# Lecture 3: Processor Arrays, Multiprocessors and Multicomputers

Lecture Outline

(Reading: Chapter 3 of textbook)

1. Processor organizations
2. Popular processor organizations
3. Processor arrays
4. Multiprocessors and UMA multiprocessors
5. Multicomputers
6. Flynn's Taxonomy of computer architectures
7. Superlinear speedup, scaled speedup, and parallelizability

1. Processor organizations

(1) Processor organizations: methods of connecting processors in a parallel computer.
Can be represented by a graph.

(2) Criteria:

- **Diameter**: the longest distance between two processors in a network. Low diameter is better.
- **Bisection width**: the bisection width of a network is the minimum number of edges that have to be removed in order to divide the network into halves. High BW is better.
- **Number of edges per node**. Better be a constant, independent of the network size.
- **Maximum edge length**. The nodes and edges will have to be laid out in three-dimensional space. The MEL better be a constant.

2. Popular processor organizations

(1) **Mesh networks** (Fig.3-1,p.54)
A $q$-dimension **mesh network** has $k^q$ nodes arranged into a $q$-dimensional lattice, where $k \geq 1$; communication is allowed only between neighboring nodes; hence each interior node communicate with $2q$ other processors (three variaties: with no wrap-around connections; with wrap-around connections between processors in the same row and column; with toroidal wrap-around connections).

- Diameter: The diameter of a $q$-dimensonal mesh (without wrap-around structure with $k^q$ nodes is $q(k-1)$.

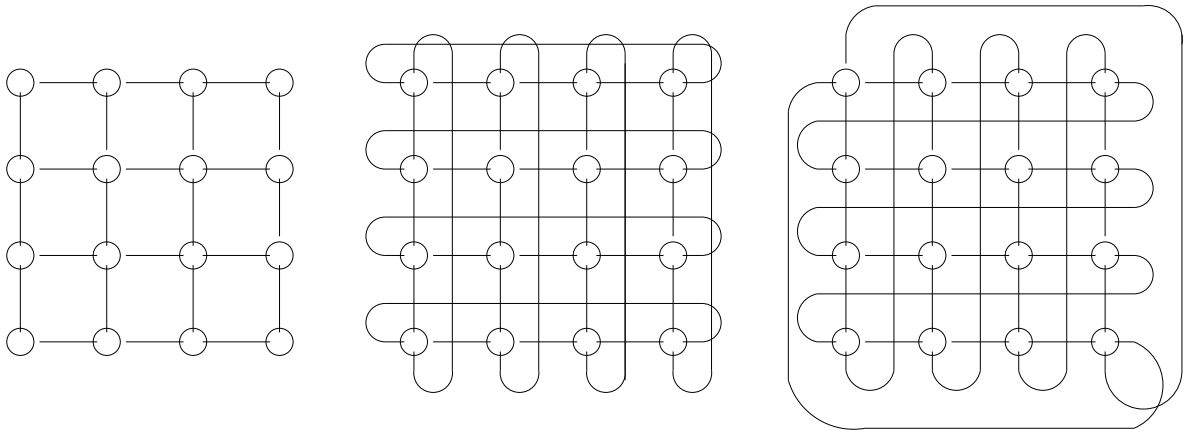


Figure 30: Two-dimensional meshes. (a) Mesh with no wrap-around connections. (b) Mesh with wrap-around connections between processors in same row or column. (c) Mesh with wrap-around connections between processors in adjacent rows or columns (Fig.3-1,p.54)

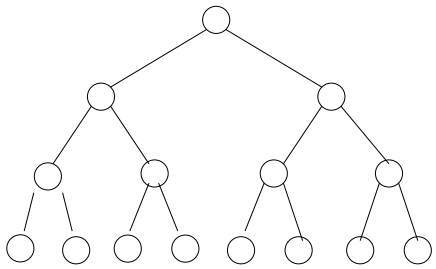

Figure 31: Binary tree network of size 15 and depth 3 (Fig.3-2,p.54)

- BSW: for size $k^q$, is $k^{q-1}$.
- Maximum number of links per node: $2q$.
- Length of longest edges: for $q > 1$, is a constant, independent of number of nodes.
- **Usages:** Sorting, matrix multiplication, solving second-order differential equations.
- **Disadvantages:** Preventing $O(\log^k n)$ parallel algorithms.
  Sorting $n$ elements on a 2-dimensional mesh needs time $\Omega(\sqrt{n})$ (compare with $\log^2 n$ complexity achievable by Bitonic merge sorting).

(2) **Binary tree networks** (Fig.3-2,p.54)
A **binary tree network** of size $n$ consists of $n = 2^k - 1$ nodes arranged in a

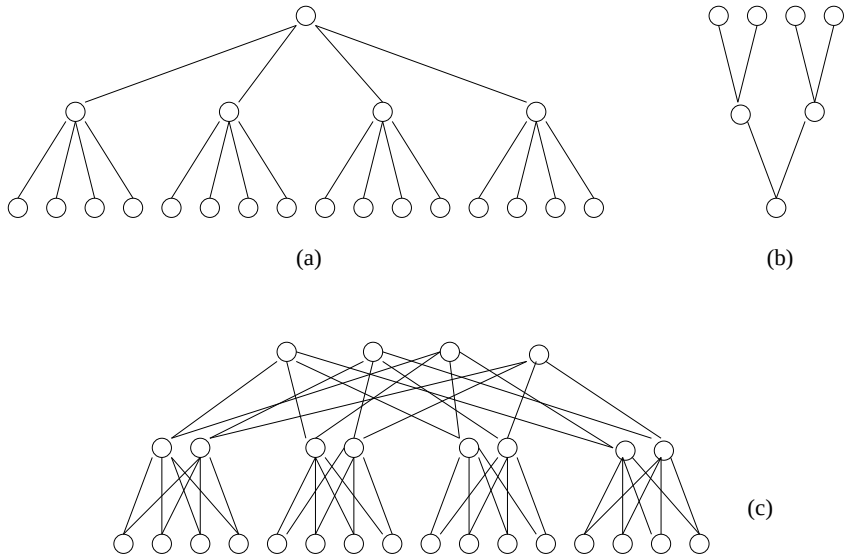


Figure 32: Hypertree network of degree 4 and depth 2. (a) Front view. (b) Side view. (c) Complete network (Fig.3-3,p.55)

complete binary tree of depth $k-1$.

- Diameters: $2(k-1)$ (low).
- BSW: 1 (very poor).
- Maximum number of links per node: <= 3.
- Length of longest edges: not a constant. It's impossible to maintain constant maximum edge length in a 3-dimension space.

(3) **Hypertree networks** (Fig.3-3,p.55)
A **hypertree network** consists of two parts: the downward part and upward part. Both parts are of the form of a complete tree. A 4-ary hypertree with depth $d$ has $4^d$ leaves and $2^d(2^{d+1}-1)$ total nodes.

- Diameter: $2d$.
- BSW: $2^{d+1}$.
- Maximum num of edges per node: <= 6.
- Length of longest edges: is a function of network size.

(4) **Pyramid networks** (Fig.3-4,p.56)
A **pyramid network** of size $p$ is a complete 4-ary rooted tree of height $\log_4 p$ augmented with additional links so that the processors in every tree level form a 2-dimensional mesh network. At the base is a 2-dimensional mesh containing

$p = k^2$ nodes. The total number of processors in a pyramid of size $p$ is $\frac{4}{3}p - \frac{1}{3}$. Levels are numbered $0, 1, \cdots, \log_4 p$ from bottom to the top of the pyramid. Every interior node is connected to 9 other nodes: 1 parent, 4 mesh neighbors, 4 children.

- Diameter: for a pyramid of size $k^2$, is $2 \log k$. Reduced over the 2-D mesh.
- BSW: (not increased much) $2k$ for a pyramid of size $k^2$
- Maximum number of links per node: <= 9, independent of network size.
- Length of longest edges: increasing function of the net size.

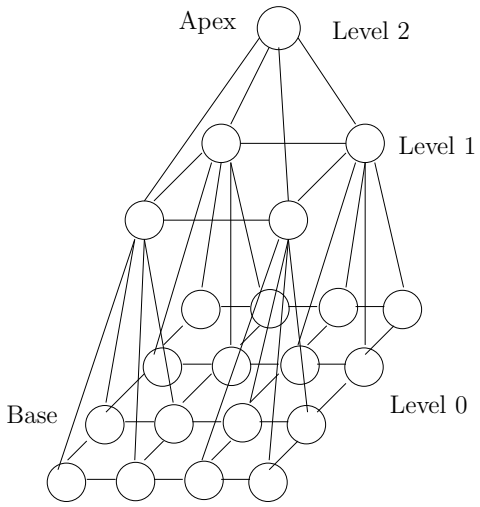


Figure 33: A pyramid network of size 16 (Fig.3-4,p.56)

(5) **Butterfly networks** (Fig.3-5,p.57)
A **butterfly network** consists of $(k+1)2^k$ nodes divided into $k+1$ rows (or called ranks) of $n = 2^k$ nodes each.

Let $(i, j)$ be the $j^{th}$ node on the $i^{th}$ rank, where $0 \leq i \leq k$, $0 \leq j \leq 2^k - 1$. Node $(i, j)$ is connected to two nodes on rank $i-1$: node $(i-1, j)$ and node $(i-1, m)$, where $m$ is the binary integer obtained by inverting the $i^{th}$ most significant bit of the binary representation of $j$. So if $j = a_k a_{k-1} \cdots a_{k-i+2} 1 a_{k-i} \cdots a_1$, then $m = a_k a_{k-1} \cdots a_{k-i+2} 0 a_{k-i} \cdots a_1$.
If node $(i, j)$ is connected to node $(i-1, m)$, then node $(i, m)$ is connected to node $(i-1, j)$. (Rank 0 and rank $k$ are often identified). Therefore each node is connected to exactly 4 other nodes, forming a *butterfly*.

- Diameter: $2k$ for a net of size $(k+1)2^k$.
- BSW: $2^k$.

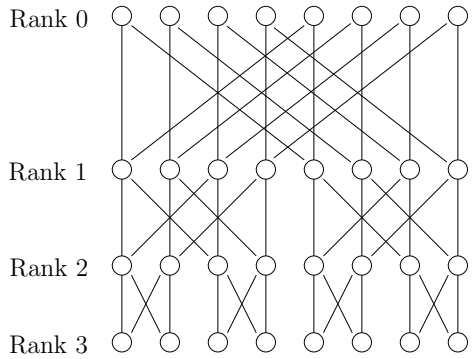


Figure 34: Butterfly network with 32 nodes (Fig.3-5,p.57)

- Maximum number of links per node: <= 4.
- Length of longest edges: increases as a function of the net size.

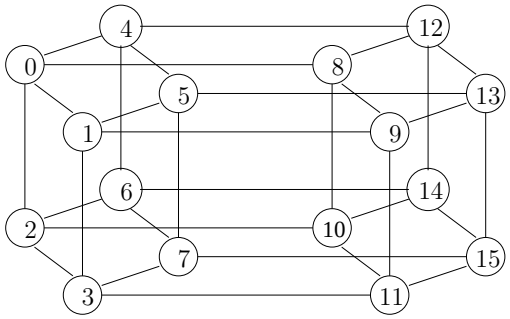


Figure 35: A four-dimensional (16 node) hypercube (Fig.3-6,p.58)

(6) **Hypercube (cube-connected) networks** (Fig.3-6,p.58)
A **hypercube network** is a butterfly with its columns collapsed into single nodes. Formally a hypercube network consists of $n = 2^k$ nodes forming a $k$-dimensional hypercube. The $n$ nodes are labeled $0, 1, \cdots, 2^k - 1$. Two nodes are adjacent if the binary representations of their labels differ in exact one bit. So each node is connected to exactly $k$ other nodes.

- Diameter: $k = \log n$ for a net of size $2^k$.
- BSW: $2^{k-1}$.
- Maximum number of links per node: $k$.
- Length of longest edges: increases as a function of the net size.

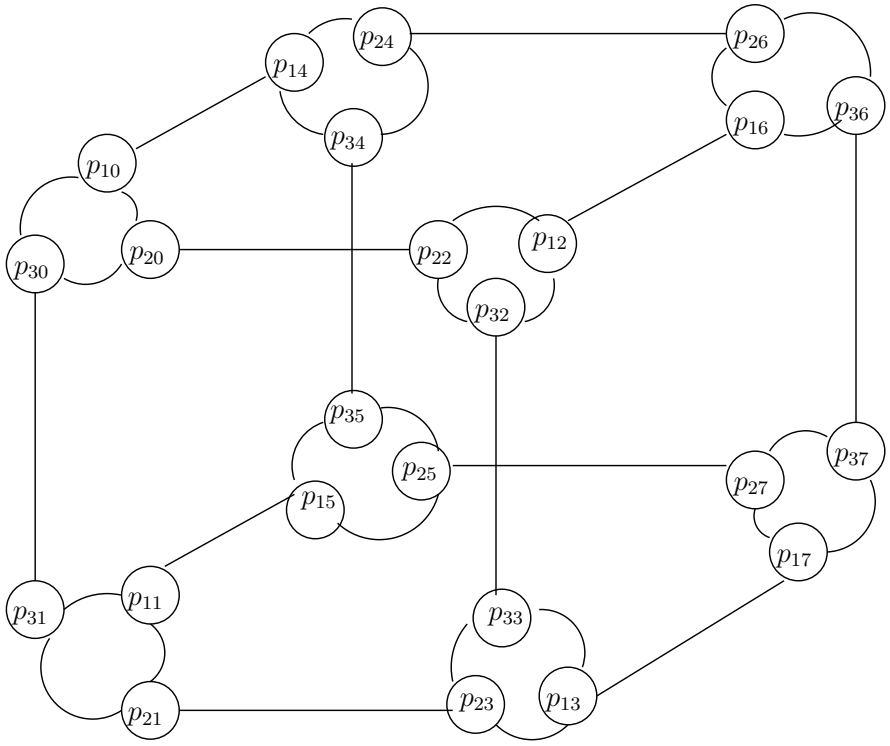


Figure 36: Cube-connected cycles network with 24 nodes. The first subscript of each node denotes the rank; the second subscript of each node denotes column (Fig.3-7,p.59)

(7) **Cube-connected cycles networks** (Fig.3-7,p.59)
A CCC network with $k2^k$ nodes can be thought as a $k$-dimensional hypercube whose $2^k$ "vertices" are cycles of $k$ nodes formed by columns of a butterfly network, whose ranks 0 and $k$ have been identified. For each dimension, every cycle has a node connected to a node in the neighboring cycle in that dimension. Node $(i, j)$ is connected to node $(i, m)$ iff $m$ is the result of inverting the $i^{th}$ most significant bit of the binary representation of $j$. (Different from butterfly network): if node $(i, j)$ is connected to node $(i-1, m)$ in the butterfly network, where $j \neq m$, then node $(i, j)$ is connected to node $(i, m)$ in the cube-connected cycles network.

- Diameter: $2k$ for a net of size $k2^k$. A disadvantage compared with the hypercube.
- BSW: $2^{k-1}$.
- Maximum number of links per node: 3, a constant. An advantage over the hypercube.
- Length of longest edges: increases as a function of the net size.

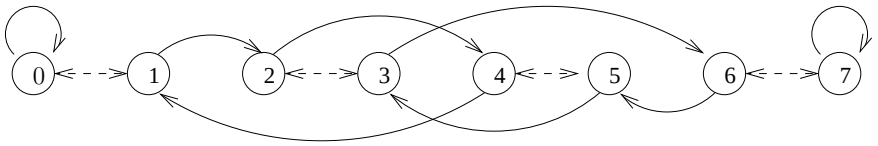


Figure 37: Shuffle-exchange network with eight nodes. Solid arrows denote shuffle connections. Dashed arrows denote exchange connections. (Fig.3-8,p.59)

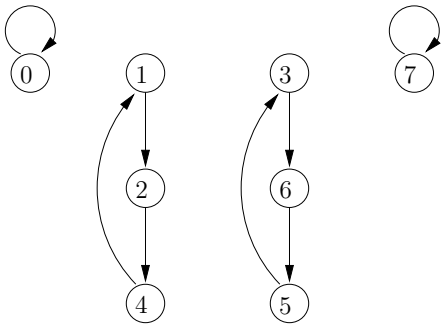


Figure 38: Necklaces of the shuffle-exchange network with eight nodes (Fig.3-9,p.60)

(8) **Shuffle-exchange networks** (Fig.3-8,p.59)
A **shuffle-exchange network** with $n$ processors, (where $n = 2^k$, for some $k > 0$), has $n$ nodes numbered $0, 1, \cdots, n-1$. There are two types of connections: **perfect shuffle** and **exchange**. Exchange connections link pairs of nodes whose number differ in their least significant bit. The perfect shuffle connections link nodes $i$ with nodes $2i$ modulo $(n-1)$, with the exception that node $n-1$ is connected to itself.

- Diameter: $2k-1$ for a net of size $2^k$.
- BSW: $2^{k-1}/k$.
- Maximum number of links per node: two outgoing and two incoming links for every node.
- Length of longest edges: increases as a function of the net size.
- Why called shuffle-exchange network? Let 0,1,2,3,4,5,6,7 be divided into two equal halves: 0,1,2,3 and 4,5,6,7. A perfect shuffle of these two sequences will produce 0,4,1,5,2,6,3,7.
- Let $n = 2^k$ and $a_k a_{k-1} \cdots a_1$ be the binary address of a node. A data item at this node will be at node $a_{k-1} a_{k-2} \cdots a_1 a_k$ after a shuffle operation (left cyclic rotation of address bits). $k$ shuffle operations move a data item back to its original location.

- **necklace:** Let $d_i$ denote the data item at node $i$. The nodes that $d_i$ travels through in response to repeated shuffle operations is called the **necklace** of $i$. Example: Fig.3.9,p.60
- No necklace can be longer than $\log n$, however it could be shorter than $\log n$ (called **short necklace**).
- The number of necklaces of an $n$-node shuffle-exchange network is $O(2^k/k)$, where $n = 2^k$.

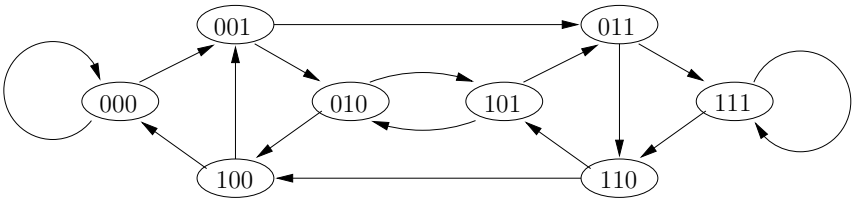


Figure 39: An 8-processor de Bruijn network (Fig.3-10,p.60)

(9) **de Bruijn Networks** (Fig.3-10,p.60)
A **de Bruijn network** of size $n$ consists of $n = 2^k$ nodes. Let $a_{k-1}a_{k-2}\ldots a_1a_0$ be the address of a node in a DBN. There are two nodes reachable via directed edges from this node:

$$a_{k-2}a_{k-3}\ldots a_1a_0 0$$

$$a_{k-2}a_{k-3}\ldots a_1a_0 1$$

- Diameter: $k$
- BSW: $2^k/k$
- Maximum num of links per node: 2, a constant.
- Length of longest edges: increases with the size of the network

(10) Summary and comparisons of various processor organizations (Table 3-1, p.61)

| Network | Nodes | Diameter | Bisection Width | Constant Number of Edges | Constant Edge Length |
|---|---|---|---|---|---|
| 1-D mesh | $k$ | $k-1$ | 1 | Yes | Yes |
| 2-D mesh | $k^2$ | $2(k-1)$ | $k$ | Yes | Yes |
| 3-D mesh | $k^3$ | $3(k-1)$ | $k^2$ | Yes | Yes |
| Binary tree | $2^k-1$ | $2(k-1)$ | 1 | Yes | No |
| 4-ary hypertree | $2^k(2^{k+1}-1)$ | $2k$ | $2^{k+1}$ | Yes | No |
| Pyramid | $(4k^2-1)/3$ | $2\log k$ | $2k$ | Yes | No |
| Butterfly | $(k+1)2^k$ | $2k$ | $2^k$ | Yes | No |
| Hypercube | $2^k$ | $k$ | $2^{k-1}$ | No | No |
| Cube-connected cycles | $k2^k$ | $2k$ | $2^{k-1}$ | Yes | No |
| Shuffle-exchange | $2^k$ | $2k-1$ | $\geq 2^{k-1}/k$ | Yes | No |
| de Bruijn | $2^k$ | $k$ | $2^k/k$ | Yes | No |

**Table 3-1** Characteristics of various processor organizations (Table 3-1, p.61)

3. Processor arrays

(1) Definitions and concepts

a. A *processor array* is a vector computer implemented as a sequential computer (called the **front end**) connected to a set of identical and synchronized processing elements capable of performing the same operation on different data (Fig.3-11,p.62).

(a) The front end is a general-purpose CPU that stores the program and the data that are not manipulated in parallel and also executes the sequential portion of the program.

(b) Each processing element may have a local memory. These local memories collectively store the vector data to be manipulated in parallel.

(c) The front end will direct processing elements to perform an instruction in parallel when an instruction whose operand is a vector is encountered.

(d) Processing units may be programmed to ignore any particular instruction. This ability to mask processing elements allow synchronization to be maintained through the various kinds of control structures, such as clauses of an **if** $\cdots$ **then** $\cdots$ **else** statements.

(e) Data flows from the front end to the processor array, between the processing elements, and from processor array to the front end.

b. Processor arrays usually posses two important abilities:

(a) An efficient mechanism for the front end to broadcast instructions and data items to individual processing elements.

(b) An efficient method to access by the front end a particular memory location of an arbitrary processing element.

c. Processor arrays can vary in three aspects:

(a) The organization of the processor array (such as the nine previously introduced organizations).

(b) Methods of communucations between processing elements. It may be either shared-memory (SM) or through interconnection network, although the latter is more popular.

(c) Number of processing elements may be fixed or vary;

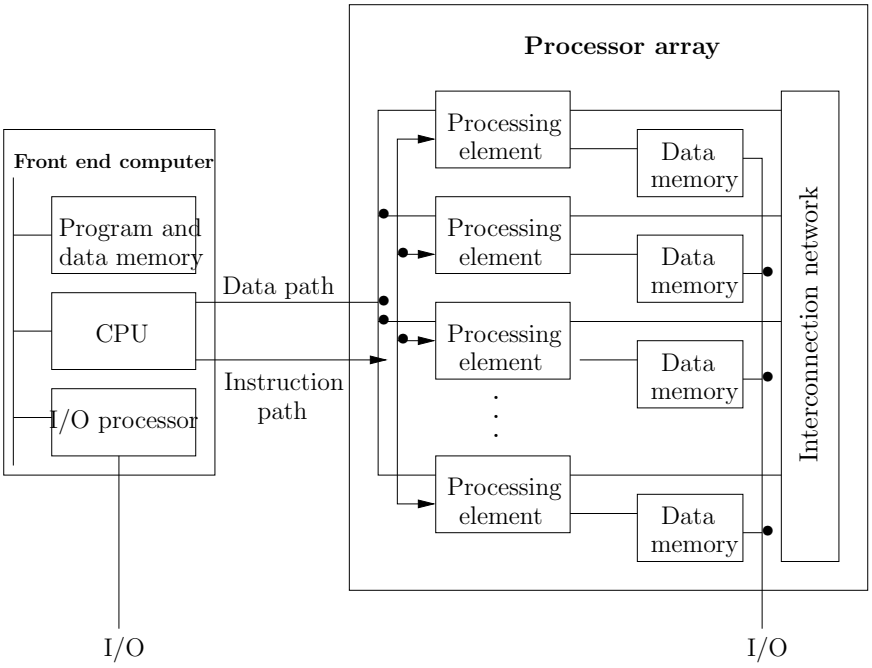


Figure 40: A realistic processor array model. Each processor has its own private memory, and processors can pass data only via a limited interconnection network (Fig.3-11,p.62)

(2) Shared-memory SIMD model

a. General structure: a control CPU, $p$ parallel processing units (PPU) can simultaneously access any $p$ locations in the entire memory space in constant time.

* EREW RAM: exclusive read, exclusive write
* CREW RAM: concurrent read, exclusive write

* CRCW RAM: concurrent read, concurrent write
  Which write succeeds? (1) If $m$ simultaneous writes, each will succeed with probability $1/m$; (2) Some PPU's have higher priority than others.

Notice that these three access models are similar to those we discussed from the PRAM model.

b. Despite literature popularity, no commercial SM SIMD processor array systems were built: it's impossible to allow simultaneous access to any $p$ locations.

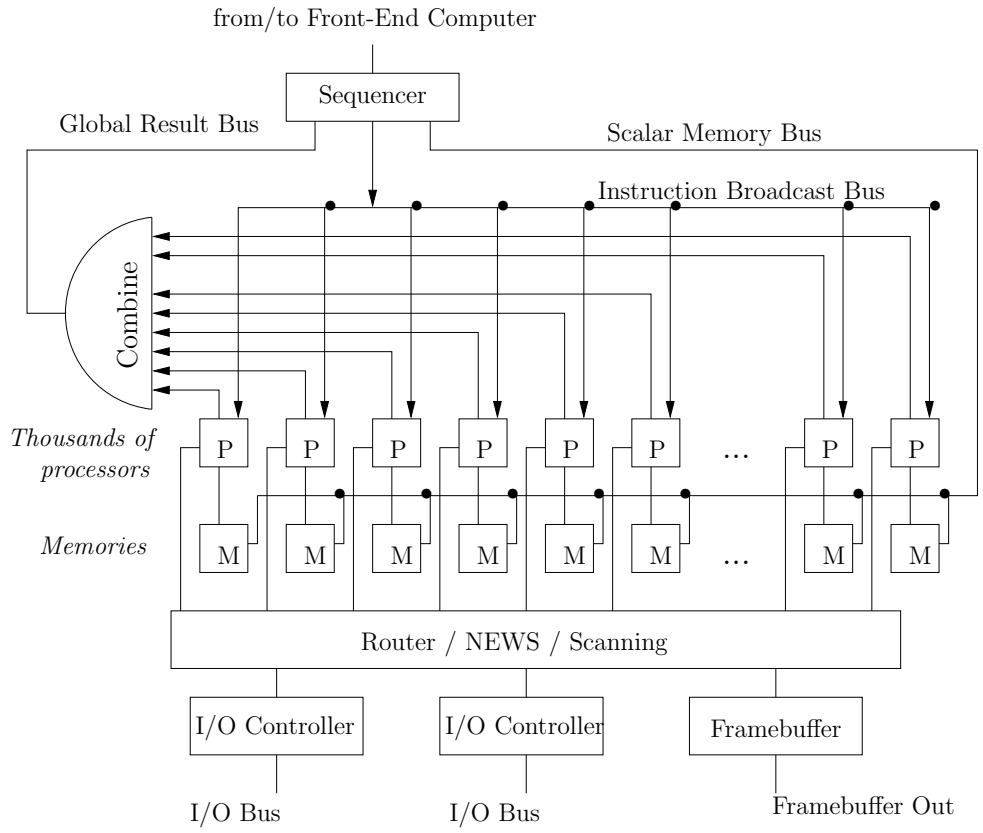


Figure 41: Block diagram of the Connection Machine CM-200 processor array. (Reprinted by permission of Thinking Machines Corporation.) (Fig.3-12,p.63)

(3) Connection-Machine CM-200 (manufactured by the Thinking Machines Corporation)

a. Three main components (Fig.3-12,p.63):

(a) a front-end computer;

(b) a parallel processing unit; and

(c) an I/O system

b. The front end (usually a SUN workstation):

(a) Store the serial data and executes the sequential portions.

(b) Store the parallel data.

(c) Broadcast parallel instructions and data to the parallel processing units.

c. The front end can exchange data with the processing elements in three different eays:

(a) Broadcast: a single value is broadcast to all processing elements.

(b) *Global combining*: obtain the sum, maximum, or global OR and etc of one value from each processing element.

(c) By using the scalar memory bus, the front end can read or write 32-bit values stored in any processing element.

d. The processing unit: contains between 2,048 to 65,536 processing elements, an instruction sequencer, interprocessor communication networks, I/O controllers, and/or framebuffer modules.

e. The front end issues parallel processing instructions to the sequencer, which interprets each instruction and generates a series of "nanoinstructions". It broadcasts these nanoinstructions over an instruction bus to the processing elements.

f. PARIS (PARallel Instruction Set of Connection Machine): simplifies the development of compilers by providing a variety of operations similar to a more typical machine's instruction set.

(a) Some PARIS instructions perform arithmetic operations on various data types, others facilitate communications between the processing elements, and still others facilitate communications between the processing elements and the front end.

(b) **Virtual processors**: PARIS supports the VP concept, thus it also serves the function of insulating the user from the underlying processor array – a program can assume the existence of any number of processing elements, thus the same program can run on Connection Machines with different number of processing elements. Virtual processing elements are mapped to physical PEs.

(c) For each PARIS arithmetic instruction performed, each physical processing element may execute the operation many times.

g. Each individual PE is a bit-serial processor.

(a) Each PE has a context flag, indicating whether or not it's screened. Screened PEs do not store the results of their computations.

(b) Every PE has three input bits (two from memory and one to a flag) and two result bits (one to memory and one to a flag).

| INPUT BITS | | | OUTPUT BITS | |
|---|---|---|---|---|
| Memory | Memory | Flag | Memory | Flag |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Fig.3-13** Truth table for bit-serial addition on CM-200 processor array. The sum bit is output to memory; the carry-out bit is output to a hardware flag.

(c) A PE can compute any two boolean functions on three inputs. These functions are specified as two 8-bit bytes representing the truth tables for these two functions. See Fig.3-13 shows a truth table for bit-serial addition.

i. The two functions on three inputs are "carry-out" and "sum".

ii. To add two $k$-bit integers stored in its local memory, a PE first loads the VP context flag into a hardware flag register – all ALU operations are conditional upon the state of this flag.

iii. Second, the PE clears a second flag that serves as the carry bit.

iv. Third, the PE iterates $k$ times through a cycle in which it reads the carry-bit and one bit of each operand and computes the sum bit and the carry-out bit.

v. The computation begins with the least significant bits and ends with their most significant bits.

h. Architecture of processor chip pair (Fig.3-14,p.66): a single VLSI chip containing 16 PEs plus routing hardware.

(a) Each pair of processor chips share a group of memory chips, a floating-point interface chip, and a floating point-execution chip.

(b) The memory chips provide a 44-bit wide data path (32-bit data, 12-bit error correction code).

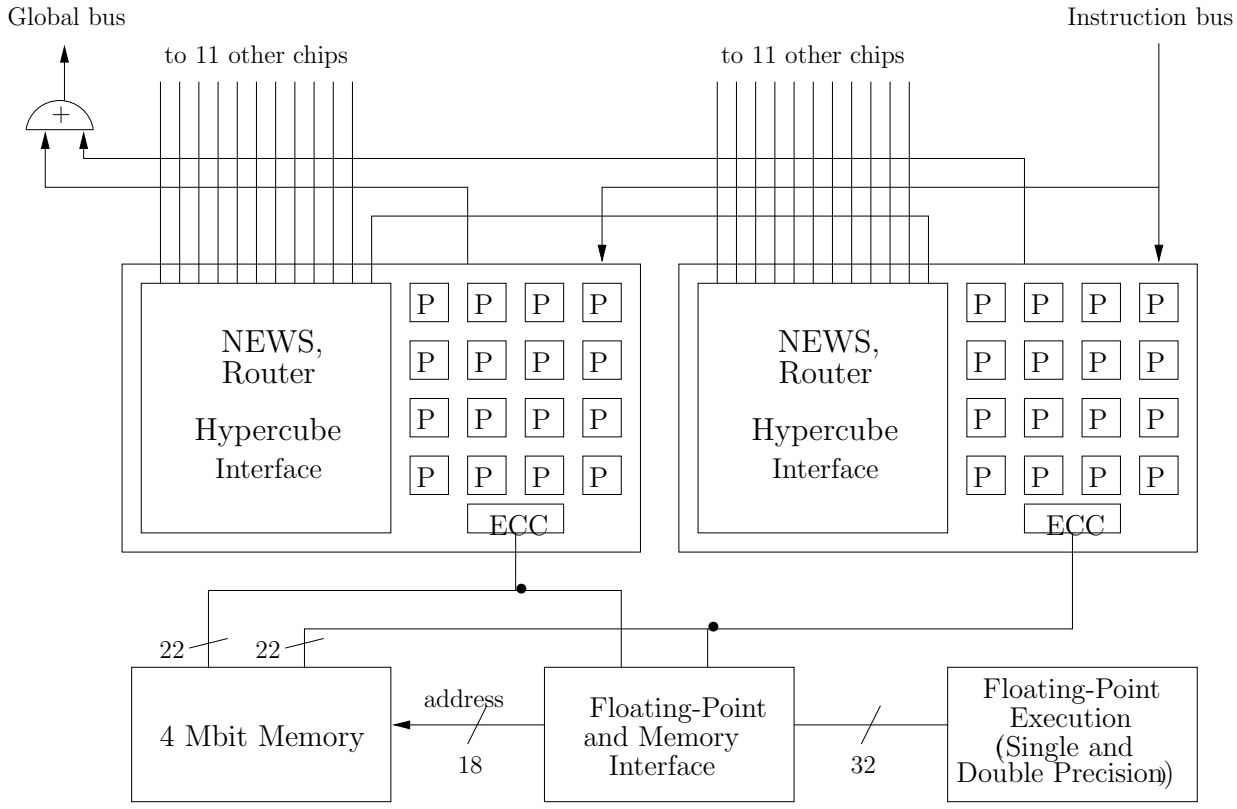


Figure 42: Architecture of processor chip pair on ther Connection Machine CM-200. (Reprinted by permission of Thinking Machines Corporation.) (Fig.3-14,p.66)

(c) The 32 data bits can be sent to the 32 bit-serial PEs (one per PE inside the 32 PEs in a processor pair). They can also be directed to the floating-point interface chip, which may use this data for memory address control for indirect addressing, or it may send the data to the floating-point execution chip.

i. The CM has three routing mechanisms:

(a) The *router*: most general, allows any PE to communicate with any other PE. Each processor chip contains one router node which services all 16 PEs on the chip.

i. The router nodes are wired together to form a hypercube. In a fully configured CM-200 there 65,546 PEs on 4,096 processor chips. The hypercube is 12-dimensional.

ii. Each message travels through the router nodes until it reachs the chip containing the destination PE.

iii. The router nodes automatically forward messages and perform

some dynamic load balancing.

iv. Each router node has an ALU capable of performing some arithmetic operations. The router checks to see if any messages share the same destination. If so, it combines the messages based upon the semantics of the parallel instruction set.

(b) The **NEWS grid**:

i. A $j$-dimensional mesh can be embedded in a $k$-dimensional hypercube if $j \leq k$ (the concept of *embedding* will be discussed in next lecture). Hence a subset of the wires that support the router can form a Cartesian mesh (called NEWS grid).

ii. If all comm. between PEs are neighbors in a NEWS grid of any dimension less than or equal to the dimension of the hypercube, then the message passing speed is much higher than if the messages are passed through the router.

(c) Message passing – *scan* and *spread*:

i. Scan: a NEWS operation that computes prefix sums;

ii. Spread: a form of broadcast that allows individual processors to send messages to every processor in the system.

j. The CM-200 uses an innovative and large I/O mechanism.

(a) The DataVault used is a parallel array of disk drives. A set of 8 DataVault, which can be connected to as many as 65,536 PEs, can provide 480 gigabytes of secondary storage and data transfer rate above 100 megabytes per second (this was a very decent number at that time).

(b) It also supports high speed output to a frame buffer driving color monitor.

4. Multiprocessors and UMA multiprocessors

(1) Multi-CPU computers: consisting of a number of fully programmable processors (each capable of executing its own programs).

a. Multiprocessors: multi-CPU computers with a shared memory.

b. UMA (uniform memory access) multiprocessors

(a) The shared memory has a centralized control (Fig.3-15,p65).

(b) How to reach the shared memory: common bus to global memory, the crossbar switch, packet-switched network.

(c) Bus based: eg. Multimax and Symmetry, limited in size. It becomes saturated if too many processors are connected.

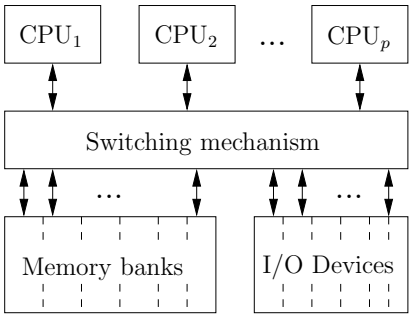


Figure 43: The uniform memory access (UMA) multiprocessor model. All the processors work through a central switching mechanism to reach a shared global memory and I/O devices (Fig.3-15,p.68)

(d) Switching network based: can connect a large number of processors. NYU's Ultracomputer (an experimental UMA multiprocessor) is based on an omega switching network. The cost of an omega network for a $p$-processor system is $\Theta(p \log p)$ – lower than the $\Theta(p^2)$ cost of a crossbar switch.

(2) **Symmetry** UMA multiprocessor (Fig,3-16, p.69)

a. Manufactured by Sequent Computer Systems Inc. (of Beaverton, Oregon. Note: now part of IBM).

b. A commercial example of UMA multiprocessor system. It's not a supercomputer – some contemporary workstations have higher floating-point performance.

c. Uses a pipelined 64-bit bus to connect the processors, memory, and I/O devices.

(a) The SSB (Sequent System Bus) carries 32-bit or 64-bit data items and addresses upto 32 bits in length.

(b) Read and write operations on the SSB are pipelined (it can handle another read/write transaction before the the previous one has been served by the memory).

(c) The sustained data transfer rate on the SSB is 53.3 MBytes/second.

d. The processors

(a) They are based on Intel 80386 CPU (a 32-bit microprocessor) and an Intel 80387 floating-point co-processor.

(b) An optional Weitek WTL 1167 chip is available to enhance the speed of some floating-point operations.

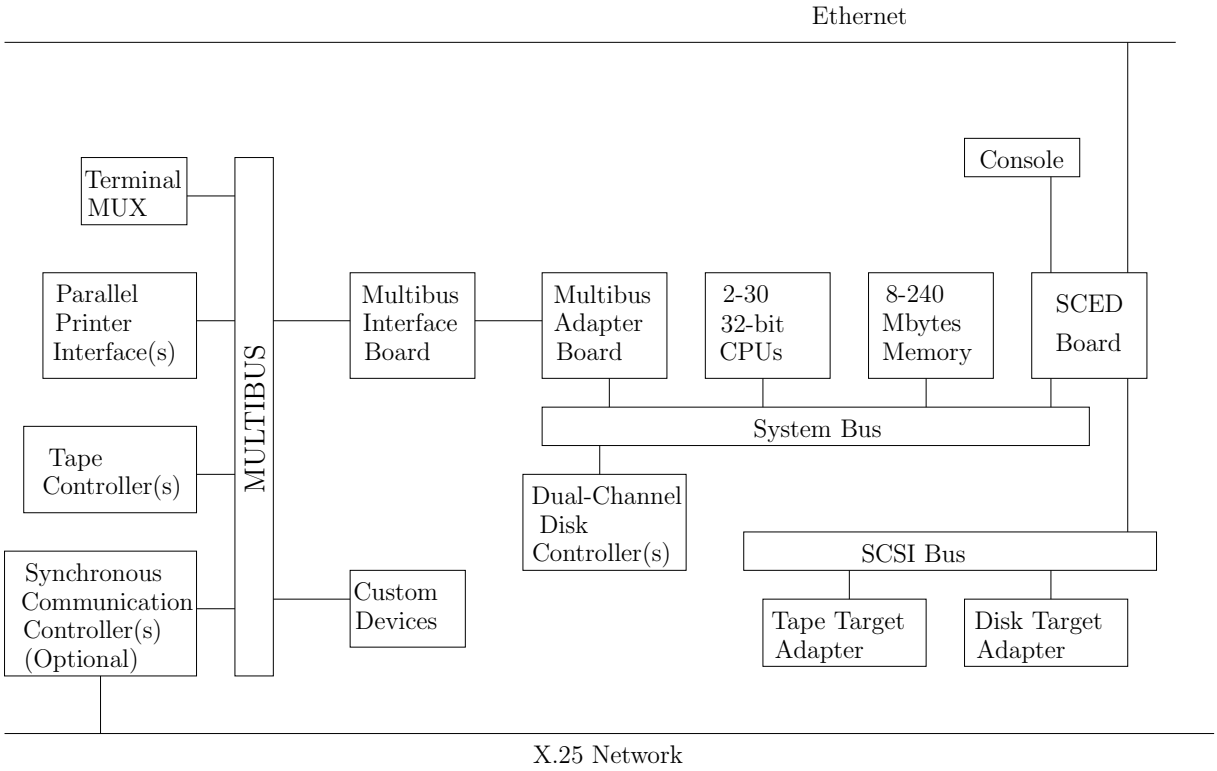


Figure 44: Block diagram of the Sequent Symmetry, a UMA mulitprocessor. (Courtesy Hatcher and Quinn [1991].) (Fig.3-16,p.69)

(c) Each processor has a special System Link and Interrupt Controller to manage interactions among processors (386 is not designed for multiprocessors).

e. The processors uses another 1-bit wide bus to exchange interrupts and other low-level control and error signals.

f. Memories

(a) Each processor has a 64 Kbytes cache.

(b) Cache memories: used to reduce contentions of SSB and keep processors busy.

(c) If a memory fault occurs, the cache control suspends the processor and issues a 16-byte read request on the SSB.

g. Cache consistency: how to maintain consistency between copies of data in the main memory and the local cache?

(a) *Write-through* caching policy (used in older Symmetry machines): every write is sent directly to the system memory, and all copies of the data item in the other processors are invalidated.

(b) *Copy-back* caching policy (used in new Symmetry machines): when a processor modifies a data item in its cache, it does not write the updated value to system memory until it swaps out the cache block or until another processor needs that data item. It does signal other processors that their value for that data item is no longer valid.

(c) Single-byte and 16-/32-byte loads and stores operations always execute atomically.

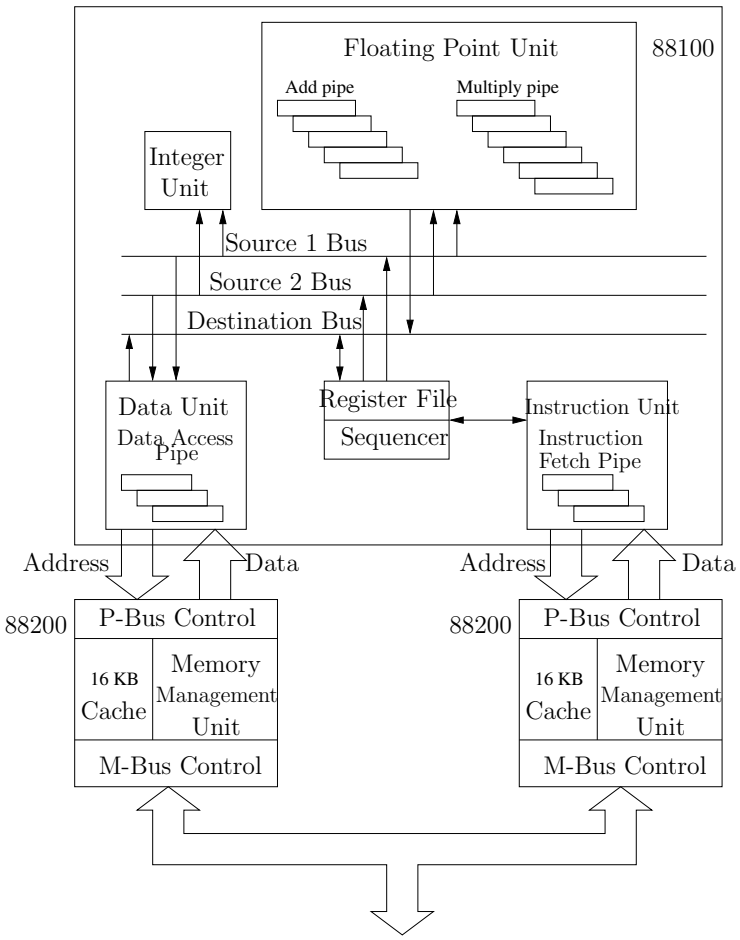


Figure 45: A processor node of the BBN TC2000. (Reprinted by permission of Bolt, Beranek, and Newman, Inc.) (Fig.3-17,p.71)

(3) NUMA Multiprocessors

a. NUMA (Non-uniform memory access) multiprocessors

(a) Still characterized by a shared memory.

(b) However, the shared memory is distributed. Every processor has some nearby memory. The shared address space on a NUMA multiprocessor is formed by combining these local memories.

b. TC2000, manufactured by BBN Systems and Technologies (of Cambridge, MA).

(a) Can have upto 128 processor nodes.

i. Each processor node contains a Motorola 88100 CPU.

ii. Each processor node also contains three Motorola 88200 chips (providing separate instruction and data cache between 4 and 16 Mbytes of primary memory, and interface to the Butterfly switch network, and a VME bus).

iii. The transaction bus connects these processor subsystems.

(b) The maximum performance of a single processor node is 29 megaflops. For a 128-node system, peak speed of 2.5 gigaflops.

(c) The hardware of each processor converts a 32-bit virtual address into a 34-bit physical address: Cache memory, the local memory, and memory in another processor. No cache consistency is maintained.

(d) The Butterfly switching network (Fig.3-18,p.72).

i. Each element in the butterfly network has 8 input and 8 output pins. For a $p$-processor machine, $p/8 \log_8 p$ (the book said it was $p \log_8 p$, which was incorrect). such switching elements are needed.

ii. How to make the memory reference over the butterfly network?

5. Multicomputers

(1) Characterized by the lack of shared memory. Processors communicate by message passing: Intel's Paragon, XP/S, Meiko's Computing Surface, nCUBE's nCUBE 2, Parsytec's SuperCluster, and Thinking Machine's CM-5.

(2) Difference between early multicomputer systems and contemporary multicomputer systems (Fig.3-20,p.73, Fig.3-21,p.74):

a. Early's use software managed *store-and-forward message passing*.

b. Contemporary use *circuit-switched message routing*: *direct-connect modules* are used to set up a circuit.

(3) nCUBE 2.

a. Manufactured by nCUBE Corp. (of Foster City, CA).

b. System structure

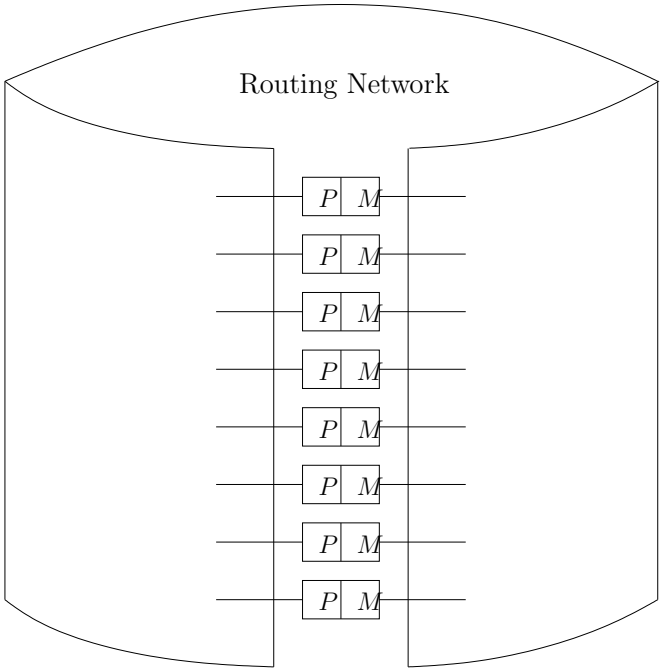


Figure 46: Nonlocal memory references on the TC2000 are sent through the switch. (Repriented by permission of Bolt, Beranek, and Newman, Inc.) (Fig.3-18,p.72)

(a) Three main components: front-end computer, a back-end array of processors, and parallel I/O interface (Fig.3-22,p.74).

(b) The back-end processors are organized as a hypercube. Although it resembles a processor array on the surface, they are actually different:

i. Processors in a processor array are controled by its front end. They execute the same instruction streams fed by the front end.

ii. Processors in nCUBE are independent. Each processor is an totally independent CPU. It executes its own instruction streams.

(c) The hypercube edges are hardware-routed, DMA communication channels. The nCUBE 2 hence is classified as a the second generation multicomputer.

(d) Each node also has a dedicated DMA channel connected to the I/O devices that can perform at the same speed as the hypercube edges.

c. Physical specifications

(a) The backend hypercube can have upto 8,192 nodes. Each node has a peak performance of 2.5 megaflops. The theoretical peak performance hence is about 20 gigaflops for a 8192-node nCUBE.

(b) The hypercube edges can transfer data between processors at a peak

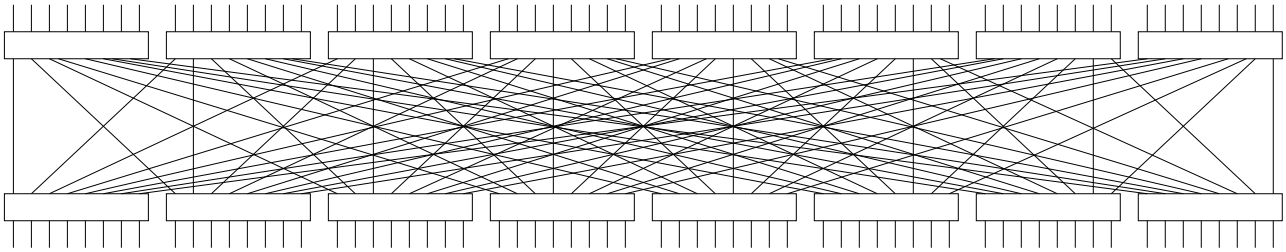

Figure 47: Switching network for a 64-processor TC2000 (Fig.3-19,p.72)

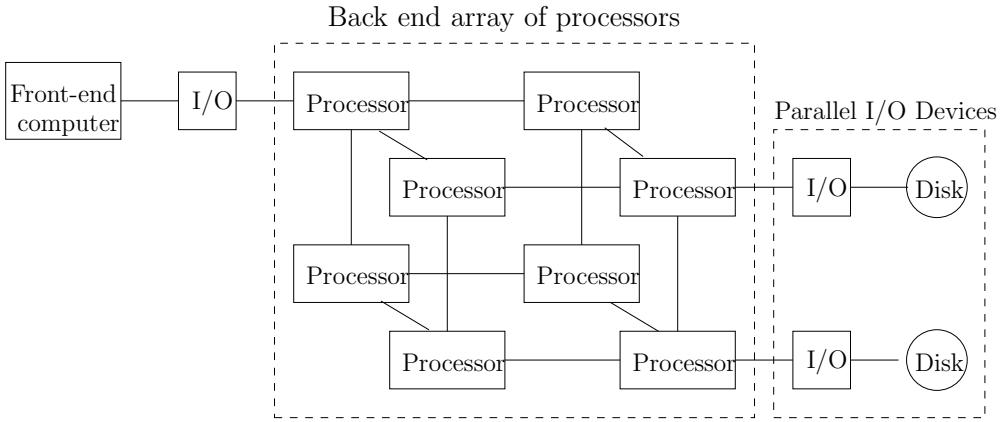


Figure 48: Block diagram of an nCUBE 2 multicomputer with eight back-end processors and two parallel I/O devices (Fig.3-22,p.74)

rate of 2.5MBytes/sec.

d. Problems with nCUBE 2. These two problems are common with multicomputer systems:

(a) The computational power of the frond-end computer does not scale well with the number of back-end processors. When the number of users scales with the number of processors, at some point the front-end will be unable to support the users.

(b) The front-end is isolated from the back-end.

6. Flynn's Taxonomy of computer architectures

(1) Many different classifications, inherently contravertial. Use Flynn's method (1966). Its based on notions of instruction stream and data stream.

a. **Instruction stream**: a sequence of instructions performed by a computer.

b. **Data stream**: a sequence of data items used to execute an instruction stream.

(2) Four classes of computers classified using Flynn's taxonomy:

a. **Single instruction stream, single data stream** (SISD). Most sequential computers are in this group. Even some pipelining computers.

b. **Single instruction stream, multiple data stream** (SIMD). Processor arrays. In any time unit a single operation is in the same stage of operation on multiple processing units, each manipulating different data items.

c. **Multiple instruction stream, single data stream** (MISD). No computers fall in this category.

d. **Multiple instruction stream, multiple data stream** (MIMD). Most general and powerful, including multiprocessor and multicomputer machines.

7. Superlinear speedup, scaled speedup, and parallelizability

(1) Superlinear speedup

a. When the speedup of a parallel algorithm is larger than $p$, the number of processing elements, the speedup is called *superlinear* speedup.

b. Is superlinear speedup possible. Two views, no and yes.

c. Arguments that superlinear speedup is not possible.

(a) A single processor can always emulate parallel processors.

(b) If a parallel algorithm $A$ solves a given instance of problem $\Pi$ in $T_p$ time units on a parallel computer with $p$ processors, then algorithm $A$ can solve the same problem instance in $p \times T_p$ time units on the same computer with one processor through time slicing. Hence the speedup cannot be more than $p$.

(c) In fact, because sequential algorithms do not have communication and synchronization overhead, it is more likely that there exists a sequential algorithm that can solve the given problem instance in less $p \times T_p$ time units.

d. Believers of superlinear speedup: the two assumptions made in above arguments are not sound.

(a) Speedup should be defined as the ratio between the time needed the best known sequential algorithm and the time taken by the parallel algorithm, over all possible problem instances of the problem. Is it unreasonable to choose an algorithm after the problem instance is given. Due to the nature of parallel algorithms, it is possible to have parallel algorithms that exhibit superlinear speedup for some problem

instances. Namely, for these special problem instances, a parallel algorithm that uses $p$ processors can do more than $p$ times faster than best known sequential algorithms.

(b) It is always debatable whether a single processor can emulate multiple processors without loss of efficiency. For example, with a group of cache memory blocks distributed across each processor element in a parallel computer, the hit ratio is much higher than using only a single processor element with one cache memory block.

(2) Scaled speedup and scaled efficiency

a. Recall Amdahl's law and Amdahl effect:
(a)

$$S \leq \frac{1}{f + (1-f)/p}$$

(b) Amdahl effect: the portion of the operations that are sequential decreases as the sizes of the problems increase.

b. Corollary of Amdahl's law: a small fraction of sequential computation in a problem can significantly limit the speedup achievable by a parallel computer.
Example: if 10% of the operations must be performed in sequence, then the maximum possible speedup is 10, regardless of the number of processing elements avaiable in a parallel computer.

c. Problem's with Amdahl's law.

(a) The base of Amdahl's law: parallel processing is used to reduce the time in which a problem of some particular size can be solved. This assumption is valid in some cases and is debatable in some other instances.

(b) More intuitively, it is believed that

> The concept of speedup alone cannot accurately measure and reflect the efficiency and utilization of parallel algorithms. Amdahl's law only reflects one facet of parallel computation.

d. Scaled speedup and scaled efficiency

(a) Scaled speedup is the ratio between how long a given optimal sequential program would have taken, had it been able to run on a single processor of a parallel computer, and the length of the time that the parallel program requires when being executed at full speed on the same parallel computer.

(b) Scaled efficiency is the scaled speedup divided by the number of processors used.

(c) Notes: the time taken by a single processor of a parallel computer may not be equal to the time taken by a sequential computer. Parallel computers have multiple resources such as memory chips that are scattered around in the system. These resources are not available on many sequential computers.

Scaled speedup is also called *parallelizability*.

e. Example application of scaled speedup and scaled efficiency

(a) Assume: we intend to solve the largest system of linear equations in limited time, say one minute. We wish to verify whether we can make efficient use of a particular available parallel computer to perform the task.

(b) Assume that a good sequential algorithm performs about $2n^3/3$ floating-point operations to solve a dense $n \times n$ linear system. Given a processor capable of 100 million floating-point operations per second, the largest linear system we can solve in one minute is approximately:

$$\frac{(2n^3/3) \text{ ops}}{100,000,000 \text{ ops/sec}} \leq 60 \text{ sec} \Rightarrow n \leq 2,080$$

(c) Now we investigate the time needed by a 1024-processor parallel computer to solve the linear system.

i. Each processor can perform 100 million floating-point operations per second.

ii. Assume that the communication cost is 1 microsecond (message initiation) plus 5 nanoseconds (message transmission) for transfering a floating point value.

iii. The total number of floating point operations is $2n^3/3$. For $p$ processors, each processor perform $2n^3/(3p)$ operations.

iv. Assume that each of the $p$ processors sends or receives $4n \log p$ messages that contains a total of about $n^2/2$ floating point values.

v. With these assumptions, and assuming that the entire linear system to be solved fits into the RAM of the parallel computer, the largest linear systems that can be solved in one minute on a 1024-processor computer is (the 1st item is the total time of performing floating point operations, the 2nd is the total message initiation time, and the 3rd is the total message transmission time. For the 3rd item, because message transmission time is 5 nanoseconds per

floating point value, that is equal to $2 \times 10^8$ floating point values per second):

$$\frac{(2n^3/3072) \text{ ops}}{1 \times 10^8 \text{ ops/sec}} + \frac{(40n) \text{ msgs}}{1 \times 10^6 \text{ msgs/sec}} + \frac{(n^2/2) \text{ floats}}{2 \times 10^8 \text{ floats/sec}} \leq 60 \text{ sec} \Rightarrow$$

$$6.51 \times 10^{-12} n^3 + 4.00 \times 10^{-5} n + 2.50 \times 10^{-9} n^2 \leq 60 \Rightarrow$$

$$n \leq 20,700$$

(d) Scaled efficiency:

i. A single 100-megaflop processor executing the sequential algorithm needs 59,000 seconds to solve a problem of size 20,700.

ii. The scaled speedup is then

$$\frac{59,000}{60} = 983$$

ii. The scaled efficiency is then

$$\frac{983}{1024} = 96\%$$

That is to say the utilization of the 1024 processors is very high when solving a system of size 20,700 in 60 seconds.