Lecture 5: Communication Architectures and Communication Costs

The Lecture Contains:

Previous Next

Fundamentals of Parallel Computers

Agenda

- Convergence of parallel architectures
- Fundamental design issues
- ILP vs. TLP

Communication Architecture

- Historically, parallel architectures are tied to programming models
  - Diverse designs made it impossible to write portable parallel software
  - But the driving force was the same: need for fast processing
- Today parallel architecture is seen as an extension of microprocessor architecture with a communication architecture
  - Defines the basic communication and synchronization operations and provides hw/ sw implementation of those

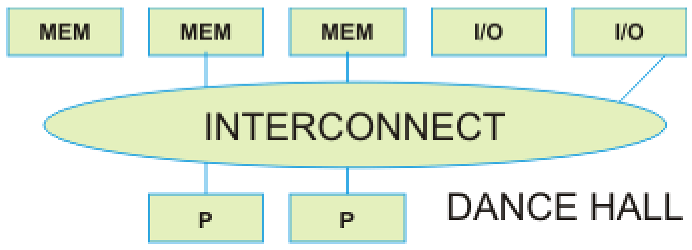
Layered Architecture

- A parallel architecture can be divided into several layers
  - Parallel applications
  - Programming models: shared address, message passing, multiprogramming, data parallel, dataflow etc
  - Compiler + libraries
  - Operating systems support
  - Communication hardware
  - Physical communication medium
- Communication architecture = user/system interface + hw implementation (roughly defined by the last four layers)
  - Compiler and OS provide the user interface to communicate between and synchronize threads

Previous Next

Shared Address

- Communication takes place through a logically shared portion of memory
    - User interface is normal load/store instructions
    - Load/store instructions generate virtual addresses
    - The VAs are translated to PAs by TLB or page table
    - The memory controller then decides where to find this PA
    - Actual communication is hidden from the programmer
- The general communication hw consists of multiple processors connected over some medium so that they can talk to memory banks and I/O devices
    - The architecture of the interconnect may vary depending on projected cost and target performance
- Communication medium



    - Interconnect could be a crossbar switch so that any processor can talk to any memory bank in one “hop” (provides latency and bandwidth advantages)
    - Scaling a crossbar becomes a problem: cost is proportional to square of the size
    - Instead, could use a scalable switch-based network; latency increases and bandwidth decreases because now multiple processors contend for switch ports

Previous Next

Shared Address

- Communication medium
  - From mid 80s shared bus became popular leading to the design of SMPs
  - Pentium Pro Quad was the first commodity SMP
  - Sun Enterprise server provided a highly pipelined wide shared bus for scalability reasons; it also distributed the memory to each processor, but there was no local bus on the boards i.e. the memory was still “symmetric” (must use the shared bus)
  - NUMA or DSM architectures provide a better solution to the scalability problem; the symmetric view is replaced by local and remote memory and each node (containing processor(s) with caches, memory controller and router) gets connected via a scalable network (mesh, ring etc.); Examples include Cray/SGI T3E, SGI Origin 2000, Alpha GS320, Alpha/HP GS1280 etc.

Message Passing

- Very popular for large-scale computing
- The system architecture looks exactly same as DSM, but there is no shared memory
- The user interface is via send/receive calls to the message layer
- The message layer is integrated to the I/O system instead of the memory system
- Send specifies a local data buffer that needs to be transmitted; send also specifies a tag
- A matching receive at dest . node with the same tag reads in the data from kernel space buffer to user memory
- Effectively, provides a memory-to-memory copy

Previous Next

Message Passing

- Actual implementation of message layer
    - Initially it was very topology dependent
    - A node could talk only to its neighbors through FIFO buffers
    - These buffers were small in size and therefore while sending a message send would occasionally block waiting for the receive to start reading the buffer (synchronous message passing)
    - Soon the FIFO buffers got replaced by DMA (direct memory access) transfers so that a send can initiate a transfer from memory to I/O buffers and finish immediately (DMA happens in background); same applies to the receiving end also
    - The parallel algorithms were designed specifically for certain topologies: a big problem

- To improve usability of machines, the message layer started providing support for arbitrary source and destination (not just nearest neighbors)
    - Essentially involved storing a message in intermediate “hops” and forwarding it to the next node on the route
    - Later this store-and-forward routing got moved to hardware where a switch could handle all the routing activities
    - Further improved to do pipelined wormhole routing so that the time taken to traverse the intermediate hops became small compared to the time it takes to push the message from processor to network (limited by node-to-network bandwidth)
    - Examples include IBM SP2, Intel Paragon
    - Each node of Paragon had two i860 processors, one of which was dedicated to servicing the network (send/ recv . etc.)

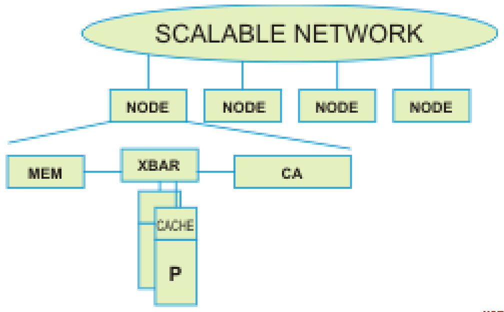
Previous Next

Convergence

- Shared address and message passing are two distinct programming models, but the architectures look very similar
  - Both have a communication assist or network interface to initiate messages or transactions
  - In shared memory this assist is integrated with the memory controller
  - In message passing this assist normally used to be integrated with the I/O, but the trend is changing
  - There are message passing machines where the assist sits on the memory bus or machines where DMA over network is supported (direct transfer from source memory to destination memory)
  - Finally, it is possible to emulate send/ recv . on shared memory through shared buffers, flags and locks
  - Possible to emulate a shared virtual mem. on message passing machines through modified page fault handlers

A Generic Architecture

- In all the architectures we have discussed thus far a node essentially contains processor(s) + caches, memory and a communication assist (CA)
  - CA = network interface (NI) + communication controller
- The nodes are connected over a scalable network
- The main difference remains in the architecture of the CA
  - And even under a particular programming model (e.g., shared memory) there is a lot of choices in the design of the CA
  - Most innovations in parallel architecture takes place in the communication assist (also called communication controller or node controller)



Previous Next

Design Issues

- Need to understand architectural components that affect software
  - Compiler , library, program
  - User/system interface and hw/ sw interface
  - How programming models efficiently talk to the communication architecture?
  - How to implement efficient primitives in the communication layer?
  - In a nutshell, what issues of a parallel machine will affect the performance of the parallel applications?
- Naming, Operations, Ordering, Replication, Communication cost

Naming

- How are the data in a program referenced?
  - In sequential programs a thread can access any variable in its virtual address space
  - In shared memory programs a thread can access any private or shared variable (same load/store model of sequential programs)
  - In message passing programs a thread can access local data directly
- Clearly, naming requires some support from hw and OS
  - Need to make sure that the accessed virtual address gets translated to the correct physical address

Operations

- What operations are supported to access data?
  - For sequential and shared memory models load/store are sufficient
  - For message passing models send/receive are needed to access remote data
  - For shared memory, hw (essentially the CA) needs to make sure that a load/store operation gets correctly translated to a message if the address is remote
  - For message passing, CA or the message layer needs to copy data from local memory and initiate send, or copy data from receive buffer to local memory

Previous Next

Ordering

- How are the accesses to the same data ordered?
  - For sequential model, it is the program order: true dependence order
  - For shared memory, within a thread it is the program order, across threads some "valid interleaving" of accesses as expected by the programmer and enforced by synchronization operations (locks, point-to-point synchronization through flags, global synchronization through barriers)
  - Ordering issues are very subtle and important in shared memory model (some microprocessor re-ordering tricks may easily violate correctness when used in shared memory context)
  - For message passing, ordering across threads is implied through point-to-point send/receive pairs (producer-consumer relationship) and mutual exclusion is inherent (no shared variable)

Replication

- How is the shared data locally replicated?
  - This is very important for reducing communication traffic
  - In microprocessors data is replicated in the cache to reduce memory accesses
  - In message passing, replication is explicit in the program and happens through receive (a private copy is created)
  - In shared memory a load brings in the data to the cache hierarchy so that subsequent accesses can be fast; this is totally hidden from the program and therefore the hardware must provide a layer that keeps track of the most recent copies of the data (this layer is central to the performance of shared memory multiprocessors and is called the cache coherence protocol)

Previous Next

Communication Cost

- Three major components of the communication architecture that affect performance
  - Latency: time to do an operation (e.g., load/store or send/ recv .)
  - Bandwidth: rate of performing an operation
  - Overhead or occupancy: how long is the communication layer occupied doing an operation
- Latency
  - Already a big problem for microprocessors
  - Even bigger problem for multiprocessors due to remote operations
  - Must optimize application or hardware to hide or lower latency (algorithmic optimizations or prefetching or overlapping computation with communication)
- Bandwidth
  - How many ops in unit time e.g. how many bytes transferred per second
  - Local BW is provided by heavily banked memory or faster and wider system bus
  - Communication BW has two components: 1. node-to-network BW (also called network link BW) measures how fast bytes can be pushed into the router from the CA, 2. within-network bandwidth: affected by scalability of the network and architecture of the switch or router
- Linear cost model: Transfer time = T0 + n/B where T0 is start-up overhead, n is number of bytes transferred and B is BW
  - Not sufficient since overlap of comp. and comm. is not considered; also does not count how the transfer is done (pipelined or not)

Previous Next

Communication Cost

- Better model:
  - Communication time for n bytes = Overhead + CA occupancy + Network latency + Size/BW + Contention
  - T(n) = Ov + Oc + L + n/B + Tc
  - Overhead and occupancy may be functions of n
  - Contention depends on the queuing delay at various components along the communication path e.g. waiting time at the communication assist or controller, waiting time at the router etc.
  - Overall communication cost = frequency of communication x (communication time – overlap with useful computation)
  - Frequency of communication depends on various factors such as how the program is written or the granularity of communication supported by the underlying hardware

ILP vs. TLP

- Microprocessors enhance performance of a sequential program by extracting parallelism from an instruction stream (called instruction-level parallelism)
- Multiprocessors enhance performance of an explicitly parallel program by running multiple threads in parallel (called thread-level parallelism)
- TLP provides parallelism at a much larger granularity compared to ILP
- In multiprocessors ILP and TLP work together
  - Within a thread ILP provides performance boost
  - Across threads TLP provides speedup over a sequential version of the parallel program

Previous Next