Chapter 2: Parallel Programming Platforms

Trevor Steil

January 22, 2017

1 2.3 Dichotomy of Parallel Computing Platforms

- SIMD
  - global control unit splits work among processors
  - all processors must perform exact same instructions at exact same time
    - think of idling in if/else statements
    - works well for structured problems
    - can use "activity mask" to tell whether different processors are supposed to take part in an individual calculation (leads to idling)
- MIMD
  - Each processing element can execute separate programs on different data at the same time
  - Is this parallelism used when introducing threads on a multi-core machine, say for matrix-vector multiplication?
    - If so, do we not have a global processing unit, i.e., a processing unit that splits work up among cores?
      - Possible answer: On the level of software that is running, a single processor core is distributing tasks. On the level of computer architecture, each core is still functioning independently. A better way of thinking of typical computer as MIMD machine is to think of multitasking on a computer.
      - Would this mean that SIMD machine may be better (faster, less resource-intensive) for matrix-vector multiplication because we are simulating in software the architecture of a SIMD machine
  - Requires more hardware than SIMD machine because each processing unit must also have control unit
  - Requires more memory than SIMD because SIMD machine can have a single copy of program in memory
  - Supports **SPMD (Single Program Multiple Data)** model
    - Multiple instances of the same program running on different data

- Shared-Address-Space Platforms
  - common data space accessible to all processors
  - multiprocessors are shared-address-space platforms supporting SPMD programming
  - can support local and global memory
  - UMA (Uniform Memory Access)
    - time for processor to access any data in memory is identical
    - Excludes cache in consideration (serial machine is not UMA machine otherwise)
  - NUMA (Non-Uniform Memory Access)
    - Processors can take longer to access certain memory locations
  - Reading global memory occurs as normal
  - Writing to global memory can cause issues with concurrent writes
    - How is cache of other processors changed when one processor writes value in global memory
  - Not the same as shared memory (in which all equal access to all memory, i.e. an UMA)
- Message-Passing Platforms
  - Used for passing messages between processing nodes with exclusive address space
  - Supports `send`, `receive`, `whoami` (returns ID of process), and `numprocs` (returns number of processes in ensemble)
  - Examples
    - Message Passing Interface (MPI)
    - Parallel Virtual Machine (PVM)
  - Can emulate message passing architecture on shared-address-space computer by partitioning memory and "sending" and "receiving" messages by writing to other nnode's memory and using appropriate synchronization primitives to communication partner when finished

2 Physical Organization of Parallel Platforms

- Parallel Random Access Machine (PRAM)
  - Ideal parallel machine consisting of p processors and unbounded global memory with uniform memory access
  - Divided into EREW, CREW, ERCW, and CRCW based on whether reads or writes are exclusive or concurrent
  - Concurrent writes can cause issues
    - Protocols for concurrent writes
    - Common

            - Concurrent write is allowed if all are trying to write the same value
        * Arbitrary
            - Arbitrary processor is allowed to write with rest failing
        * Priority
            - Processors are given predefined priorities and processor with highest priority succeeds with rest failing
        * Sum
            - Sum of all values processors are attempting to write is written
            - Sum can be replaced by any associative operation
    - Ideal EREW PRAM would have switches to only allow processor to access memory location of no other processors are already accessing it. This would require $\Theta(mp)$ switches where m is the number of words in global memory and p is the number of processors (impractical)

- Interconnection Networks for Parallel Computers
    - Static (aka Direct) Networks
        * Consist of point-to-point communication links between processing nodes
    - Dynamic (aka Indirect) Networks
        * Consist of links as well as switches, which dynamically connect processing nodes and memory banks
    - Switches
        * Consist of input and output ports
        * Minimum functionality is mapping of input ports to output ports
        * Total number of ports is degree
        * May support:
            - Internal buffering (when output port is busy)
            - Routing (to alleviate network congestion)
            - Multicast (same output on multiple ports)
        * Cost of switch is determined by cost of mapping hardware, the peripheral hardware, and packaging costs
            - What are these things?
    - Network Interfaces
        * Provide connectivity between nodes and network
        * Has ports to pipe data into and out of network
        * Responsibilities:
            - Packetizing data
            - Computing routing information
            - Buffering incoming and outgoing data for matching speeds of network and processing elements

        - Error checking
    - Conventionally hang off I/O buses. In highly parallel machines, hang off the memory bus (higher bandwidth).
        - What are the I/O and memory buses?

- Network Topologies
    - Bus-Based Networks
        - All nodes connected to shared transmission medium
        - Cost of additional node is that of a new bus interface ($O(p)$)
        - Low overhead communication cost compared with point-to-point message transfer
        - Distance between any two nodes is $O(1)$
        - Bottleneck associated with bus bandwidth
        - Scalable in cost, not scalable in performance
    - Crossbar Networks
        - Grid of switches connecting every processor to every memory bank
        - Scalable in performance, not scalable in cost
    - Multistage Networks
        - Lies between bus-based and crossbar to get network scalable in cost and performance
        - Omega Network
            - Contains $\log p$ stages, where p is number of processing nodes and memory banks
            - Each stage performs a perfect shuffle
            - Each stage uses $p/2$ switches, each of which is in pass-through or cross-over configuration
            - Has a total of $\frac{p}{2} \log p = \Theta(p \log p)$ switches (better than $\Theta(p^2)$ of crossbar)
            - Distinct accesses from and to distinct locations can block communication for each other
            - How does this work with differing numbers of processing nodes and memory banks?
    - Completely-Connected Network
        - Static counterpart of crossbar switching network
        - Are these networks connecting processing nodes to each other rather than processing nodes to memory banks like before?
    - Star-Connected Network
        - Similar to bus-based networks
        - Central processor is bottleneck
    - Linear Arrays, Meshes, and $k - d$ Meshes
        - Necessary communication for parallel computations often shaped like these (think of finite difference grid)

    * $k - d$ Mesh
        · Has d dimensions with k nodes in each
        · Linear array is one extreme with $d = 1$
        · Hypercube is other extreme with $d = \log p$ and $k = 2$
            1. Constructed recursively by connecting corresponding vertices of lower dimensional hypercubes (0-dimensional hypercube is a point)
            2. See page 41 for numbering scheme and associated minimum distance between nodes
  – Tree-Based Networks
    * Exactly one path between any two nodes
    * Examples: Linear Array, Star-Connected Network, Complete Binary Tree
    * In static tree network, all nodes are processing nodes
    * In dynamic tree network, intermediate levels are switches with all leaf nodes being processing nodes
    * Bottleneck created at higher levels of network (only in complete binary tree?)
        · Can be fixed by adding extra connections and switches at higher levels
        · Called fat tree

- Evaluating Static Networks
  – Diameter: maximum distance between any two nodes
  – Arc Connectivity: minimum number of arcs that must be removed to break into two disconnected networks
    * Is an arc the same as a link?
    * Measures multiplicity of paths between nodes
  – Bisection Width: minimum number of communication links removed to partition the network into two equal halves
    * How is bisection width of a star 1 (as in book)? What is the meaning of equal halves?
  – Channel Width: number of bits that can be communicated simultaneously over a link connecting two nodes (number of physical wires in each communication link)
  – Channel Rate: peak rate at which a single physical wire can deliver bits
  – Channel Bandwidth: peak rate data can be communicated over communication link (product of channel width and channel rate)
  – Bisection Bandwidth (aka Cross-Section Bandwidth): minimum volume of communication allowed between any two halves of the network (product of bisection width and channel bandwidth)
  – Cost
    * Number of communication links

        * Alternatively, bisection bandwidth gives lower bound on physical area (or volume) of network
            · How exactly does this work? How much "space" does a vertex take up?

- Evaluating Dynamic Interconnection Networks
    - Similar to metrics used for static networks, but we must also consider switches as nodes (there is an overhead associated to a switch)
    - Connectivity can be defined in terms of edges or nodes
        * Node connectivity is minimum number of (switch) nodes that must fail to fragment the network into two parts
        * Arc connectivity same as before
    - Bisection width
        * Consider partition of p processing nodes into equal parts (without restricting switch node partition)
        * Then select an induced partitioning of switches that minimizes the number of edges crossing the partition
        * See Figure 2.20
    - Cost is determined by link cost and switch cost (typically more expensive)
    - Numbers in Table 2.2 for Dynamic Tree seem off. Isn't dynamic tree made of processors just as leaf nodes, so diameter and cost are much higher (double?)? How is arc connectivity not 1?

- Cache Coherence in Multiprocessor Systems
    - Want to keep multiple copies of same data consistent with each other in a shared-address-space system (i.e. data in different caches)
    - Update Protocol
        * When data item is written, all copies in system are updated
        * If processor reads item but never uses it, data will need to be updated any time value is written elsewhere in program (latency and bandwidth costs)
        * Only updates on write to global memory, not write to cache
    - Invalidate Protocol
        * When data item is written, all other copies are invalidated
        * Only needs to invalidate item on first write (subsequent writes won't have any affect, as opposed to update protocol)
        * Invalidation happens whenever write is done in cache, not necessarily global memory
            · Will computers write to cache without updating in global memory
        * Most modern computers will us Invalidate Protocol (will be assumed in text)

- False Sharing
  - When one item is updated, all other items within cache-line will be updated as well
    - Can one item be updated at one processor, an update sent out, then another item at another processor being updated in the same cache-line be changed before update from other processor arrives, leading to the second item being returned to its original value?
  - With invalidate protocol, can lead to different processors repeatedly fetching data from other processors
    - Can two processors end up simultaneously invalidating each other's data?
  - With update protocol, all reads can be done locally, and only writes require cache-line to be updated
- Tradeoff between update protocol and invalidate protocol is tradeoff between communications (update) and idling (stalling in invalidates)
  - What leads to stalling in invalidate? Needing to fetch after cache-line is invalidated
- Implementation of Coherence Protocols
  - Snoopy Cache Systems
    - Uses bus or ring for broadcast interconnection network
    - Each processor monitors the bus for transactions and updates according to state diagram of its coherence protocol
    - Bus has finite bandwidth, which presents bottleneck when many coherence operations performed
    - All processors must broadcast all memory operations over bus, and all processors must snoop on all messages from all other processors. This is not scalable.
    - Why does $y = x + y$ use an invalidated copy of x stored at Processor 1 in Figure 2.23?
  - Directory Based Systems
    - Data in memory augmented with directory maintaining bitmap of cache blocks and processors where data is cached
    - Bitmap entries are called presence bits
    - Directory is contained in memory and can only be updated at a finite speed causing possible bottlenecks in program with large number of coherence actions (read/write of overlapping data blocks)
    - Memory required scales as $\Theta(mp)$ where m is number of memory blocks and p is the number of processors, which can be a bottleneck with many processors
      1. Can increase block size to fight this (i.e. reduce m), but this will cause larger costs of false sharing

3 Communication Costs in Parallel Machines

- Message Passing Costs in Paralllel Computers
  - Startup time (t_s): Time required to prepare message, execute routing algorithm, and establish an interface between node and router
  - Per-hop time (t_h): Time required for header of message to pass directly between adjacent nodes (aka node latency)
  - Per-word transfer time (t_w): $t_w = 1/r$ where r is the channel bandwidth (words/second)
  - Store-and-Forward Routing
    - Each intermediate node on path receives and stores entire message before forwarding to next node
    - Communication time for message of length m to traverse l links is given by
      $$t_{comm} = t_s + (mt_w + t_h)l.$$
      t_h is typically small, so can be approximated by
      $$t_{comm} = t_s + mt_w l.$$
  - Packet Routing
    - Message is broken into smaller pieces to reduce downtime waiting for entire message before forwarding
    - Reduced overhead due to recovering lost packets
    - Packets may take different paths (reducing congestion)
    - Better error correction capability How?
    - Increased overhead from each packet needing to carry routing, error correction, and sequencing information
  - Cut-Through Routing
    - Can place additional restrictions to reduce overhead of packet switching
      - Forcing all packets to take same path eliminates routing info with every packet
      - Forcing in-sequence delivery eliminates sequence info in packets
      - Associating erro info to message rather than packet reduces overhead of error detection and correction
      - Parallel machines tend to have low error rates, so a lean error detection mechanism can usually be used
    - Implementing above features results in cut-through routing
    - Message is split into fixed-size units called flits (smaller than packets because of reduced overhead info)
    - First, tracer is sent to establish connection
    - Then, flits are sent end-to-end along the same route
    - Intermediate nodes send flits on immediately after receiving them

· Reduces memory usage because whole message doesn't need to be stored as in Store-and-Forward

* Communication cost given by

$$t_{comm} = t_s + lt_h + t_w m$$

* Flit sizes vary for different networks and applications
  · For parallel programming paradigm where short messages are frequently passed (cache lines), latency is biggest concern
  · When longer variable-length messages are sent, bandwidth is biggest concer
* Deadlock: when no flits are able to move because buffers and links needed for continuing on route are being used for all flits (no flits can move)

– Simplified Cost Model for Communicating Messages

* For communicating message with cut-through routing, cost is given by

$$t_{comm} = t_s + lt_h + t_w m.$$

To minimize this cost:

· Communicate in bulk: t_s is often large, so amortize this (fixed) cost by sending larger messages
· Minimize volume of data (m)
· Minimize distance of data transfer (l)

1. Often impractical because programmer doesn't have control over mapping of tasks to physical processors and networks will often use randomized (two-step) routing by first sending to a random node then to destination, and t_h and l are often small

· Leads to simplified communication cost model

$$t_{comm} = t_s + t_w m$$

1. Communication time is constant between any pair of nodes (implies model is working on a completely-connected network)
2. Can design algorithms with this cost model in mind to give an architecture-independent algorithm and port to any machine
3. Requires uncongested network to be valid

– Communication Costs in Shared-Address-Space Machines

* Harder to determine than for message passing:
  · Programmer has limited control, and local and remote access times can be vastly different in distributed memory system
  · Finite cache size can result in cache thrashing in which values in cache are overwritten and retrieved multiple times in course of computation. This problem can be present in serial programs as well, but each miss is more costly in multiprocessor system because of coherence operations and interprocessor communications involved

- Hard to quantify overhead associated to invalidate and update operations. Overhead depends on how frequently data is invalidated and needs to be fetched, or how many copies of data are being used when an update occurs.
- Hard to model spatial locality. Access times may be very different based on size of cache lines. Programmer has no control other than to permute data structures.
- Prefetching can reduce overhead, but it is dependent on compiler and resource availability
- False sharing can add overhead when data is not being operated on by different processors
- Contention is a major overhead (What is contention?)

* Cost model with these considerations becomes too cumbersome to use and too specific to individual machines to be generally applicable

4 Routing Mechanisms for Interconnection Networks

- Routing Mechanism
  - Determines path a message takes through network
  - Depends on source and destination, possibly state of network
  - Returns one or more paths through network
  - Minimal Routing Mechanism: always uses one of the shortest path between two points (each node brings message closer to destination)
  - Non-Minimal Routing Mechanism: may use a longer route to avoid congestion
  - Deterministic Routing: gives a unique based on just source and destination
    * Example: dimension-ordered routing - message traverses dimensions (in mesh, hypercube, etc.) one at a time
  - Adaptive Routing: Uses state of network to determine path to take (routes around congestion)
  - Book will assume deterministic and minimal message routing
  - Is there any relation between dynamic and static networks and what type of routing can be done on them?

5 Impact of Process-Processor Mapping and Mapping Techniques

- Communication among processors can be given same structure as communication between processes in program, leading to no congestion, but if processes aren't mapped to right processors, this advantage may be lost

and congestion can occur (programmer often doesn't have control of these mappings)