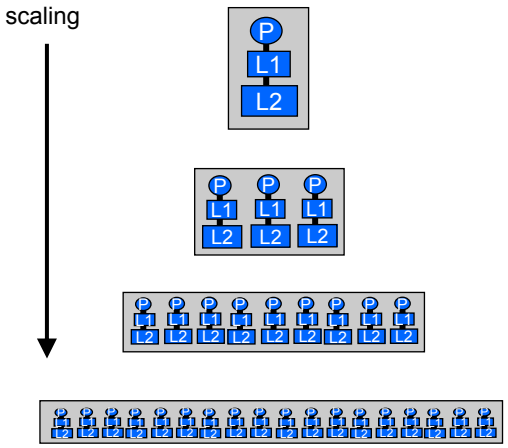
18-742

Lecture 12

Scalable Multiprocessors



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.

Readings

In-class midterm Wednesday of next week

Reading assignment homework will be posted today

- **Due on Monday**

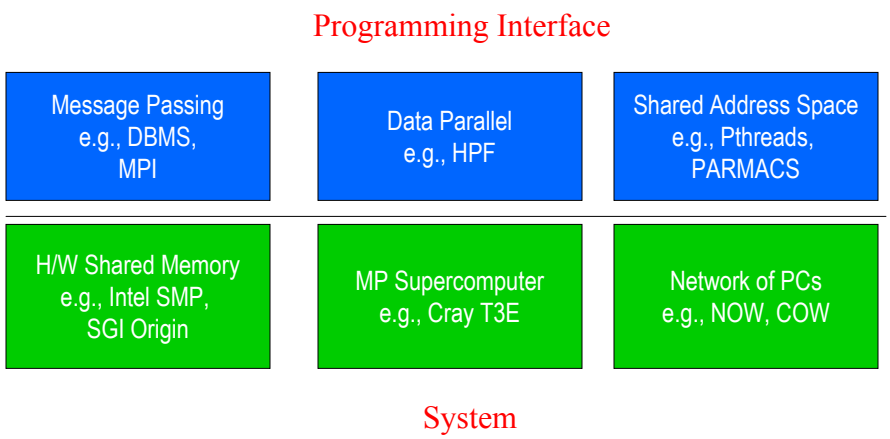
Chapter 7

Outline

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

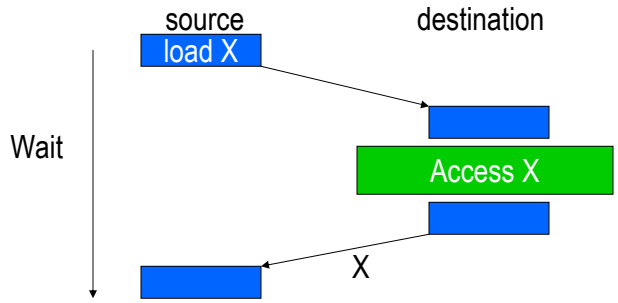
Shared Address Space vs. Message Passing

- **Do not confuse model/abstraction with system implementation:**



Shared Address Space Requirements

- Shared address space on a distributed memory computer
- Two-way request/response protocol:
  - basic: read a block, write a block, and block ownership
  - more complicated protocol optimizations on top of basic



Shared Address Space Issues

- Coherent or non-coherent:
  - fully cache-coherent is like an SMP
  - coherence granularity: cache block, page, etc.
  - remote shared accesses only: Cray T3D
- Does it allow overlapping multiple transactions:
  - a.k.a. memory consistency models: chapter 9
- Implemented in H/W or S/W:
  - For S/W systems, H/W provides messaging
  - H/W or S/W can provide protection, flow control, etc.

Shared Address Space Issues

Assume implemented in H/W

- **issues handled similar to the bus (see chapter 8)**
- **protection primarily enforced by virtual memory**
- **data address: sender & receiver NIs know what to do**
- **coherent:**
  - **block transfers from cache/memory to network**
  - **H/W protocol implements coherence**
- **non-coherent:**
  - **single/double word reads and writes from proc to network**

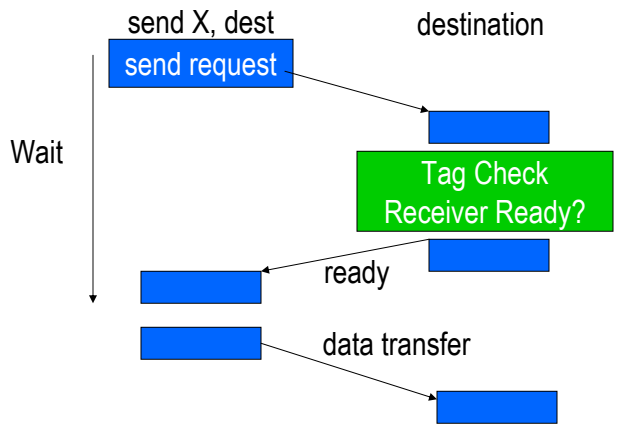
Support for larger transfers using DMA H/W?

Message Passing Requirements

- **Messaging abstraction on a distributed memory computer**
- **Primarily a one-way communication:**
  - **sender sends variable-sized message**
  - **receiver receives**
- **But,**
  - **can't implement it that way**
  - **why?**
- **Asynchronous vs Synchronous messaging**
  - **synchronous: sender waits until receiver arrives**
  - **asynchronous: sender sends and goes back to computation**

Message Passing Requirements

- **Synchronous messaging**

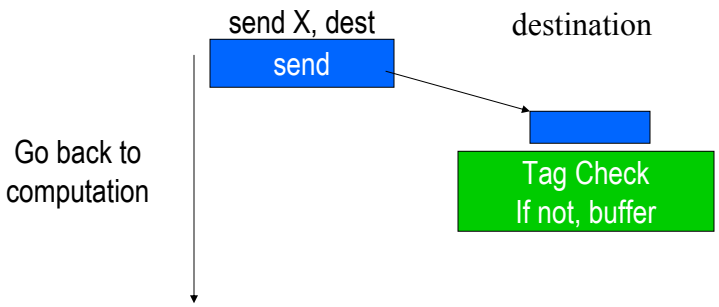


Message Passing Issues

- **What is involved in the process?**
- **Sender must specify where the data is:**
  - **typically in contiguous block of memory**
  - **can optimize for direct transfers from data structures: gather/scatter**
- **Proc or NI must transfer from memory to NI buffers**
  - **what about NI protection?**
  - **Can send through OS: but requires multiple data copies**
- **Does NI have access to application memory?**
- **Unless support for low-overhead messaging => must send large messages to amortize the startup overhead**

Message Passing Requirements

- **Asynchronous messaging: (optimistic)**



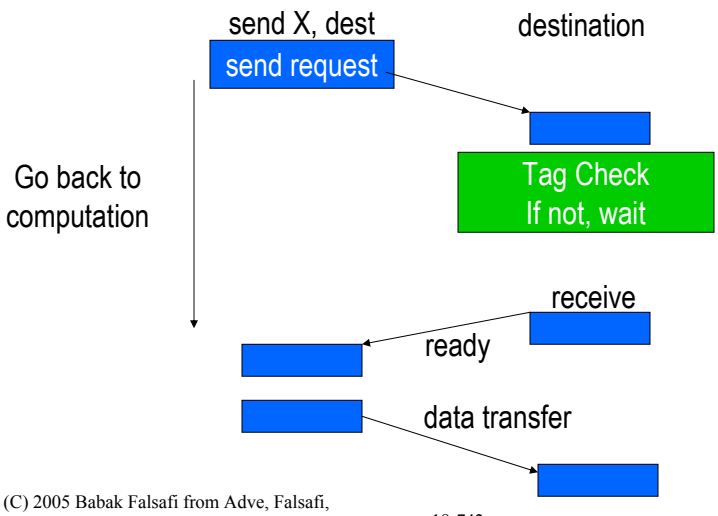
What is wrong with this approach?

Message Passing Issues

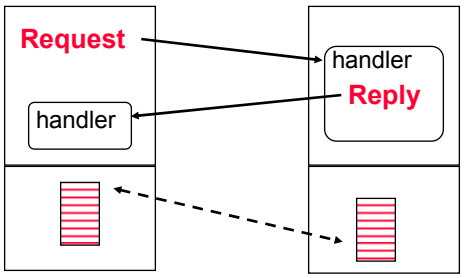
- **Must perform a tag match in software to find where to go:**
  - **slow process**
  - **can always provide buffer for later processing**
  - **how much buffering is enough?**
- **Data typically arrives at a high rate for large messages!**
- **What if multiple senders are sending to one receiver?**
- **Need flow control or can drop messages!**

Message Passing Requirements

- **Asynchronous messaging: (optimistic)**



Active Messages

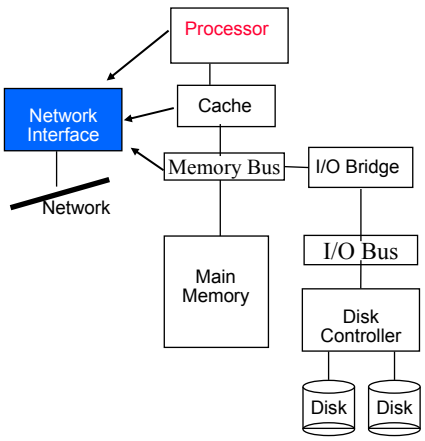


- **User-level analog of network transaction**
  - **invoke handler function at receiver to extract packet from network**
  - **grew out of attempts to do dataflow programming on msg-passing machines**
  - **handler may send reply, but no other messages**
- **Event notification: interrupts, polling, events?**
- **May also perform memory-to-memory transfer**

Outline

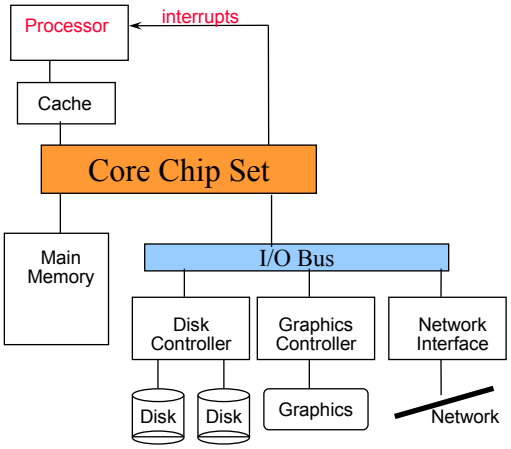
- **Motivation**
- **Network Transaction Primitive**
- **Supporting Programming Models**
- **Case Studies**

Massively Parallel Processor (MPP) Architectures



- **Network interface typically close to processor**
  - **Memory bus:**
    - » **locked to specific processor architecture/bus protocol**
  - **Registers/cache:**
    - » **only in research machines**
- **Time-to-market is long**
  - **processor already available or work closely with processor designers**
- **Maximize performance and cost**

Network of Workstations



- Network interface on I/O bus
- Standards (e.g., PCI) => longer life, faster to market
- Slow (microseconds) to access network interface
- "System Area Network" (SAN): between LAN & MPP

Transaction Interpretation

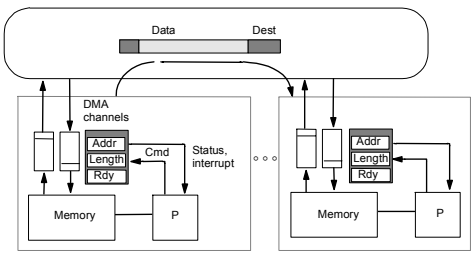
- Simplest MP: assist doesn't interpret much if anything
  - DMA from/to buffer, interrupt or set flag on completion
- User-level messaging: get the OS out of the way
  - assist does protection checks to allow direct user access to network
  - may have minimal interpretation otherwise
- Virtual DMA: get the CPU out of the way (maybe)
  - basic protection plus address translation: user-level bulk DMA
- Global physical address space (NUMA): everything in hardware
  - complexity increases, but performance does too (if done right)
- Cache coherence: even more so
  - stay tuned

Spectrum of Designs

Increasing HW Support, Specialization, Intrusiveness, Performance (???)

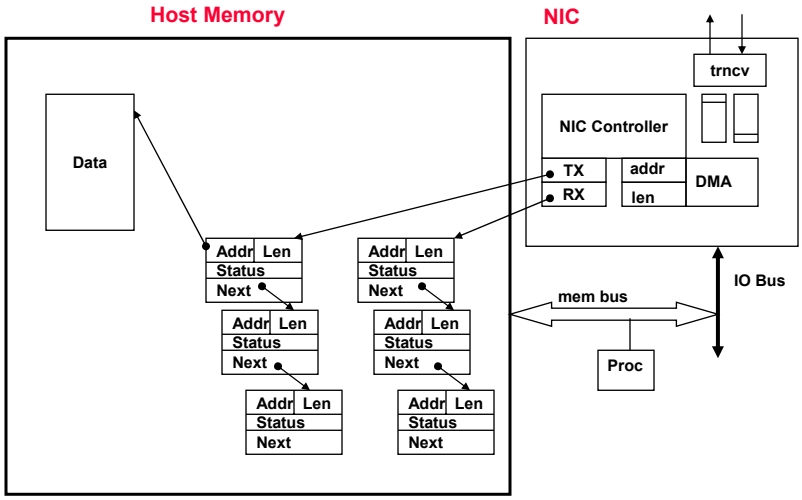
- **None: Physical bit stream**
  - physical DMA — nCUBE, iPSC, . . .
- **User/System**
  - User-level port — CM-5, *T
  - User-level handler — J-Machine, Monsoon, . . .
- **Remote virtual address**
  - Processing, translation — Paragon, Meiko CS-2, Myrinet
  - Reflective memory (?) — Memory Channel, SHRIMP
- **Global physical address**
  - Proc + Memory controller — RP3, BBN, T3D, T3E
- **Cache-to-cache (later)**
  - Cache controller — Dash, KSR, Flash

Net Transactions: Physical DMA

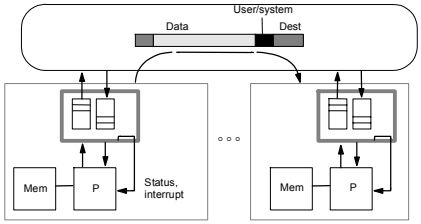


- **Physical addresses: OS must initiate transfers**
  - system call per message on both ends: ouch
- **Sending OS copies data to kernel buffer w/ header/trailer**
  - can avoid copy if interface does scatter/gather
- **Receiver copies packet into OS buffer, then interprets**
  - user message then copied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

Conventional LAN Network Interface

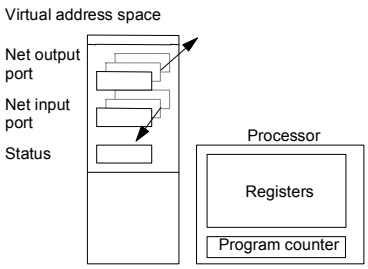


User Level Ports



- **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**

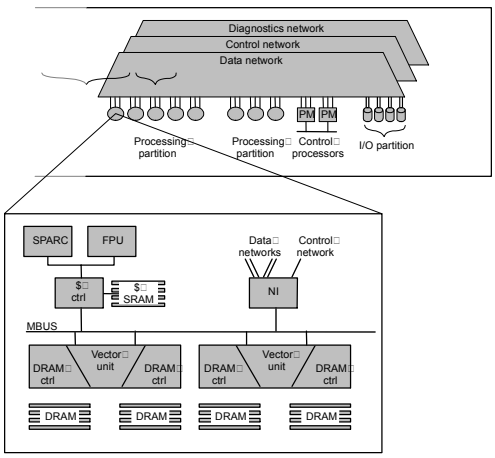
User Level Network ports



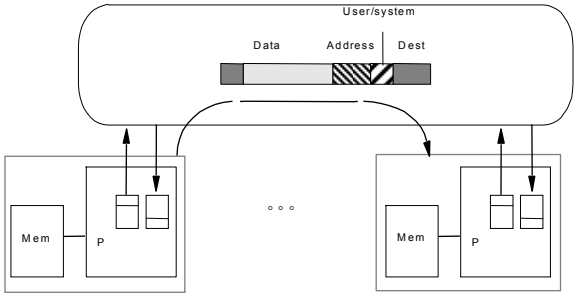
- **Appears to user as logical message queues plus status**
- **What happens if no user pop?**

Example: CM-5

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

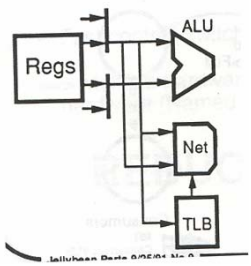


User Level Handlers



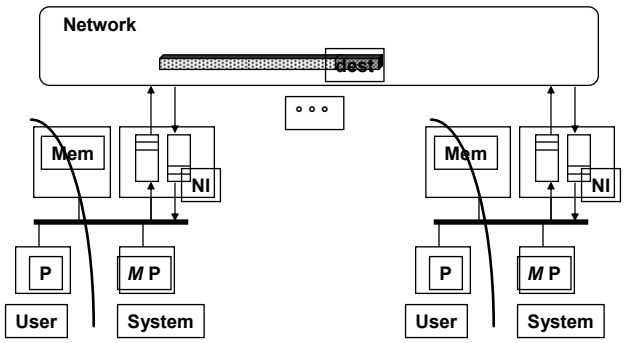
- **Hardware support to vector to address specified in message**
  - **message ports in registers**
  - **alternate register set for handler?**
- **Examples: J-Machine, Monsoon, *T (MIT), iWARP (CMU)**

J-Machine



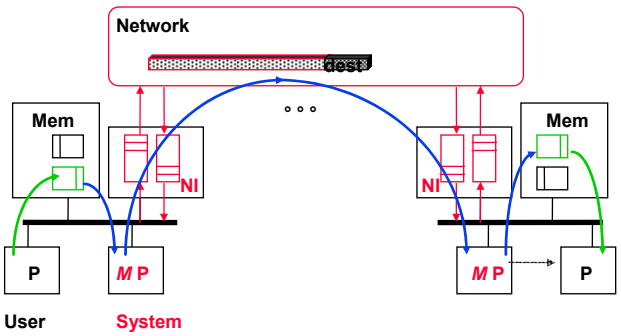
- **Each node a small message-driven processor**
- **HW support to queue msgs and dispatch to msg handler task**

Dedicated Message Processing Without Specialized Hardware



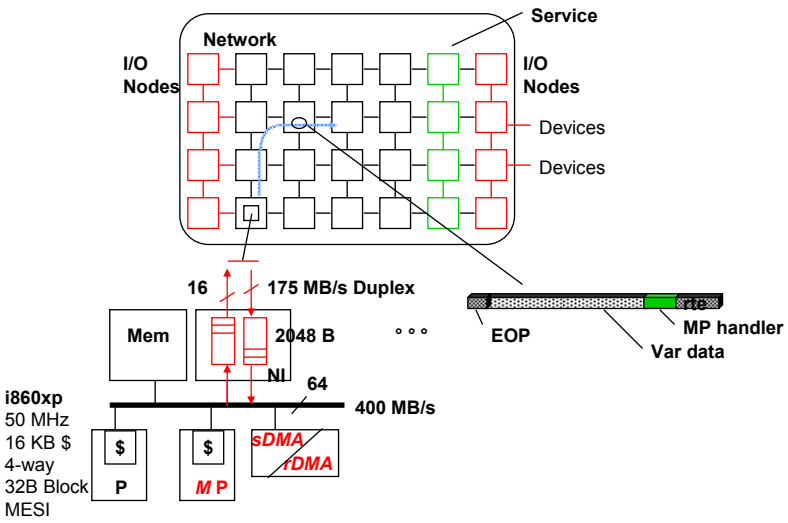
- Microprocessor performs arbitrary output processing (at system level)
- Microprocessor interprets incoming network transactions (in system)
- User Processor <–> Msg Processor share memory
- Msg Processor <–> Msg Processor via system network transaction

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**