MIMD Parallel Architectures

Slides courtesy of David Culler
CS 258, UC Berkeley

Major MIMD Styles

- Centralized shared memory ("Uniform Memory Access" time or "Shared Memory Processor")
- Decentralized memory (memory module with CPU)
  - Advantage: et more memory bandwidth, lower memory latency
  - Drawback: Longer communication latency
  - Drawback: Software model more complex
Decentralized Memory versions

- Shared Memory with "Non Uniform Memory Access" time (NUMA)
- Message passing "multicomputer" with separate address space per processor
  - Can invoke software with Remote Procedure Call (RPC)
  - Often via library, such as MPI: Message Passing Interface
  - Also called "Synchronous communication" since communication causes synchronization between 2 processes

Shared Memory Processors (SMP)

- Memory capacity increased by adding modules
- I/O by controllers and devices
- Add processors for processing!
Shared Memory Processors

- Any processor can directly reference any memory location
- Any I/O controller - any memory

- Operating system can run on any processor, or all.
  - OS uses shared memory to coordinate
- Communication occurs implicitly as result of loads and stores

SMP Historical Development

- “Mainframe” approach
  - Motivated by multiprogramming
  - Extends crossbar used for Mem and I/O
  - Processor cost-limited => crossbar
  - Bandwidth scales with $p$
  - High incremental cost
    - use multistage instead

- “Minicomputer” approach
  - Almost all microprocessor systems have bus
  - Motivated by multiprogramming, TP
  - Used heavily for parallel computing
  - Called symmetric multiprocessor (SMP)
  - Latency larger than for uniprocessor
  - Bus is bandwidth bottleneck
    - caching is key: coherence problem
  - Low incremental cost
**Shared Virtual Address Space**

- **Process** = address space plus thread of control
- **Virtual-to-physical mapping** can be established so that processes shared portions of address space.
  - User-kernel or multiple processes
- **Multiple threads of control** on one address space.
  - Popular approach to structuring OS’s
  - Now standard application capability (ex: POSIX threads)
- **Writes to shared address visible to other threads**
  - Natural extension of uniprocessors model
  - Conventional memory operations for communication
  - Special atomic operations for synchronization
    » also load/stores

**Structured Shared Address Space**

- **Add hoc parallelism used in system code**
- **Most parallel applications** have structured SAS
- **Same program on each processor**
  - Shared variable X means the same thing to each thread
Engineering: Intel Pentium Pro Quad

- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Engineering: SUN Enterprise

- Proc + mem card - I/O card
  - 16 cards of either type
  - All memory accessed over bus, so symmetric
  - Higher bandwidth, higher latency bus
Evolution from SMP to Distributed Memory

- Problem is interconnect: cost (crossbar) or bandwidth (bus)
- Dance-hall: bandwidth still scalable, but lower cost than crossbar
  » latencies to memory uniform, but uniformly large
- Distributed memory or non-uniform memory access (NUMA)
  » Construct shared address space out of simple message transactions across a general-purpose network (e.g. request, read-response)
- Caching shared (particularly nonlocal) data?

Engineering: Cray T3E

- Scale up to 1024 processors, 480MB/s links
- Memory controller generates request message for non-local references
- No hardware mechanism for coherence
  » SGI Origin etc. provide this
Distributed Memory/Message Passing Architectures (NUMA)

- Complete computer as building block, including I/O
  - Communication via explicit I/O operations

- Programming model
  - direct access only to private address space (local memory),
  - communication via explicit messages (send/receive)

- High-level block diagram
  - Communication integration?
    » Mem, I/O, LAN, Cluster
  - Easier to build and scale than SAS

- Programming model more removed from basic hardware operations
  - Library or OS intervention

Message-Passing Abstraction

- Send specifies buffer to be transmitted and receiving process
- Recv specifies sending process and application storage to receive into
- Memory to memory copy, but need to name processes
- Optional tag on send and matching rule on receive
- User process names local data and entities in process/tag space too
- In simplest form, the send/recv match achieves pairwise synch event
  » Other variants too
- Many overheads: copying, buffer management, protection
Evolution of Message-Passing Machines

• Early machines: FIFO on each link
  – HW close to prog. Model;
  – synchronous ops
  – topology central (hypercube algorithms)

Diminishing Role of Topology

• Shift to general links
  – DMA, enabling non-blocking ops
    » Buffered by system at destination until recv
  – Store&forward routing

• Diminishing role of topology
  – Any-to-any pipelined routing
  – node-network interface dominates communication time
    \[ H \times (T_0 + n/B) \]
    \[ \text{vs} \]
    \[ T_0 + H\Lambda + n/B \]
  – Simplifies programming
  – Allows richer design space
    » grids vs hypercubes
Example Intel Paragon

Sandia’s Intel Paragon XPS-based Supercomputer

2D grid network with processing node attached to every switch

8 bits, 175 MHz, bidirectional

Building on the mainstream: IBM SP-2

• Made out of essentially complete RS6000 workstations

• Network interface integrated in I/O bus (bw limited by I/O bus)
Berkeley NOW and Beowulf Clusters

- Off the shelf processors
- Intelligent network interface
  - proc + mem
- Myrinet Network
  - 160 MB/s per link
  - 300 ns per hop

Toward Architectural Convergence

- Evolution and role of software have blurred boundary
  - Send/recv supported on shared addr. space machines via buffers
  - Can construct global address space on distributed memory machines
  - Page-based (or finer-grained) shared virtual memory
- Hardware organization converging too
  - Tighter NI integration even for MP (low-latency, high-bandwidth)
  - Hardware SAS passes messages
- Even clusters of workstations/SMPs are parallel systems
  - Emergence of fast system area networks (SAN)
- Programming models distinct, but organizations converging
  - Nodes connected by general network and communication assists
  - Implementations also converging, at least in high-end machines
Convergence: Generic Parallel Architecture

- Node: processor(s), memory system, plus communication assist
  - Network interface and communication controller
- Scalable network
- Convergence allows lots of innovation, within framework
  - Integration of assist with node, what operations, how efficiently...

Towards architectural convergence

- Systolic Arrays
- Dataflow
- Generic Architecture
- SIMD
- Message Passing
- Shared Memory
Some important non-MIMD architectures

• Data parallel
• Dataflow
• Systolic

Data Parallel Systems

• Programming model
  – Operations performed in parallel on each element of data structure
  – Logically single thread of control, performs sequential or parallel steps
  – Conceptually, a processor associated with each data element

• Architectural model
  – Array of many simple, cheap processors with little memory each
    » Processors don’t sequence through instructions
  – Attached to a control processor that issues instructions
  – Specialized and general communication, cheap global synchronization

• Original motivations
  – Matches simple differential equation solvers
  – Centralize high cost of instruction fetch/sequencing
Application of Data Parallelism

- Each PE contains an employee record with his/her salary
  
  If salary > 100K then
  salary = salary * 1.05
  
  else
  salary = salary * 1.10

- Logically, the whole operation is a single step
- Some processors enabled for arithmetic operation, others disabled

• Other examples:
  - Finite differences, linear algebra, ...
  - Document searching, graphics, image processing, ...

• Some recent machines:
  - Thinking Machines CM-1, CM-2 (and CM-5)
  - Maspar MP-1 and MP-2,

Connection Machine

(Tucker, IEEE Computer, Aug. 1988)
Evolution and Convergence

- SIMD Popular when cost savings of centralized sequencer high
  - 60s when CPU was a cabinet
  - Replaced by vectors in mid-70s
    - More flexible w.r.t. memory layout and easier to manage
  - Revived in mid-80s when 32-bit datapath slices just fit on chip
- Simple, regular applications have good locality

- Programming model converges with SPMD (single program multiple data)
  - need fast global synchronization
  - Structured global address space, implemented with either SAS or MP

CM-5

- Repackaged SparcStation
  - 4 per board
- Fat-Tree network
- Control network for global synchronization
**Dataflow Architectures**

- Represent computation as a graph of essential dependences
  - Logical processor at each node, activated by availability of operands
  - Message (tokens) carrying tag of next instruction sent to next processor
  - Tag compared with others in matching store; match fires execution

**Evolution and Convergence**

- **Key characteristics**
  - Ability to name operations, synchronization, dynamic scheduling
- **Problems**
  - Operations have locality across them, useful to group together
  - Handling complex data structures like arrays
  - Complexity of matching store and memory units
  - Expose too much parallelism (?)
- **Converged to use conventional processors and memory**
  - Support for large, dynamic set of threads to map to processors
  - Typically shared address space as well
  - But separation of progr. model from hardware (like data-parallel)
- **Lasting contributions:**
  - Integration of communication with thread (handler) generation
  - Tightly integrated communication and fine-grained synchronization
  - Remained useful concept for software (compilers etc.)
Systolic Architectures

• VLSI enables inexpensive special-purpose chips
  – Represent algorithms directly by chips connected in regular pattern
  – Replace single processor with array of regular processing elements
  – Orchestrate data flow for high throughput with less memory access

• Different from pipelining
  – Nonlinear array structure, multidirection data flow, each PE may have (small) local instruction and data memory
• SIMD? : each PE may do something different

Example: Systolic array for 1-D convolution
\[ y(i) = w_1 x(i) + w_2 x(i + 1) + w_3 x(i + 2) + w_4 x(i + 3) \]

Systolic Arrays (contd.)

Example: Systolic array for 1-D convolution

- Practical realizations (e.g. iWARP) use quite general processors
  - Enable variety of algorithms on same hardware
- But dedicated interconnect channels
  - Data transfer directly from register to register across channel
- Specialized, and same problems as SIMD
  - General purpose systems work well for same algorithms (locality etc.)