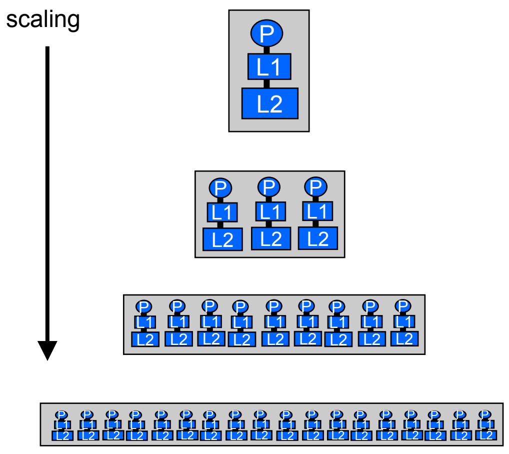
18-742
Lecture 13

Scalable MP Case Studies

Spring 2005
Prof. Babak Falsafi
http://www.ece.cmu.edu/~ece742



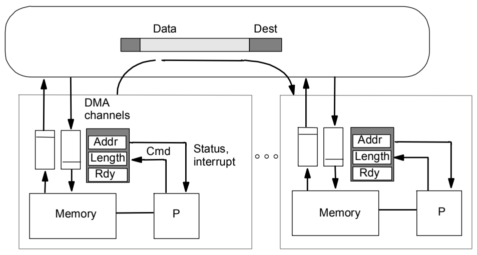
Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

Exam Coverage

Chapters 1-6 from the book
Readers 1, 2, and 3
Topics:
- **Convergence of machines**
- **Parallel programming models**
- **Symmetric Multiprocessors: Basics**
- **Synchronization**
- **Symmetric Multiprocessors: Implementations**

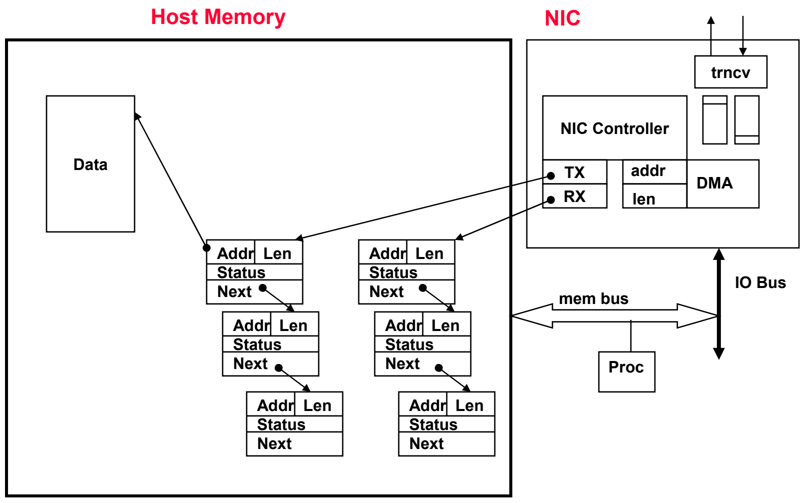
Closed book, closed notes
No calculators

Net Transactions: Physical DMA

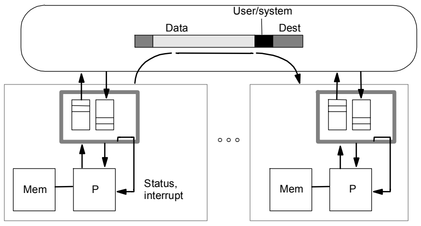


- **Physical addresses: OS must initiate transfers**
  - **system call per message on both ends: ouch**
- **Sending OS copies data to kernel buffer w/ header/trailer**
  - **can avoid copy if interface does scatter/gather**
- **Receiver copies packet into OS buffer, then interprets**
  - **user message then copied (or mapped) into user space**

Conventional LAN Network Interface

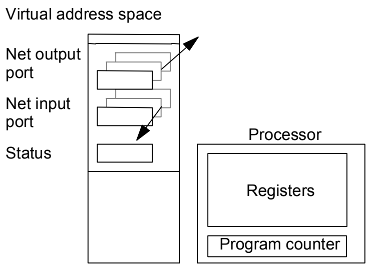


User Level Ports



- map network hardware into user's address space
  - talk directly to network via loads & stores
- user-to-user communication without OS intervention: low latency
- protection: user/user & user/system
- DMA hard… CPU involvement (copying) becomes bottleneck

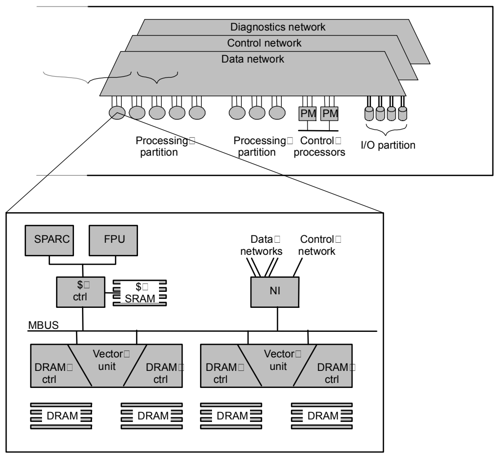
User Level Network ports



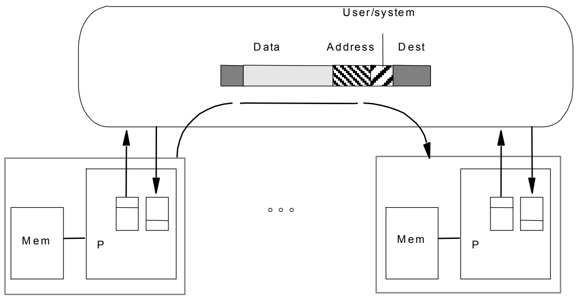
- Appears to user as logical message queues plus status
- What happens if no user pop?

Example: CM-5

- Input and output FIFO for each network
- Two data networks
- Save/restore network buffers on context switch

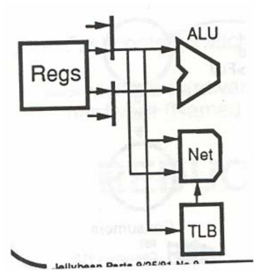


User Level Handlers



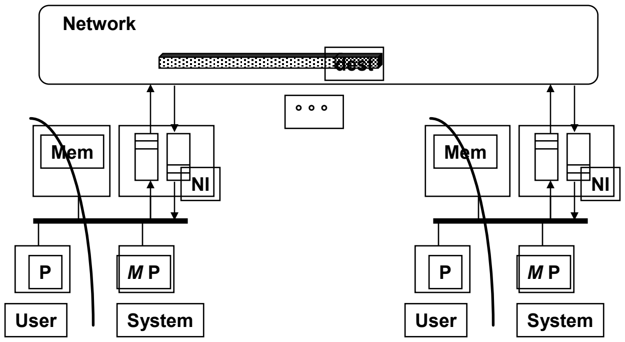
- Hardware support to vector to address specified in message
  - message ports in registers
  - alternate register set for handler?
- Examples: J-Machine, Monsoon, *T (MIT), iWARP (CMU)

J-Machine



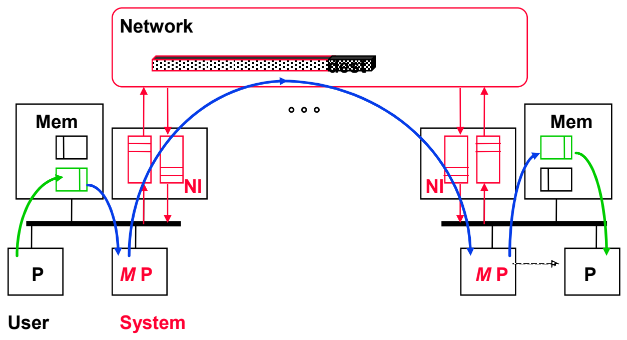
- Each node a small message-driven processor
- HW support to queue msgs and dispatch to msg handler task

Dedicated Message Processing Without Specialized Hardware



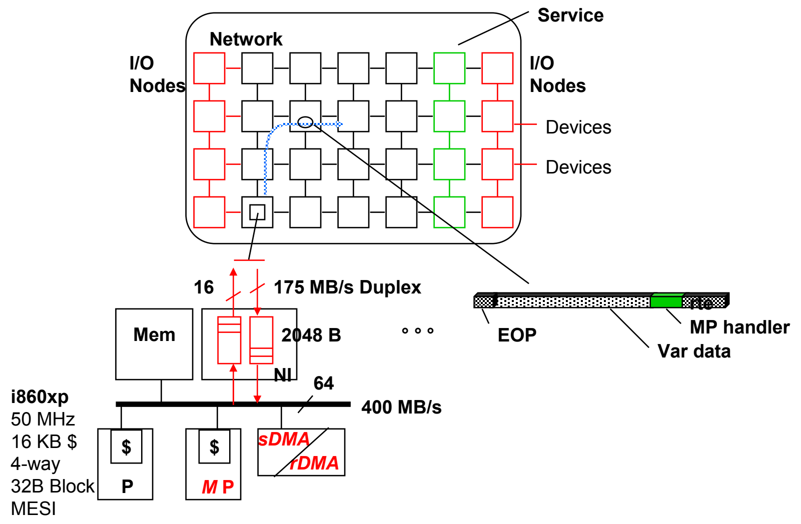
- Microprocessor performs arbitrary output processing (at system level)
- Microprocessor interprets incoming network transactions (in system)
- User Processor <–> Msg Processor share memory
- Msg Processor <–> Msg Processor via system network transaction

Levels of Network Transaction



- User Processor stores cmd / msg / data into shared output queue
  - must still check for output queue full (or grow dynamically)
- Communication assists make transaction happen
  - checking, translation, scheduling, transport, interpretation
- Avoid system call overhead
- Multiple bus crossings likely bottleneck

Example: Intel Paragon



Dedicated MP w/specialized NI: Meiko CS-2

- Integrate message processor into network interface
  - active messages-like capability
  - dedicated threads for DMA, reply handling, simple remote memory access
  - supports user-level virtual DMA
    - » own page table
    - » can take a page fault, signal OS, restart
      - meanwhile, nack other node
- Problem: processor is slow, time-slices threads
  - fundamental issue with building your own CPU

Myricom Myrinet (Berkeley NOW)

- Programmable network interface on I/O Bus (Sun SBUS or PCI)
  - embedded custom CPU (“Lanai”, ~40 MHz RISC CPU)
  - 256KB SRAM
  - 3 DMA engines: to network, from network, to/from host memory
- Downloadable firmware executes in kernel mode
  - includes source-based routing protocol
- SRAM pages can be mapped into user space
  - separate pages for separate processes
  - firmware can define status words, queues, etc.
    - » data for short messages or pointers for long ones
    - » firmware can do address translation too… w/OS help
  - poll to check for sends from user
- Bottom line: I/O bus still bottleneck, CPU could be faster

Shared Physical Address Space

- **Implement SAS model in hardware w/o caching**
  - actual caching must be done by copying from remote memory to local
  - programming paradigm looks more like message passing than Pthreads
    - » yet, low latency & low overhead transfers thanks to HW interpretation; high bandwidth too if done right
    - » result: great platform for MPI & compiled data-parallel codes
- **Implementation:**
  - "pseudo-memory" acts as memory controller for remote mem, converts accesses to network transaction (request)
  - "pseudo-CPU" on remote node receives requests, performs on local memory, sends reply
  - split-transaction or retry-capable bus required (or dual-ported mem)

Example: Cray T3D

- **Up to 2,048 Alpha 21064s**
  - no off-chip L2 to avoid inherent latency
- **In addition to remote mem ops, includes:**
  - prefetch buffer (hide remote latency)
  - DMA engine (requires OS trap)
  - synchronization operations (swap, fetch&inc, global AND/OR)
  - message queue (requires OS trap on receiver)
- **Big problem: physical address space**
  - 21064 supports only 32 bits
  - 2K-node machine limited to 2M per node
  - external "DTB annex" provides segment-like registers for extended addressing, but management is expensive & ugly

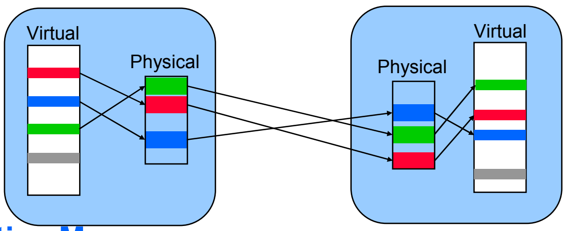
Cray T3E

- Similar to T3D, uses Alpha 21164 instead of 21064 (on-chip L2)
  - still has physical address space problems
- E-registers for remote communication and synchronization
  - 512 user, 128 system; 64 bits each
  - replace/unify DTB Annex, prefetch queue, block transfer engine, and remote load / store, message queue
  - Address specifies source or destination E-register and command
  - Data contains pointer to block of 4 E-regs and index for centrifuge
- Centrifuge
  - supports data distributions used in data-parallel languages (HPF)
  - 4 E-regs for global memory operation: mask, base, two arguments
- Get & Put Operations

T3E (continued)

- Atomic Memory operations
  - E-registers & centrifuge used
  - F&I, F&Add, Compare&Swap, Masked_Swap
- Messaging
  - arbitrary number of queues (user or system)
  - 64-byte messages
  - create msg queue by storing message control word to memory location
- Msg Send
  - construct data in aligned block of 8 E-regs
  - send like put, but dest must be message control word
  - processor is responsible for queue space (buffer management)
- Barrier and Eureka synchronization

DEC Memory Channel (Princeton SHRIMP)



- Reflective Memory
- Writes on Sender appear in Receiver's memory
  - send & receive regions
  - page control table
- Receive region is pinned in memory
- Requires duplicate writes, really just message buffers

Performance of Distributed Memory Machines

- Microbenchmarking
- One-way latency of small (five-word) message
  - echo test
  - round-trip divided by 2
- Shared Memory remote read
- Message Passing Operations
  - The LogP model (Culler et al.)
  - Os – observed send overhead
  - Or – observed receive overhead
  - L – observed network latency
  - Gap – observed pipelined time per req./resp.
    - » i.e., longest pipe stage?

Network Transaction Performance

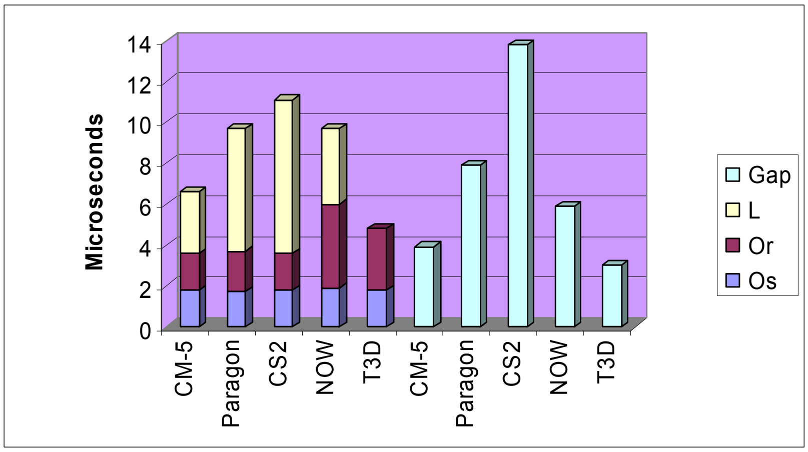


Figure 7.31

Remote Read Performance

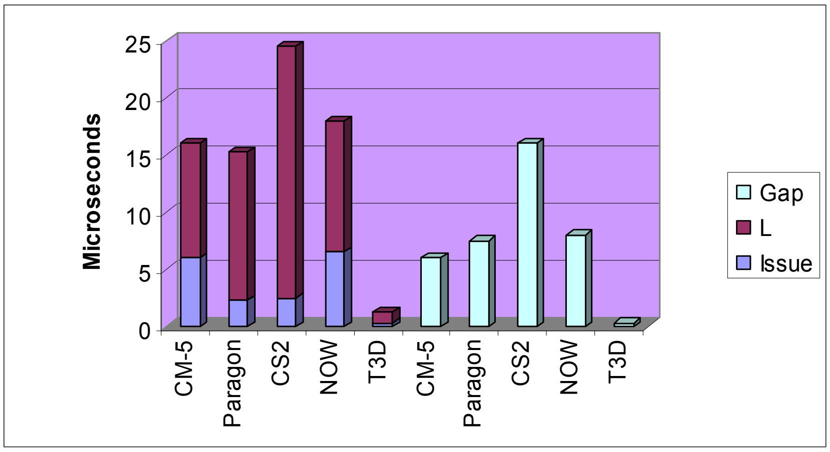


Figure 7.32

Summary of Distributed Memory Machines

- Convergence of architectures
  - everything "looks basically the same"
  - processor, cache, memory, communication assist
- Communication Assist
  - where is it? (I/O bus, memory bus, processor registers)
  - what does it know?
    - » does it just move bytes, or does it perform some functions?
  - is it programmable?
  - does it run user code?
- Network transaction
  - input & output buffering
  - action on remote node

Outline

- Motivation
- Network Transaction Primitive
- Supporting Programming Models
- Case Studies