Scalable Multiprocessors

[From Chapter 7 of Culler, Singh, Gupta]

Previous Next

Agenda

- Basics of scalability
- Programming models
- Physical DMA
- User-level networking
- Dedicated message processing
- Shared physical address
- Cluster of workstations (COWs) and Network of workstations (NOWs)
- Scaling parallel software
    - Scalable synchronization

Basics of scalability

- Main problem is the communication medium
    - Busses don't scale
    - Need more wires that are not always shared by all
    - Replace bus by a network of switches
- Distributed memory multiprocessors
    - Each node has its own local memory
    - To access remote memory, node sends a point-to-point message to the destination
    - How to support efficient messaging?
    - Main goal is to reduce the ratio of remote memory latency to local memory latency
    - In shared memory, how to support coherence efficiently?

Bandwidth scaling

- Need a large number of independent paths between two nodes
    - Makes it possible to support a large number of concurrent transactions
    - They get initiated independently (as opposed to a single centralized bus arbiter)
- Local accesses should be higher bandwidth
- Since communication takes place via point-to-point messages, only the routers or switches along the path are involved
    - No global visibility of messages (unlike a bus)
    - Must send separate messages to make sure that global visibility is guaranteed when necessary (e.g., invalidations)

Latency scaling

- End-to-end latency of a message involves three parts (log model)
    - Overhead (o): time to initiate a message and to terminate a message (at sender and receiver respectively); normally involves kernel overhead in message passing and the coherence overhead in shared memory
    - Node-to-network time or gap (g): number of bytes/link bandwidth where this is the bandwidth offered by the router to/from network (how fast you can push packets into the network or pull packets from the network); normally the bandwidth between network interface (NI) and the router is at least as big and hence is not a bottleneck
    - Routing time or hop time (L): determined by topology, routing algorithm, and router circuitry (e.g., arbitration, number of ports etc.)

  - Importance: $L < g < o$ for most scientific applications

Cost scaling

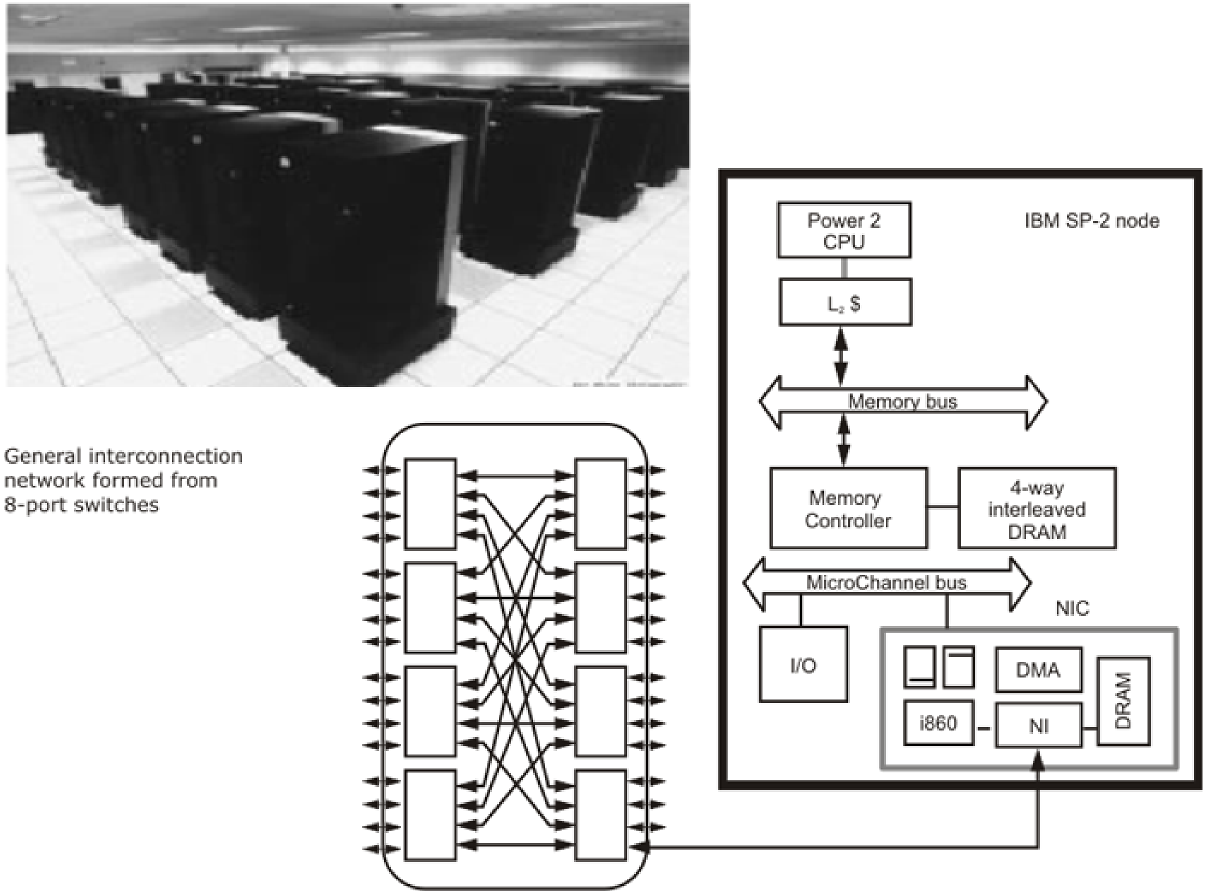
- Cost of anything has two components
  - Fixed cost
  - Incremental cost for adding something more (in our case more nodes)
- Bus-based SMPs have too much of fixed cost
  - Scaling involves adding a commodity processor and possibly more DRAM
  - Need to have more modular cost scaling i.e. don’t want to pay so much even for a small scale machine
- Costup = cost of P nodes / cost of single node
- Parallel computing on a machine is cost-effective if speedup > costup on average for target applications

Previous Next

Physical scaling

- Integration at various levels
    - Chip-level integration to keep wires short: nCUBE/2 (1990) puts the processor, router, MMU, DRAM interface on a single chip
    - Board-level integration: Thinking machine CM-5 uses the core of a Sun SparcStation 1 and other peripherals on a single board to realize a node
    - System-level integration: IBM SP1 and SP2 exploit commodity RS6000 workstations and connect them through an external communication assist / network interface card and a router

IBM SP-2



Programming model

- Shared address space
    - Communication initiated by load/store instructions
    - Requires local or remote memory access before the data can be sent
    - Request/response protocol initiated by receiver
- Message passing
    - A one way communication: normally initiated by the sender
    - Number of sends can be buffered before a matching receive shows up

Synchronous vs. asynchronous sends

- Active messages
    - Restricted form of remote procedure call

Previous Next

Common challenges

- Input buffer overflow
    - Reserve space per source
    - Refuse input when full
    - Let the network backup naturally: tree saturation
    - Deadlock-free networks
    - Traffic not bound to hot-spot nodes may get severely affected
    - Keep a reserved NACK channel
    - May drop packets depending on the network protocol
- Fetch deadlock
    - Nodes must continue to serve new messages while waiting for queue space so that it can put new requests
    - Separate request and response virtual networks (essentially disjoint set of queues in each port of the router) or have enough buffer space to never run into this problem (may be too expensive)

Spectrum of designs

- In increasing order of hardware support and probably performance and cost
    - Physical bit stream, physical DMA (nCUBE, iPSC)
    - User-level network port (CM-5, MIT *T)
    - User-level handler (MIT J machine, Monsoon)
    - Remote virtual address (Intel Paragon, Meiko CS-2)
    - Global physical address (Cray T3D, T3E)
    - Cache-coherent shared memory (SGI Origin, Alpha GS320, Sun S3.mp)

Physical DMA

- A reserved area in physical memory is used for sending and receiving messages
    - After setting up the memory region the processor takes a trap to the kernel
    - The interrupt handler typically copies the data into kernel area so that it can be manipulated
    - Finally, kernel instructs the DMA device to push the message into the network via the physical address of the message (typically called DMA channel address)
    - At the destination the DMA device will deposit the message in a predefined physical memory area and generates an interrupt for the processor
    - The interrupt handler now can copy the message into kernel area and inspect and parse the message to take appropriate actions (this is called blind deposit)

nCUBE/2

- Independent DMA channels per link direction
- Segmented messages (first segment can be inspected to decide what to do with the rest)
- Active messages: 13 µs outbound, 15 µs inbound
- Dimension-order routing on hypercube

User-level ports

- Network ports and status registers are memory-mapped in user address space

- User program can initiate a transaction by composing the message and writing to the status registers
- Communication assist does the protection check and pushes the message into the physical medium
- A message at the destination can sit in the input queue until the user program pops it off
- A system message generates an interrupt through the destination assist, but user messages do not require OS intervention
- Problem with context switch: messages are now really part of the process state; need to save and restore them
- Thinking machine CM-5 has outbound message latency of 1.5 µs and inbound 1.6 µs

Previous Next

User-level handling

- Instead of mapping the network ports to user memory, make them processor registers
  - Even faster messaging
  - Communication assist now looks really like a functional unit inside the processor (just like a FPU)
  - Send and receive are now register to register transfers
  - iWARP from CMU and Intel, *T from MIT and Motorola, J machine from MIT
  - iWARP binds two processor registers to the heads of the network input and output ports; the processor accesses the message word-by-word as it streams in
  - *T extended Motorola 88110 RISC core to include a network function unit containing dedicated sets of input and output registers; a message is spread over a set of such registers and a special instruction initiates the transfer

Message co-processor

- Nodes equipped with a dedicated message processor or communication processor (CP)
  - Two possible organizations: main processor and CP sit on a shared memory bus along with the main memory and NI; otherwise the CP may be integrated into the NI
  - The main processor and CP talk to each other via the normal cache coherence protocol i.e. while sending a message the main processor fills a shared buffer and sets a flag and while receiving a message CP does the same thing
  - Possible inefficiency due to invalidation-based coherence protocol (Update protocol would be worse)
  - CP may need to handle a lot of concurrent transactions e.g., from main processor and from network: a single dispatch loop serializes the processing (multi-threaded CP?)

Intel Paragon

- One i860XP processor per SMP node (MESI) is dedicated as CP
- One receive and one transmit DMA engine for transferring data from/to shared memory to/from NI transmit/receive queue (each 2 KB)
- While sending a large message the NI queue may become full and the network may not drain that fast
  - To avoid deadlock the transmit DMA is stalled by hardware flow control and the bus is relinquished
- Takes about 10 µs to send a small message (about two cache lines) from the register file of source to the register file of destination

Meiko CS-2

- The CP is tightly integrated with the NI and has separate concurrent units
  - The command unit sits directly on the shared bus and is responsible for fielding processor requests
  - The processor executes an atomic swap between one register and a fixed memory location which is mapped to the head of the CP input queue
  - The command contains a command type and a VA
  - Depending on the command type the command processor can invoke the DMA processor (may need assistance from VA to PA unit), an event processor (to wake up

- some thread on the main processor), or a thread processor to construct and issue a message
- The input processor fields new messages from the network and may invoke the reply processor, or any of the above three units

Shared physical addr.

- Memory controller on each node accepts PAs from the system bus
    - The processor initially issues a VA
    - The TLB provides the translation and the upper few bits of PA represent the home node for this address (determined when the mapping is established for the first time)
    - If the address is local i.e. requester is the home node, the memory controller returns data just as in uniprocessor
    - If address is remote the memory controller instructs the communication assist (essentially the NI) to generate a remote memory request
    - In the remote home the CA issues a request to the memory controller to read memory and eventually data is returned to the requester

Previous Next

Caching shared data?

- All transactions are no longer visible to all
    - Whether a page should be cached or not is normally part of the VA to PA translation
    - For example, in some graphics co-processors all operations must be through uncached writes or storing command/data to memory-mapped control registers is also uncached
    - Private memory lines can be cached without any problem and does not depend on if the line is local or remote
    - Caching shared lines introduce coherence issues

COWs and NOWs

- Historically, used to build a multi-programmed multi-user system
    - Connect a number of PCs or workstations over a cheap commodity network and schedule independent jobs on machines depending on the load
- Increasingly, these clusters are being used as parallel machines
    - One major reason is the availability of message passing libraries to express parallelism over commodity LAN or WAN
    - Also, technology breakthrough in terms of high-speed interconnects (ServerNet, Myrinet, Infiniband, PCI Express AS, etc.)
- Varying specialized support in CA
    - Conventional TCP/IP stack imposes an enormous overhead to move even a small amount of data (often more than common Ethernet): network processor architecture has been a hot research topic
    - Active messages allow user-level communication
    - Reflective memory allows writes to special regions of memory to appear as writes into regions on remote processors
    - Virtual interface architecture (VIA): each process has a communication end point consisting of a send queue, receive queue, and status; a process can deposit a message in its send queue with a dest. id so that it actually gets into the receive queue of target process
    - The CA hardware normally plugs on to I/O bus as opposed memory bus (fast PCI bus supports coherence); could be on the graphics bus also

Scalable synchronization

- In message-passing a send/receive pair provides point-to-point synchronization
- Handle all-to-all synchronization via tree barrier
- Also, all-to-all communication must be properly staggered to avoid hot-spots in the system
    - Classical example of matrix transpose
- Scalable locks such as ticket or array should be used
- Any problem with array locks?
    - Array locations now may not be local: invalidation causes remote misses

Distributed queue locks

- Goodman, Vernon, Woest (1989)
    - Logically arrange the array as a linked list
    - Allocate a new node (must be atomic) when a processor enters acquire

- The node is allocated on the local memory of the contending processor
- A tail pointer is always maintained

Previous Next