Registers



+ Exploit temporal locality (status & control registers)
+ Transfer messages in cache block units (e.g., 32 - 128 bytes)
+ Automatic buffering
+ Out-of-order access, speculative loads ....(later)

Allow OOO Access & Speculation for NI register access

Absence of OOO/Speculation on I/O access

- I/O buses do not usually support split-transactions
- I/O space is usually uncached
- Side-effects (e.g., TMC CM-5, Princeton User-Level DMA)
  - In-order access
  - Single point of access
  - NI register access overloaded with device commands

Solutions

place NI on memory bus

cache NI memory

design API carefully to avoid side-effects ....(next)

Use Memory-Based Queues as API

API = Application Programming Interface

User-Level NI Access

Program-Controlled I/O (uncached load/store)

Princeton SHRIMP's User-Level DMA

Use Memory-Based Queue as user API

e.g., Application Device Channels (Druschel, et al. SIGCOMM '94)

+ Decouples NI and processor
+ Avoids side-effects => enables caching, OOO/speculation
  - device commands are simple memory operations
  - e.g., to send message increment tail pointer of send queue
+ Simple reuse handshake
+ Simplifies multiprogramming

Use Virtual Memory to Buffer Messages

NI can require multi-megabytes of message buffer space

- mismatch in rate of message generation, transfer, & consumption
  - independent growth rate of commodity components (CPU, SAN)
  - wide variety of communication protocols & applications
- bursts of messages
- large degree of multiprogramming
- flow control (e.g., return-to-sender, Dally, et al. M-Machine Architecture v1.0)

=> Use Virtual Memory as large message buffer

Problems

protected NI access to main memory

address translation for user virtual addresses in NI

Transfer Messages Between Processor and NI in Cache Blocks

I/O Transfer mechanisms (largely ad hoc)

- Uncached loads/stores transfer very few bytes
- DMA incurs high initiation
- User-Level DMA has side-effects, data steered via memory, etc.
- Special block transfer instructions (e.g., ultraSPARC)

Use cache block transfers

- Memory buses optimized for cache block transfers

Today DMA usually uses cache block transfers

- Coherence

Notification via Interrupts is Expensive

Processor executes 100s of instructions for an interrupt

- activate hardware interrupt line
- trap to OS
- save user state
- find and execute trap handler
- restore user state

Switching to OS pollutes hardware structures

- e.g., caches, TLB, speculation tables

Interrupts are considered exceptions, not common case

Use Cache Invalidations

Some systems allow polling on uncached status register

+ cheaper than interrupts
- more expensive than cache hits
- ties up system resources

Poll on cached status register

Use cache invalidation as notification

- NI invalidates cached status register in processor's cache
- Processor incurs cache miss, retrieves new status from NI
- Hybrid between interrupts and cache invalidations possible

NI can use "virtual polling"

- NI starts polling after invalidation

Conclusions

Treat NI Access as Memory Access

- Virtualize NI using virtual memory hardware
- Place NI on memory bus
- Allow processors & NIs to cache NI registers
- Allow OOO and speculative accesses to NI registers
- Use memory-based queues as API
- Use virtual memory to buffer network messages
- Transfer data between processor and NI in cache block units
- Use cache invalidations as notification signals

A Case for Making Network Interfaces Less Peripheral. Shubhendu S. Mukherjee and Mark D. Hill. Submitted for publication.

A Survey of User-Level Network Interfaces for System Area Networks. Shubhendu S. Mukherjee and Mark D. Hill. Computer Sciences Department, UW-Madison Technical Report, # 1340.

Related Work

Virtualize NI using virtual memory hardware

TMC CM-5, Princeton SHRIMP, Arizona ADC, Cornell U-Net
Wisconsin CNI (ISCA 1996)

Place NI on memory bus (or closer)

TMC CM-5, Intel Teraflop, Stanford FLASH, MIT J-machine,
MIT M-machine, Henry & Joerg (ASPLOS 1992),
Illinois DI-Multicomputer, Forth Telegraphos, Wisconsin CNI

Allow processors & NIs to cache NI registers

Wisconsin CNI

Allow OOO and speculative accesses to NI registers

Not aware of any

Related Work (contd.)

Use memory-based queues as API

Arizona ADC, Cornell U-Net, Liu & Culler (ISUG 1995),
Wisconsin CNI

Use virtual memory as a large message buffer space

MIT Fugu, Princeton SHRIMP, Cornell U-Net

Transfer data between processor and NI in cache block units

Wisconsin CNI

Use cache invalidations as notification signals

Wisconsin CNI