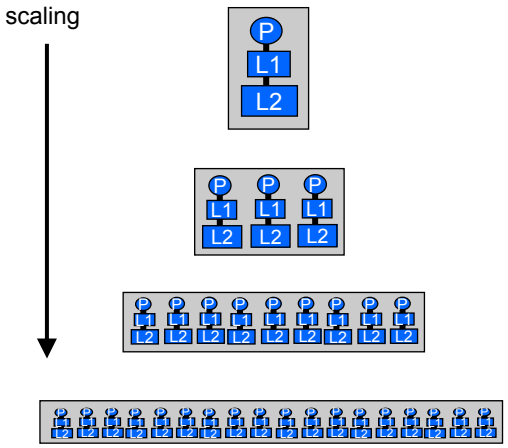
18-742

Lecture 11

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.

Announcements & Readings

In-class midterm Wednesday of next week

Starting Chapter 7 of the book

Reader 4

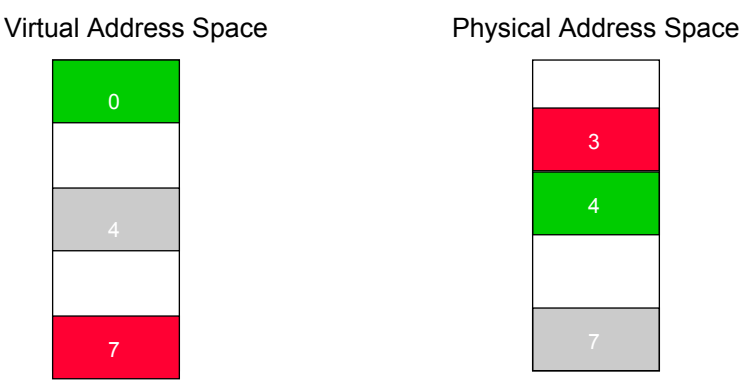
- **M. A. Blumrich, R. Alpert, Y. Chen, D. W. Clark, S. N. Damianakis, C. Dubnicki, E. W. Felten, L. Iftode, K. Li, M. Martonosi, and R. A. Shillner, *Design Choices in the SHRIMP System: An Empirical Study*, ISCA 1998.**
- **L. A. Barroso, J. Dean, U. Holzle, *Web Search for a Planet: The Google Cluster Architecture*, IEEE Micro 23(2): 22-28.**

Outline

- Coherence Control Implementation
- Writebacks, Non-Atomicity, & Serialization/Order
- Hierarchical Cache
- Split Buses
- Deadlock, Livelock, & Starvation
- Three Case Studies
- TLB Coherence
- Virtual Cache Issues

Translation Lookaside Buffer

- Cache of Page Table Entries
- Page Table Maps Virtual Page to Physical Frame



The TLB Coherence Problem

- Since TLB is a cache, must be kept coherent
- Change of PTE on one processor must be seen by all processors
- Process migration
- Changes are infrequent
  - get OS to do it
  - Always flush TLB is often adequate

TLB Shootdown

- To modify TLB entry, modifying processor must
  - LOCK page table,
  - flush TLB entries,
  - queue TLB operations,
  - send interprocessor interrupt,
  - spin until other processors are done
  - UNLOCK page table
- SLOW...
  - But most common solution today
- Some ISAs have “flush TLB entry” instructions

Virtual Caches & Synonyms

- **Problem**
  - **Synonyms: V0 & V1 map to P1**
  - **When doing coherence on block in P1 how do you find V0 & V1?**
- **Don't do virtual caches (most common today)**
- **Don't allow synonyms**
  - **Constrains software (and OS assumptions)**
- **Allow virtual cache & synonyms**
  - **How implement reverse address translation?**
  - **See Wang et al. next**

Wang et al. [ISCA89]

- **Basic Idea**
  - **Virtual L1 and physical L2**
  - **Do coherence on physical addresses**
  - **Each L2 block maintains backpointer to corresponding L1 block (if any)**
    (requires log2 #L1_blocks - log2 (page_size / block_size)
  - **Never allow block to be simultaneously cached under synonyms**
- **Example where V0 & V1 map to P2**
  - **Initially V1 in L1 and P2 in L1 points to V1**
  - **Processor references V0**
  - **L1 miss**
  - **L2 detects synonym in L1**
  - **Change L1 tag and L2 pointer so that L1 has V0 instead of V1**
  - **Resume**

Virtual Caches & Homonyms

- **Homonym**
  - **V0 of one process maps to P2, while V0 of other process maps to P3**
- **Flush cache on context switch**
  - **simple but performs poorly**
- **Address-space IDs (ASIDs)**
  - **in architecture & part of context state**

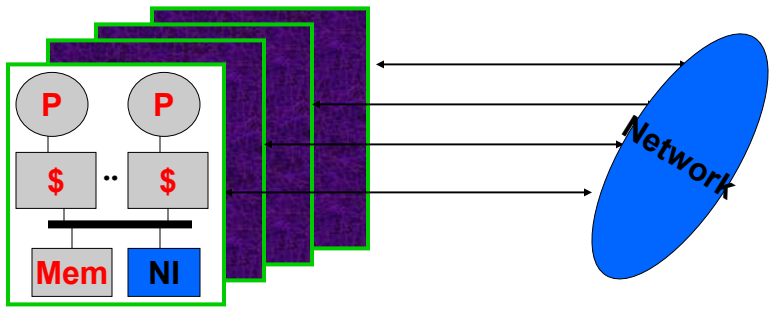
Outline

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

Scalable Systems

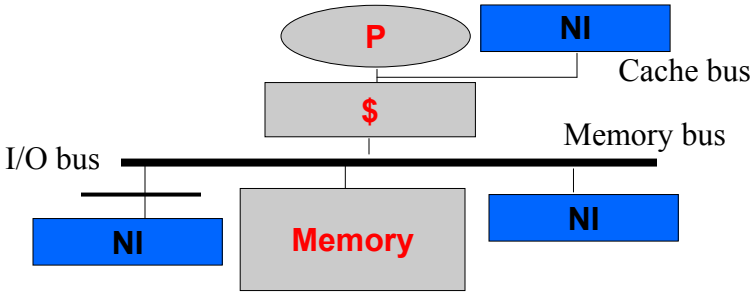
- Buses support up to 8 to 16 processors well
- What about performance after 16 processors?
- What about cost?
- How do we build larger systems that are cost-effective?

Scalable Systems



- Cluster of workstation/server like nodes
- Commodity System-Area/Local-Area Network Switches/Fiber
- What are the hardware/software issues?

Node Communication Architecture



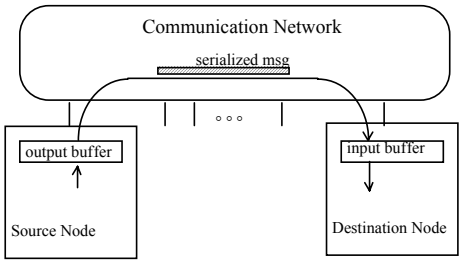
- How does a processor communicate with others?
- Is there processor/memory support for communication?
- Where does the network interface sit?

Communication Architecture Alternatives

Processor/memory support for communication:

- NI on I/O bus: Berkeley NOW, Wisconsin COW
- NI on memory bus: Intel Paragon, TMC CM-5
- Support for protected messaging: Princeton SHRIMP
- TLB support for shared address space: Cray T3D, T3E
- Full support for messaging: MIT J-Machine, nCube/2
- Full support for coherent shared memory: Alpha 21364

Network Transaction Primitive



- One-way transfer of information from source to destination
- causes some action at the destination
  - process info and/or deposit in buffer
  - state change (e.g., set flag, interrupt program)
  - maybe initiate reply (separate network transaction)

Node-to-Node Architecture

- All architectures use point-to-point messaging
- How is this different from a bus?
- Communication architecture issues:

- Protection
- Format
- Output buffering
- Input buffering
- Media arbitration & flow control
- Destination name & routing
- Action
- Completion detection
- Deadlock avoidance
- Delivery guarantees

Communication Architecture

- **Protection:**
  - **larger and more loosely coupled components**
  - **Where is data from? Where should it go?**
  - **How do you isolate faulty hardware or software?**
- **Format:**
  - **Message descriptor and length**
  - **Flexibility in the contents depending communication model**
  - **e.g., shared memory messages: commands vs data**
  - **e.g., Active Messages in WWT: src, dest, PC, args**

Communication Architecture

- **Buffering required:**
  - **when there is mismatch between arrival/deprature rate**
- **Output buffers:**
  - **data from processor/memory going to the network**
  - **what happens when output buffers fill up?**
- **Input buffers:**
  - **incoming data from other procs**
  - **may be picked it by the proc or sent to memory**
  - **what happens when input buffers fill up?**

Communication Architecture

- **Media arbitration & flow control:**
  - distributed among the network switches
  - packets traverse through multiple switches
  - (flow control later)
- **Destination name and routing:**
  - the source must specify destination name and routing
  - e.g., shared memory, a physical memory address & a proc id
  - (routing later)
- **Action:**
  - may be memory write or read, executing code
  - may also send a response

Communication Architecture

- **Completion detection:**
  - sender must know message is received
  - receiver must know there is a message
  - interrupt vs polling! (will revisit when covering CNI)
- **Transaction ordering:**
  - do all transactions from source to dest arrive in order
  - must distinguish between out-of-order routing vs delivery
  - e.g., what happens to shared-memory protocols if out of order

Communication Architecture

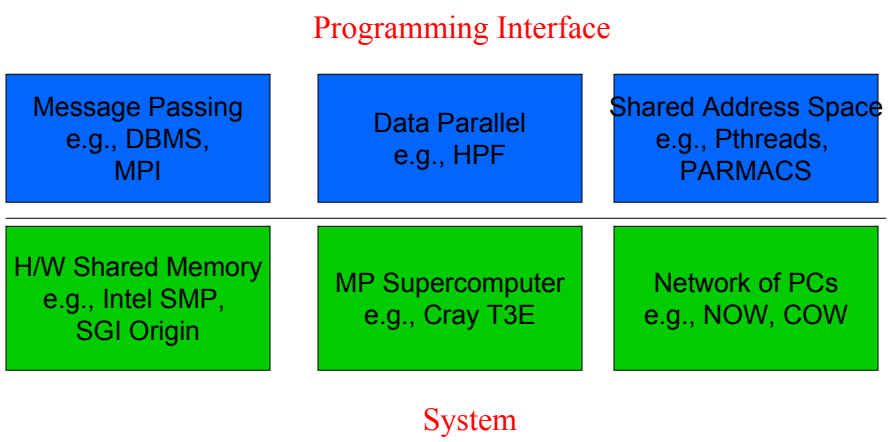
- **Deadlock avoidance:**
  - switches require buffer space for routing
  - must manage buffers carefully to avoid deadlocks
  - end-to-end: sender must always receive while sending!
- **Delivery guarantees:**
  - when input buffers fill up: drop message or control flow
  - what about data integrity?

Outline

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

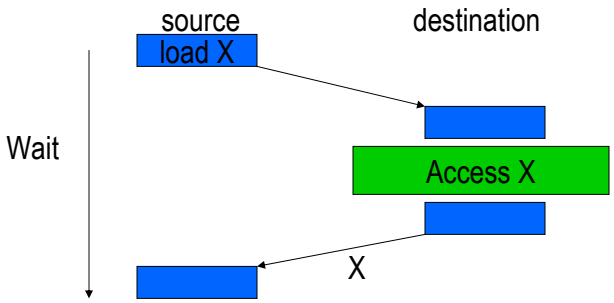
Shared Address Space vs Message Passing

- **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

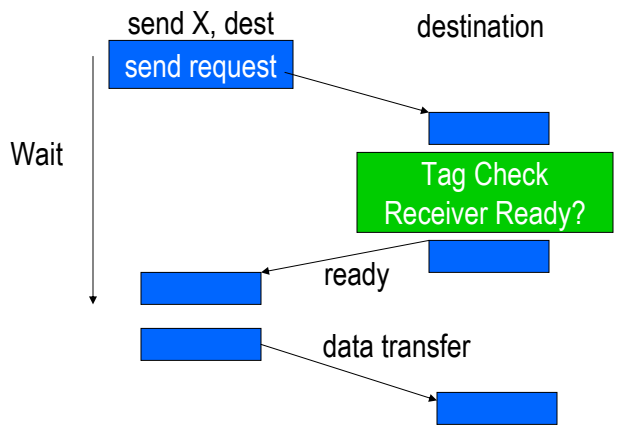
- **Assume implemented in H/W**
- **issues handled similar to the bus (see chapter 8)**
- **protection primarily enforced by virtual memory**
- **data address: sender & receiver NIs know what to do**
- **coherent:**
  - **block transfers from cache/memory to network**
  - **H/W protocol implements coherence**
- **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**

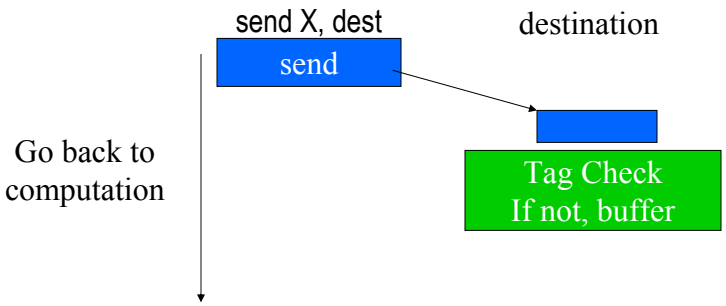


Message Passing Issues

- 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)



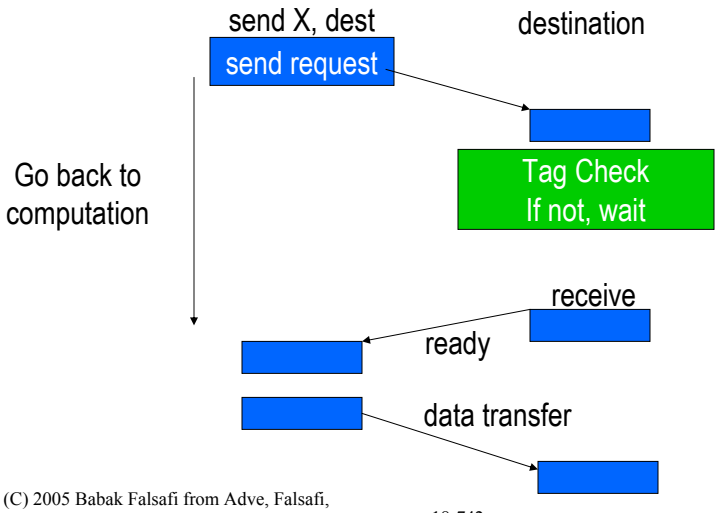
What is wrong with this approach?

Message Passing Issues

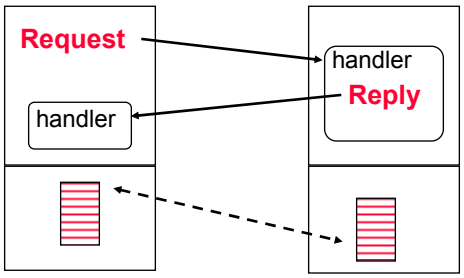
- **Must perform a tag match in software to find where to go:**
  - **slow process**
  - **can always provide buffer for later processing**
  - **how much buffering is enough?**
- **Data typically arrives at a high rate for large messages!**
- **What if multiple senders are sending to one receiver?**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