

10

ADVANCED TOPICS

In this chapter, we explore a few advanced topics in computer architecture. The early portion of the chapter covers parallel architecture, which is an approach at improving performance by applying multiple computers to work on the same problem. The latter portion of the chapter covers network architecture at a greater depth than is covered in the introduction to local area networks (LANs) covered in Chapter 8.

10.1 Parallel Architecture

One method of improving the performance of a processor is to decrease the time needed to execute instructions. This will work up to a limit of about 400 MHz (Stone, 1991), at which point an effect known as **ringing** on busses prohibits further speedup with conventional bus technology. This is not to say that higher clock speeds are not possible, but that “shared bus” approaches become impractical at these speeds. Commercial microprocessors already operate at these speeds, and as conventional architectural approaches to improving performance wear out, we need to consider alternative methods of improving performance.

One alternative approach is to increase the number of processors, and decompose and distribute a single program onto the processors. This approach is known as **parallel processing**, because a number of processors work collectively, in parallel, on a common problem. We see an example of parallel processing in Chapter 9 with pipelining. For that case, four processors are connected in series (Figure 9-6), each performing a different task, like an assembly line in a factory. The interleaved memory described in Chapter 6 is another example of pipelining.

A parallel architecture can be characterized in terms of three parameters: (1) the

number of **processing elements** (PEs); (2) the interconnection network among the PEs; and (3) the organization of the memory. In the four-stage instruction pipeline of Figure 9-6, there are four PEs. The interconnection network among the PEs is a simple ring. The memory is an ordinary RAM that is external to the pipeline.

Characterizing the architecture of a parallel processor is a relatively easy task, but measuring the performance is not nearly so simple. Although we can easily measure the increased speed that a simple enhancement like a pipeline offers, the overall speedup is data dependent: not all programs and data sets map well onto a pipelined architecture. Other performance considerations of pipelined architectures that are also data dependent are the cost of flushing a pipeline, the increased area cost, the latency (input to output delay) of the pipeline, *etc.*

A few common measures of performance are **parallel time**, **speedup**, **efficiency**, and **throughput**. The parallel time is simply the absolute time needed for a program to execute on a parallel processor. The speedup is the ratio of the time for a program to execute on a sequential (non-parallel, that is) processor to the time for that same program to execute on a parallel processor. In a simple form, we can represent speedup (S) as:

$$S = \frac{T_{\text{Sequential}}}{T_{\text{Parallel}}}$$

Since a sequential algorithm and a parallel version of the same algorithm may be programmed very differently for each machine, we need to qualify $T_{\text{Sequential}}$ and T_{Parallel} so that they apply to the best implementation for each machine.

There is more to the story. If we want to achieve a speedup of 100, it is not enough to simply distribute a single program over 100 processors. The problem is that not many computations are easily decomposed in a way that fully utilizes the available PEs. If there are even a small number of sequential operations in a parallel program, then the speedup can be significantly limited. This is summarized by **Amdahl's law**, in which speedup is expressed in terms of the number of processors p and the fraction of operations that must be performed sequentially f :

$$S = \frac{1}{f + \frac{1-f}{p}}$$

For example, if $f = 10\%$ of the operations must be performed sequentially, then

speedup can be no greater than 10 regardless of how many processors are used:

$$S = \frac{1}{0.1 + \frac{0.9}{10}} \cong 5.3 \quad p = 10 \text{ processors}$$

$$S = \frac{1}{0.1 + \frac{0.9}{\infty}} = 10 \quad p = \infty \text{ processors}$$

This brings us to measurements of **efficiency**. Efficiency is the ratio of speedup to the number of processors used. For a speedup of 5.3 with 10 processors, the efficiency is:

$$\frac{5.3}{10} = .53, \text{ or } 53\%$$

If we double the number of processors to 20, then the speedup increases to 6.9 but the efficiency reduces to 34%. Thus, parallelizing an algorithm can improve performance to a limit that is determined by the amount of sequential operations. Efficiency is drastically reduced as speedup approaches its limit, and so it does not make sense to simply use more processors in a computation in the hope that a corresponding gain in performance will be achieved.

Throughput is a measure of how much computation is achieved over time, and is of special concern for I/O bound and pipelined applications. For the case of a four stage pipeline that remains filled, in which each pipeline stage completes its task in 10 ns, the average time to complete an operation is 10 ns even though it takes 40 ns to execute any one operation. The overall throughput for this situation is then:

$$0.1 \frac{\text{operation}}{\text{ns}} = 10^8 \text{ operations per second.}$$

10.1.1 THE FLYNN TAXONOMY

Computer architectures can be classified in terms of their **instruction streams** and their **data streams**, using a taxonomy developed by M. J. Flynn (Flynn, 1972). A conventional sequential processor fits into the **single-instruction stream, single data stream (SISD)** category, as illustrated in Figure 10-1a. Only a single instruction is executed at a time in a SISD processor, although pipelining may allow several instructions to be in different phases of execution at any given

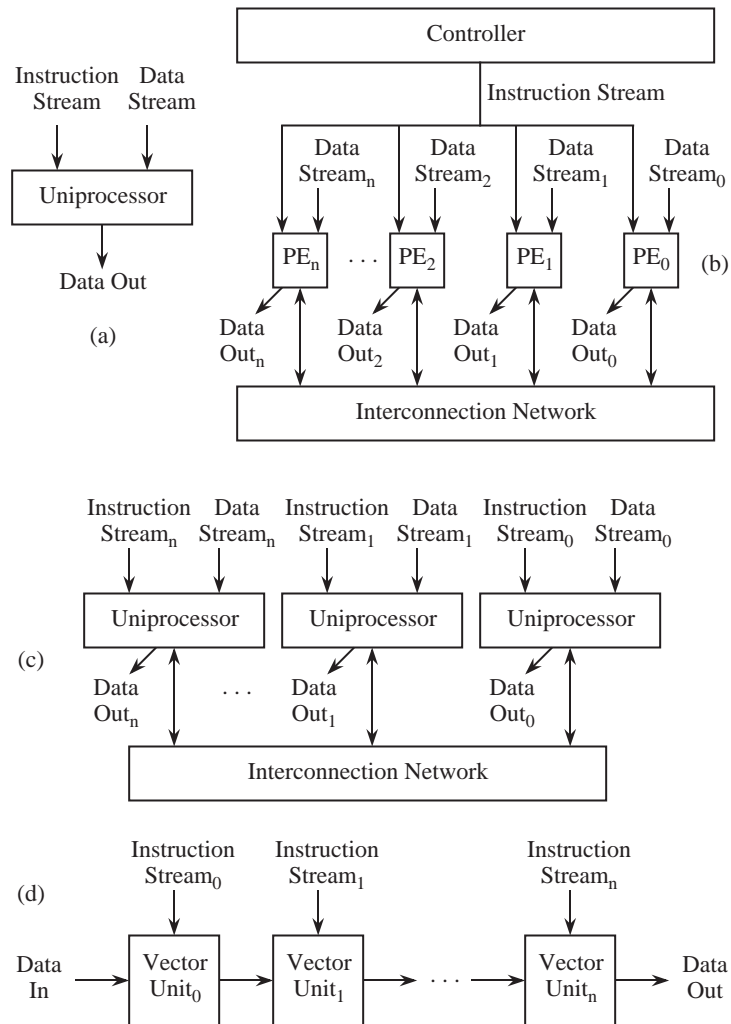


Figure 10-1 Classification of architectures according to the Flynn taxonomy: (a) SISD; (b) SIMD; (c) MIMD; (d) MISD.

time.

In a **single instruction stream, multiple data stream** (SIMD) processor, several identical processors perform the same sequence of operations on different data sets, as illustrated in Figure 10-1b. A SIMD system can be thought of as a room filled with mail sorters, all sorting different pieces of mail into the same set of bins.

In a **multiple instruction stream, multiple data stream** (MIMD) processor, several processors perform different operations on different data sets, but are all coordinated to execute a single parallel program, as illustrated in Figure 10-1c. An example of a MIMD processor is the Sega home video entertainment system, which has four processors for (1) sound synthesis (a Yamaha synthesis processor); (2) sound filtering (a Texas Instruments programmable sound generator); (3) program execution (a 68000); and (4) background processing (a Z80). We will see more of the Sega Genesis in the Case Study at the end of the chapter.

In a **multiple instruction stream, single data stream** (MISD) processor, a single data stream is operated on by several functional units, as illustrated in Figure 10-1d. The data stream is typically a vector of several related streams. This configuration is known as a **systolic array**, which we see in Chapter 3 in the form of an array multiplier.

10.1.2 INTERCONNECTION NETWORKS

When a computation is distributed over a number of PEs, the PEs need to communicate with each other through an **interconnection network**. There is a host of topologies for interconnection networks, each with their own characteristics in terms of **crosspoint complexity** (an asymptotic measure of area), **diameter** (the length of the worst case path through the network), and **blocking** (whether or not a new connection can be established in the presence of other connections). A few representative topologies and control strategies for configuring networks are described below.

One of the most powerful network topologies is the **crossbar**, in which every PE is directly connected to every other PE. An abstract view of a crossbar is illustrated in Figure 10-2a, in which four PEs are interconnected. In a closeup view illustrated in Figure 10-3, the crossbar contains **crosspoint switches**, which are configurable devices that either connect or disconnect the lines that go through it. In general, for N PEs, the crosspoint complexity (the number of crosspoint switches) is N^2 . In Figure 10-2a, $N = 4$ (not 8) because the output ports of the PEs on the left and the input ports of the PEs on the right belong to the same PEs. The crosspoint complexity is thus $4^2 = 16$. The network diameter is 1 since every PE can directly communicate with every other PE, with no intervening PEs. The number of crosspoint switches that are traversed is not normally considered in evaluating the network diameter. The crossbar is **strictly nonblocking**, which means that there is always an available path between every input and output regardless of the configuration of existing paths.

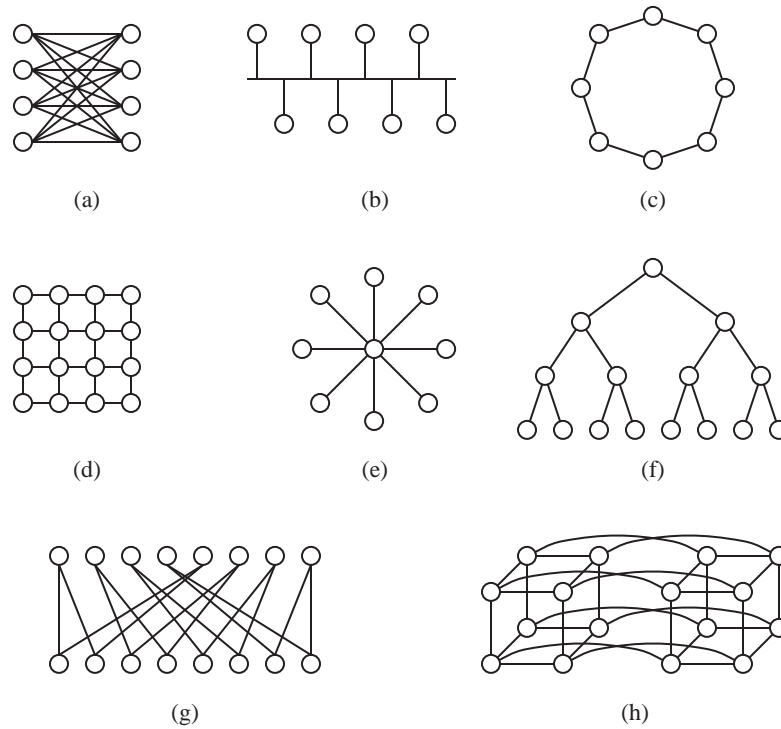


Figure 10-2 Network topologies: (a) crossbar; (b) bus; (c) ring; (d) mesh; (e) star; (f) tree; (g) perfect shuffle; (h) hypercube.

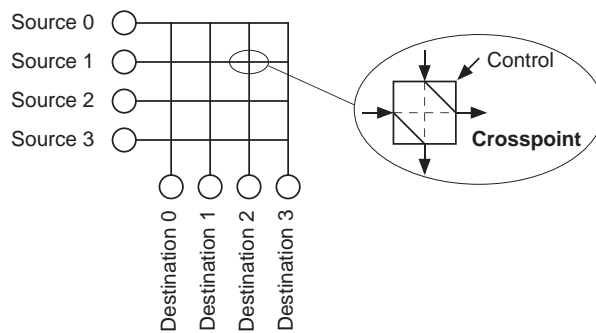


Figure 10-3 Internal organization of a crossbar.

At the other extreme of complexity is the bus topology, which is illustrated in Figure 10-2b. With the bus topology, a fixed amount of bus bandwidth is shared among the PEs. The crosspoint complexity is N for N PEs, and the network diameter is 1, so the bus grows more gracefully than the crossbar. There can only

be one source at a time, and there is normally only one receiver, so blocking is a frequent situation for a bus.

In a ring topology, there are N crosspoints for N PEs as shown in Figure 10-2c. As for the crossbar, each crosspoint is contained within a PE. The network diameter is $N/2$, but the collective bandwidth is N times greater than for the case of the bus. This is because adjacent PEs can communicate directly with each other over their common link without affecting the rest of the network.

In the mesh topology, there are N crosspoints for N PEs, but the diameter is only $2\sqrt{N}$ as shown in Figure 10-2d. All PEs can simultaneously communicate in just $3\sqrt{N}$ steps, as discussed in (Leighton, 1992) using an **offline routing algorithm** (in which the crosspoint settings are determined external to the PEs).

In the star topology, there is a central hub through which all PEs communicate as shown in Figure 10-2e. Since all of the connection complexity is centralized, the star can only grow to sizes that are bounded by the technology, which is normally less than for decentralized topologies like the mesh. The crosspoint complexity within the hub varies according to the implementation, which can be anything from a bus to a crossbar.

In the tree topology, there are N crosspoints for N PEs, and the diameter is $2\log_2 N - 1$ as shown in Figure 10-2f. The tree is effective for applications in which there is a great deal of distributing and collecting of data.

In the perfect shuffle topology, there are N crosspoints for N PEs as shown in Figure 10-2g. The diameter is $\log_2 N$ since it takes $\log_2 N$ passes through the network to connect any PE with any other in the worst case. The perfect shuffle name comes from the property that if a deck of 2^N cards, in which N is an integer, is cut in half and interleaved N times, then the original configuration of the deck will be restored. All N PEs can simultaneously communicate in $3\log_2 N - 1$ passes through the network as presented in (Wu and Feng, 1981).

Finally, the hypercube has N crosspoints for N PEs, with a diameter of $\log_2 N - 1$, as shown in Figure 10-2h. The smaller number of crosspoints with respect to the perfect shuffle topology is balanced by a greater connection complexity in the PEs.

Let us now consider the behavior of blocking in interconnection networks. Figure 10-4a shows a configuration in which four processors are interconnected

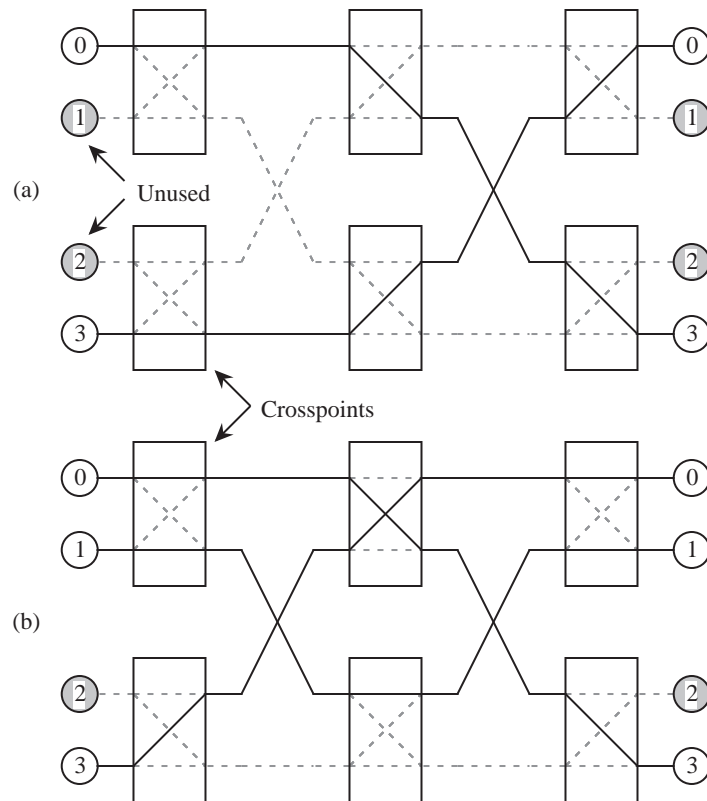


Figure 10-4 (a) Crosspoint settings for connections $0 \rightarrow 3$ and $3 \rightarrow 0$; (b) adjusted settings to accommodate connection $1 \rightarrow 1$.

with a two-stage perfect shuffle network in which each crosspoint either passes both inputs straight through to the outputs, or exchanges the inputs to the outputs. A path is enabled from processor 0 to processor 3, and another path is enabled from processor 3 to processor 0. Neither processor 1 nor processor 2 needs to communicate, but they participate in some arbitrary connections as a side effect of the crosspoint settings that are already specified.

Suppose that we want to add another connection, from processor 1 to processor 1. There is no way to adjust the unused crosspoints to accommodate this new connection because all of the crosspoints are already set, and the needed connection does not occur as a side effect of the current settings. Thus, connection $1 \rightarrow 1$ is now blocked.

If we are allowed to disturb the settings of the crosspoints that are currently in

use, then we can accommodate all three connections, as illustrated in Figure 10-4b. An interconnection network that operates in this manner is referred to as a **rearrangeably nonblocking network**.

The three-stage **Clos network** is **strictly nonblocking**. That is, there is no need to disturb the existing settings of the crosspoints in order to add another connection. An example of a three stage Clos network is shown in Figure 10-5 for four

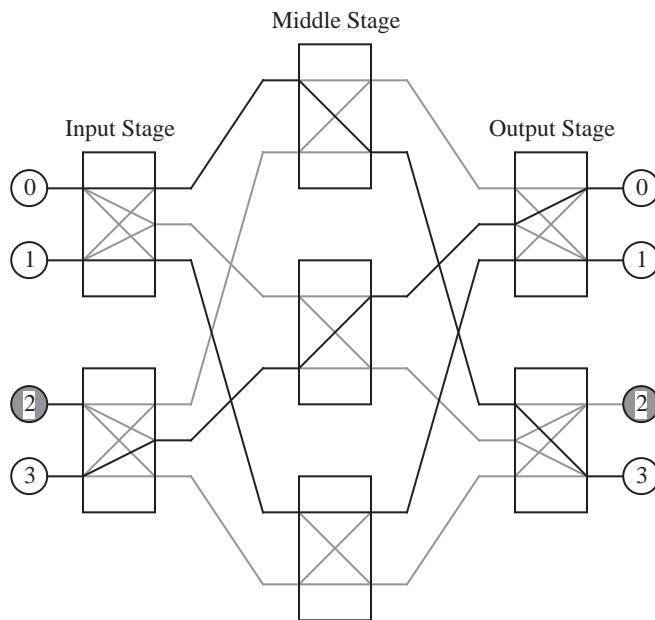


Figure 10-5 A three-stage Clos network for four PEs.

PEs. In the input stage, each crosspoint is actually a crossbar that can make any connection of the two inputs to the three outputs. The crosspoints in the middle stage and the output stage are also small crossbars. The number of inputs to each input crosspoint and the number of outputs from each output crosspoint is selected according to the desired complexity of the crosspoints, and the desired complexity of the middle stage.

The middle stage has three crosspoints in this example, and in general, there are $(n - 1) + (p - 1) + 1 = n + p - 1$ crosspoints in the middle stage, in which n is the number of inputs to each input crosspoint and p is the number of outputs from each output crosspoint. This is how the three-stage Clos network maintains a strictly nonblocking property. There are $n - 1$ ways that an input can be blocked

at the output of an input stage crosspoint as a result of existing connections. Similarly, there are $p - 1$ ways that existing connections can block a desired connection into an output crosspoint. In order to ensure that every desired connection can be made between available input and output ports, there must be one more path available.

For this case, $n = 2$ and $p = 2$, and so we need $n + p - 1 = 2 + 2 - 1 = 3$ paths from every input crosspoint to every output crosspoint. Architecturally, this relationship is satisfied with three crosspoints in the middle stage that each connect every input crosspoint to every output crosspoint.

EXAMPLE: STRICTLY NONBLOCKING NETWORK

For this example, we need to design a strictly nonblocking (3-stage Clos) network for 12 channels (12 inputs and 12 outputs to the network) while maintaining a low maximum complexity of any crosspoint in the network.

There are a number of ways that we can organize the network. For the input stage, we can have two input nodes with 6 inputs per node, or 6 input nodes with two inputs per node, to list just two possibilities. We have similar choices for the output stage. Let us start by looking at a configuration that has two nodes in the input stage, and two nodes in the output stage, with 6 inputs for each node in the input stage and 6 outputs for each node in the output stage. For this case, $n = p = 6$, which means that $n + p - 1 = 11$ nodes are needed in the middle stage, as shown in Figure 10-6. The maximum complexity of any node for this case is $6 \times 11 = 66$, for each of the input and output nodes.

Now let us try using 6 input nodes and 6 output nodes, with two inputs for each input node and two outputs for each output node. For this case, $n = p = 2$, which means that $n + p - 1 = 3$ nodes are needed in the middle stage, as shown in Figure 10-7. The maximum node complexity for this case is $6 \times 6 = 36$ for each of the middle stage nodes, which is better than the maximum node complexity of 66 for the previous case.

Similarly, networks for $n = p = 4$ and $n = p = 3$ are shown in Figure 10-8 and Figure 10-9, respectively. The maximum node complexity for each of these networks is $4 \times 7 = 28$ and $4 \times 4 = 16$, respectively. Among the four configurations studied here, $n = p = 3$ gives the lowest maximum node complexity. ■

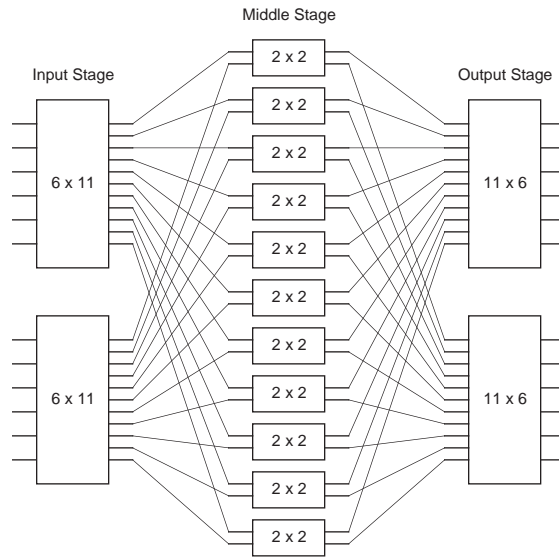


Figure 10-6 A 12-channel three-stage Clos network with $n = p = 6$.

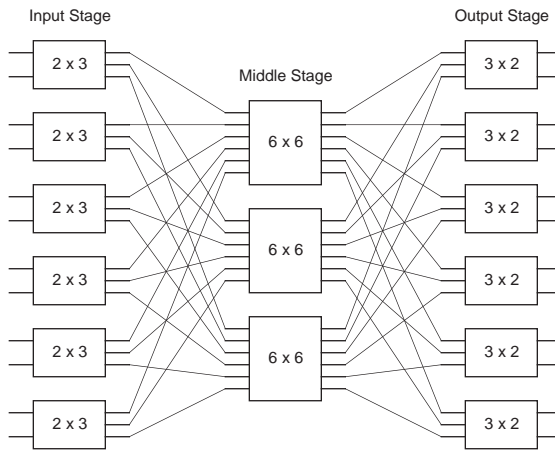


Figure 10-7 A 12-channel three-stage Clos network with $n = p = 2$.

10.1.3 MAPPING AN ALGORITHM ONTO A PARALLEL ARCHITECTURE

The process of mapping an algorithm onto a parallel architecture begins with a **dependency analysis** in which data dependencies among the operations in a program are identified. Consider the C code shown in Figure 10-10. In an ordinary SISD processor, the four numbered statements require four time steps to

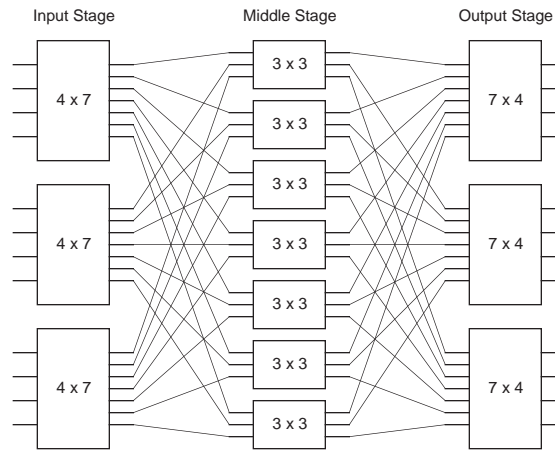


Figure 10-8 A 12-channel three-stage Clos network with $n = p = 4$.

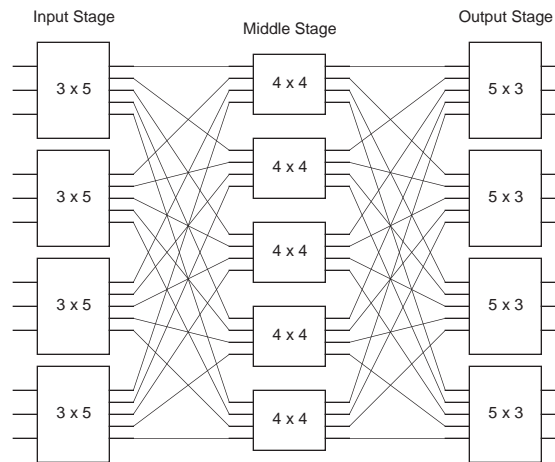


Figure 10-9 A 12-channel three-stage Clos network with $n = p = 3$.

complete, as illustrated in the control sequence of Figure 10-11a. The **dependency graph** shown in Figure 10-11b exposes the natural parallelism in the control sequence. The dependency graph is created by assigning each operation in the original program to a node in the graph, and then drawing a directed arc from each node that produces a result to the node(s) that needs it.

The control sequence requires four time steps to complete, but the dependency graph shows that the program can be completed in just three time steps, since operations 0 and 1 do not depend on each other and can be executed simulta-

Operation numbers

```

func(x, y) /* Compute  $(x^2 + y^2) \times y^2$  */
int x, y;
{
    int temp0, temp1, temp2, temp3;

    0 temp0 = x * x;
    1 temp1 = y * y;
    2 temp2 = temp1 + temp2;
    3 temp3 = temp1 * temp2;

    return(temp3);
}

```

Figure 10-10 A C function computes $(x^2 + y^2) \times y^2$.

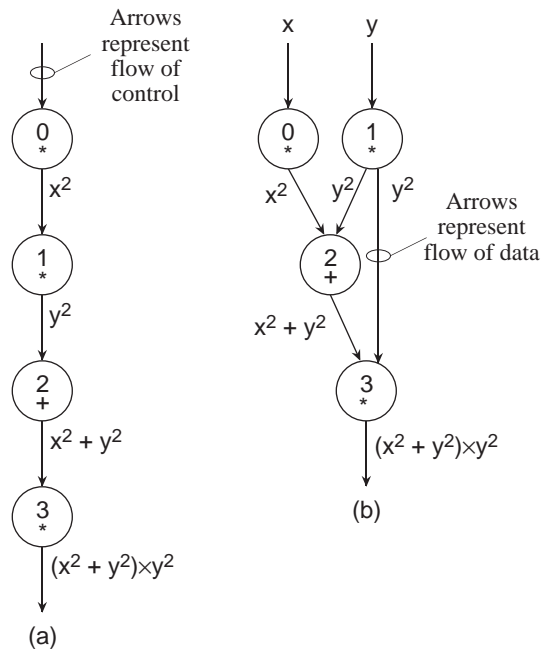


Figure 10-11 (a) Control sequence for C program; (b) dependency graph for C program.

neously (as long as there are two processors available.) The resulting speedup of

$$\frac{T_{\text{Sequential}}}{T_{\text{Parallel}}} = \frac{4}{3} = 1.\bar{3}$$

may not be very great, but for other programs, the opportunity for speedup can

be substantial as we will see.

Consider a matrix multiplication problem $\mathbf{Ax} = \mathbf{b}$ in which A is a 4×4 matrix and \mathbf{x} and \mathbf{b} are both 4×1 matrices, as illustrated in Figure 10-12a. Our goal is to

$$\begin{aligned}
 & \text{(a)} \quad \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \\
 & \quad b_0 = \overset{0}{a_{00}x_0} + \overset{4}{a_{01}x_1} + \overset{1}{a_{02}x_2} + \overset{6}{a_{03}x_3} \\
 & \quad b_1 = \overset{7}{a_{10}x_0} + \overset{11}{a_{11}x_1} + \overset{8}{a_{12}x_2} + \overset{13}{a_{13}x_3} \\
 & \text{(b)} \quad b_2 = \overset{14}{a_{20}x_0} + \overset{18}{a_{21}x_1} + \overset{15}{a_{22}x_2} + \overset{20}{a_{23}x_3} \\
 & \quad b_3 = \overset{21}{a_{30}x_0} + \overset{25}{a_{31}x_1} + \overset{22}{a_{32}x_2} + \overset{27}{a_{33}x_3}
 \end{aligned}$$

Figure 10-12 (a) Problem setup for $\mathbf{Ax} = \mathbf{b}$; (b) equations for computing the b_i .

solve for the b_i using the equations shown in Figure 10-12b. Every operation is assigned a number, starting from 0 and ending at 27. There are 28 operations, assuming that no operations can receive more than two operands. A program running on a SISD processor that computes the b_i requires 28 time steps to complete, if we make a simplifying assumption that additions and multiplications take the same amount of time.

A dependency graph for this problem is shown in Figure 10-13. The worst case path from any input to any output traverses three nodes, and so the entire process can be completed in three time steps, resulting in a speedup of

$$\frac{T_{\text{Sequential}}}{T_{\text{Parallel}}} = \frac{28}{3} = 9.\bar{3}$$

Now that we know the structure of the data dependencies, we can plan a mapping of the nodes of the dependency graph to PEs in a parallel processor. Figure

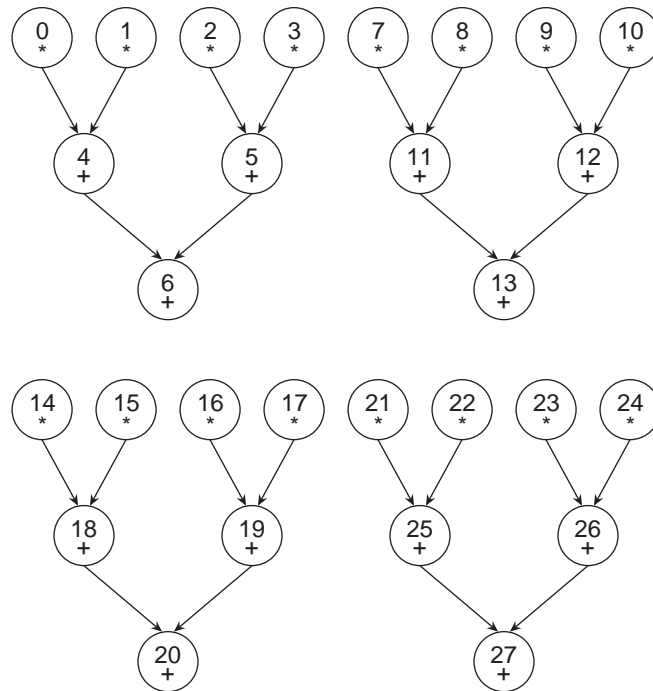


Figure 10-13 Dependency graph for matrix multiplication.

10-14a shows a mapping in which each node of the dependency graph for b_0 is

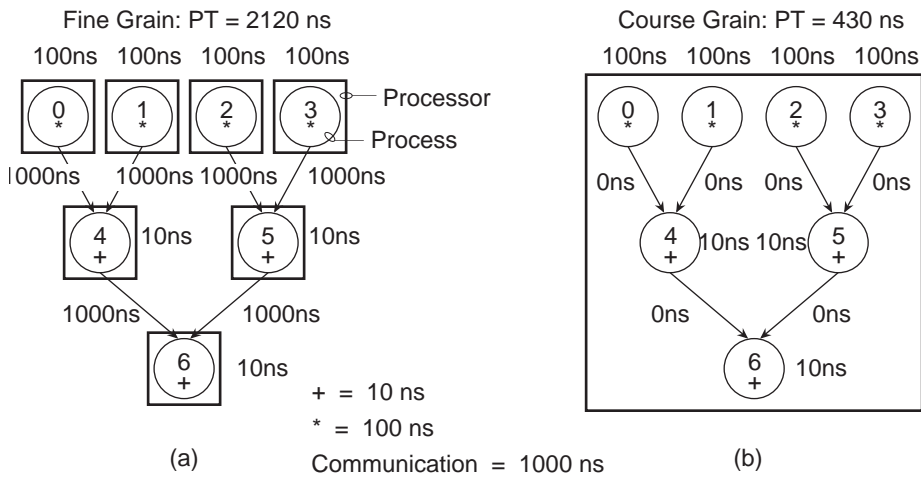


Figure 10-14 Mapping tasks to PEs: (a) one PE per operation; (b) one PE per b_i .

assigned to a unique PE. The time required to complete each addition is 10 ns,

the time to complete each multiplication is 100 ns, and the time to communicate between PEs is 1000 ns. These numbers are for a fictitious processor, but are not far off in relative magnitude from real numbers.

As we can see from the parallel time of 2120 ns to execute the program using the mapping shown in Figure 10-14a, the time spent in communication dominates performance. This is worse than a SISD approach, since the 16 multiplications and the 12 additions would require $16 \times 100 \text{ ns} + 12 \times 10 \text{ ns} = 1720 \text{ ns}$. There is no communication cost within a SISD processor, and so only the computation time is considered.

An alternative mapping is shown in Figure 10-14b in which all of the operations needed to compute b_0 are clustered onto the same PE. We have thus increased the **granularity** of the computation, which is a measure of the number of operations assigned to each PE. A single PE is a sequential, SISD processor, and so none of the operations within a cluster can be executed in parallel, but the communication time among the operations is reduced to 0. As shown in the diagram, the parallel time for b_0 is now 430 ns which is much better than either the previous parallel mapping or a straight SISD mapping. Since there are no dependencies among the b_i , they can all be computed in parallel, using one processor per b_i . The actual speedup is now:

$$\frac{T_{\text{Sequential}}}{T_{\text{Parallel}}} = \frac{1720}{430} = 4$$

Communication is always a bottleneck in parallel processing, and so it is important that we maintain a proper balance. We should not be led astray into thinking that adding processors to a problem will speed it up, when in fact, adding processors can increase execution time as a result of communication time. In general, we want to maintain a ratio in which:

$$\frac{T_{\text{Communication}}}{T_{\text{Computation}}} \leq 1$$

10.1.4 FINE-GRAIN PARALLELISM – THE CONNECTION MACHINE CM-1

The Connection Machine (CM-1) is a massively parallel SIMD processor designed and built by Thinking Machines Corporation during the 1980's. The architecture is noted for high connectivity between a large number of small processors. The CM-1 consists of a large number of one-bit processors arranged at the vertices of an n -space hypercube. Each processor communicates with other

processors via routers that send and receive messages along each dimension of the hypercube.

A block diagram of the CM-1 is shown in Figure 10-15. The host computer is a

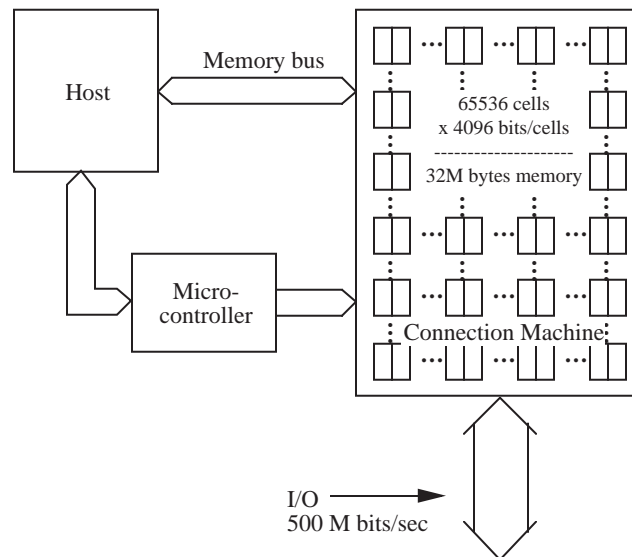


Figure 10-15 Block diagram of the CM-1 (Adapted from [Hillis, 1985]).

conventional SISD machine such as a Symbolics computer (which was popular at the time) that runs a program written in a high level language such as LISP or C. Parallelizeable parts of a high level program are farmed out to 2^n processors (2^{16} processors is the size of a full CM-1) via a memory bus (for data) and a microcontroller (for instructions) and the results are collected via the memory bus. A separate high bandwidth datapath is provided for input and output directly to and from the hypercube.

The CM-1 makes use of a 12-space hypercube between the routers that send and receive data packets. The overall CM-1 prototype uses a 16-space hypercube, and so the difference between the 12-space router hypercube and the 16-space PE hypercube is made up by a crossbar that serves the 16 PEs attached to each router. For the purpose of example, a four-space hypercube is shown in Figure 10-16 for the router network. Each vertex of the hypercube is a router with an attached group of 16 PEs, each of which has a unique binary address. The router hypercube shown in Figure 10-16 thus serves 256 PEs. Routers that are directly connected to other routers can be found by inverting any one of the four most

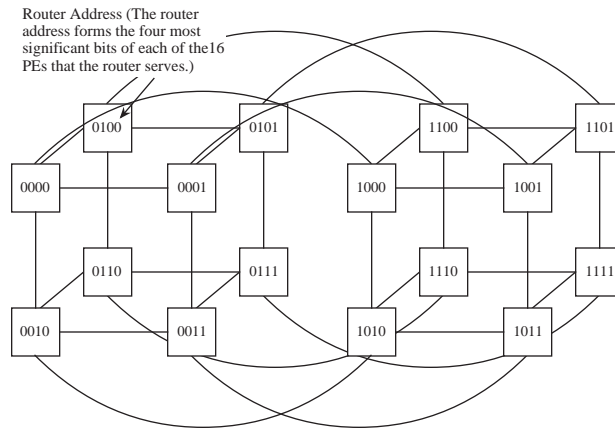


Figure 10-16 A four-space hypercube for the router network.

significant bits in the address.

Each PE is made up of a 16-bit flag register, a three-input, two-output ALU, and a 4096-bit random access memory, as shown in Figure 10-17. During operation,

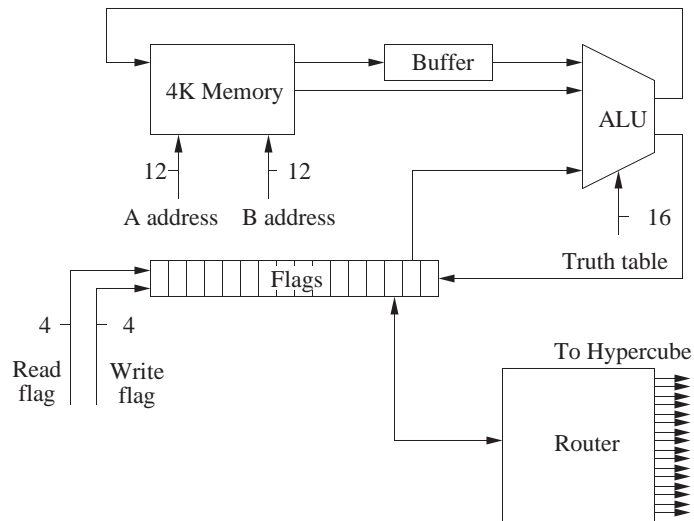


Figure 10-17 Block diagram of a CM-1 processing element

an external controller (the microcontroller of Figure 10-15) selects two bits from memory via the A address and B address lines. Only one value can be read from memory at a time, so the A value is buffered while the B value is fetched. The controller selects a flag to read, and feeds the flag and the A and B values into an

ALU whose function it also selects. The result of the computation produces a new value for the A addressed location and one of the flags.

The ALU takes three one-bit data inputs, two from the memory and one from the flag register, and 16 control inputs from the microcontroller and produces two one-bit data outputs for the memory and flag registers. The ALU generates all $2^3 = 8$ combinations (minterms) of the input variables for each of the two outputs. Eight of the 16 control lines select the minterms that are needed in the sum-of-products form of each output.

PE's communicate with other PE's through routers. Each router services communication between a PE and the network by receiving packets from the network intended for the attached PEs, injecting packets into the network, buffering when necessary, and forwarding messages that use the router as an intermediary to get to their destinations.

The CM-1 is a landmark machine for the massive parallelism made available by the architecture. For scalable problems like **finite element analysis** (such as modeling heat flow through the use of partial differential equations), the available parallelism can be fully exploited. There is usually a need for floating point manipulation for this case, and so floating point processors augment the PEs in the next generation CM-2. A natural way to model heat flow is through a mesh interconnect, which is implemented as a hardwired bypass to the message-passing routing mechanism through the North-East-West-South (NEWS) grid. Thus we can reduce the cost of PE-to-PE communication for this application.

Not all problems scale so well, and there is a general trend moving away from fine grain parallel processing. This is largely due to the difficulty of keeping the PEs busy doing useful work, while also keeping the time spent in computation greater than the time spent in communication. In the next section, we look at a coarse grain architecture: The CM-5.

10.1.5 COURSE-GRAIN PARALLELISM: THE CM-5

The CM-5 (Thinking Machines Corporation) combines properties of both SIMD and MIMD architectures, and thereby provides greater flexibility for mapping a parallel algorithm onto the architecture. The CM-5 is illustrated in Figure 10-18. There are three types of processors for data processing, control, and I/O. These processors are connected primarily by the Data Network and the Control Network, and to a lesser extent by the Diagnostic Network.

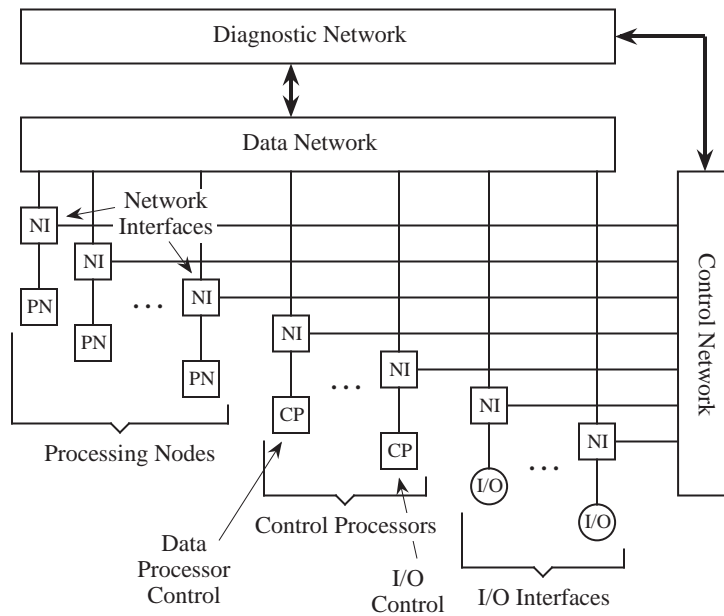


Figure 10-18 The CM-5 architecture.

The processing nodes are assigned to control processors, which form **partitions**, as illustrated in Figure 10-19. A partition contains a control processor, a number

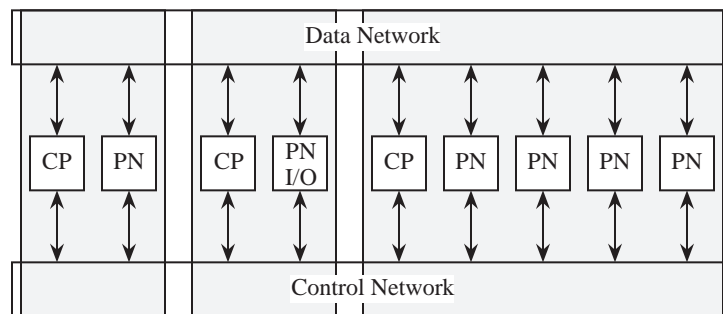


Figure 10-19 Partitions on the CM-5.

of processing nodes, and dedicated portions of the Control and Data Networks. Note that there are both user partitions (where the data processing takes place) and I/O partitions.

The Data Network uses a **fat-tree** topology, as illustrated in Figure 10-20. The general idea is that the bandwidth from child nodes to parent nodes increases as

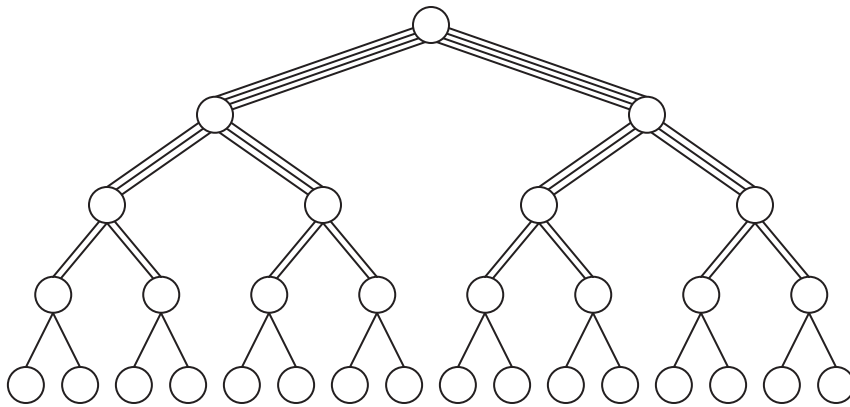


Figure 10-20 An example of a fat tree.

the network approaches the root, to account for the increased traffic as data travels from the leaves toward the root.

The Control Network uses a simple binary tree topology in which the system components are at the leaves. A control processor occupies one leaf in a partition, and the processing nodes are placed in the remaining nodes, although not necessarily filling all possible node positions in a subtree.

The Diagnostic Network is a separate binary tree in which one or more diagnostic processors are at the root. At the leaves are physical components, such as circuit boards and backplanes, rather than logical components such as processing nodes.

Each control processor is a self-contained system that is comparable in complexity to a workstation. A control processor contains a RISC microprocessor that serves as a CPU, a local memory, I/O that contains disks and Ethernet connections, and a CM-5 interface.

Each processing node is much smaller, and contains a SPARC-based microprocessor, a memory controller for 8, 16, or 32 Mbytes of local memory, and a network interface to the Control and Data Networks. In a full implementation of a CM-5, there can be up to 16,384 processing nodes, each performing 64-bit floating point and integer operations, operating at a clock rate of 32 MHz.

Overall, the CM-5 provides a true mix of SIMD and MIMD styles of processing, and offers greater applicability than the stricter SIMD style of the CM-1 and

CM-2 predecessors.

10.2 Case Study: Parallel Processing in the Sega Genesis

Home video game systems are good examples of (nearly) full-featured computer architectures. They have all of the basic features of modern computer architectures, and several advanced features. One notably lacking feature is permanent storage (like a hard disk) for saving information, although newer models even have that to a degree. One notable advanced feature, which we explore here, is the use of multiple processors in a MIMD configuration.

Three of the most prominent home video game platforms are manufactured by **Sony**, **Nintendo**, and **Sega**. For the purpose of this discussion, we will study the Sega Genesis, which exploits parallel processing for real-time performance.

10.2.1 THE SEGA GENESIS ARCHITECTURE

Figure 10-21 illustrates the external view of the Sega Genesis home video game

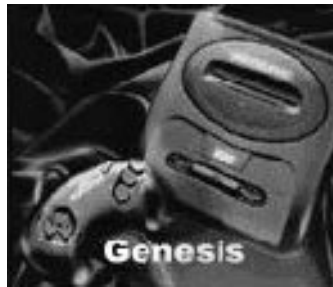


Figure 10-21 External view of the Sega Genesis home video game system.

system. The Sega Genesis consists of the motherboard, which contains all of the electronic components such as the processor, memory, and interconnects, and also a few hand-held controllers and an interface to a television set.

In terms of the conventional von Neumann model of a digital computer, the Sega Genesis has all of the basic parts: input (the controllers), output (the television set), arithmetic logic unit (inside of the processor), control unit (also inside of the processor), and memory (which includes the internal memory and the plug-in game cartridges).

The system bus model captures the logical connectivity of the Sega architecture as well as some of the physical organization. Figure 10-22 illustrates the system

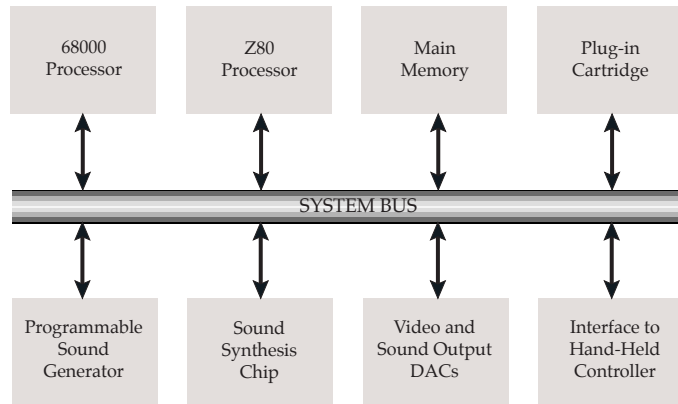


Figure 10-22 System bus model view of the Sega Genesis.

bus model view of the Sega Genesis. The Genesis contains two general-purpose microprocessors, the Motorola 68000 and the Zilog Z80. These processors are older, low cost processors that handle the general program execution. Video game systems must be able to generate a wide variety of sound effects, a process that is computationally intensive. In order to maintain game speed and quality during sound generation the Genesis off-loads sound effect computations to two special purpose chips, the Texas Instruments programmable sound generator (TI PSG) and the Yamaha sound synthesis chip. There are also I/O interfaces for the video system and the hand-held controls.

The 68000 processor runs the main program and controls the rest of the machine. The 68000 accomplishes this by transferring data and instructions to the other components via the system bus. One of the components that the 68000 processor controls is the architecturally similar, but smaller Z80 processor, which can be loaded with a program that executes while the 68000 returns to execute its own program, using an arbitration mechanism that allows both processors to share the bus (but only one at a time.)

The TI PSG has 3 square wave tones and 1 white noise tone. Each tone/noise can have its own frequency and volume.

The Yamaha synthesis chip is based on FM synthesis. There are 6 voices with 4 operators each. The chip is similar to those used in the Yamaha DX27 and

DX100 synthesizers. By setting up registers within the chips, a rich variety of sounds can be created.

The plug-in game cartridges contain the programs, and there is additional runtime memory available in a separate unit (labeled “Main memory” in Figure 10-22.) Additional components are provided for video output, sound output, and hand-held controllers.

10.2.2 SEGA GENESIS OPERATION

When the Sega Genesis is initially powered on, a RESET signal is enabled, which allows all of the electrical voltage levels to stabilize and initializes a number of runtime variables. The RESET signal is then automatically disabled, and the 68000 begins reading and executing instructions from the game cartridge.

During operation, the instructions in the game cartridge instruct the 68000 to load a program into the Z80 processor, and to start the Z80 program execution while the 68000 returns to its own program. The Z80 program controls the sound chips, while the 68000 carries out graphical operations, probes the hand-held controllers for activity, and runs the overall game program.

10.2.3 SEGA GENESIS PROGRAMMING

[Note from Authors: This section is adapted from a contribution by David Ashley, dash@xdr.com.]

The Sega Genesis is a home video game system that uses plug-in cartridges to store the game software. Blank cartridges can be purchased from third party vendors, which can then be programmed using an inexpensive **PROM burner** card that be plugged into the card cage of a desktop computer. Games can be written in high level languages and compiled into assembly language, or more commonly, programmed in assembly language directly (even today, assembly language is still heavily used for game programming). A suite of development tools translates the source code into object code that can then be burned directly into the cartridges (once per cartridge.) As an alternative to burning cartridges during development, the cartridge can be replaced with a reprogrammable development card.

The Genesis contains two general-purpose microprocessors, the Motorola 68000

and the Zilog Z80. The 68000 runs at 8 MHz and has 64 KB of memory devoted to it. The ROM cartridge appears at memory location 0. The 68000 off-loads sound effect computations to two special purpose chips, the Texas Instruments programmable sound generator (TI PSG) and the Yamaha sound synthesis chip.

The Genesis graphics hardware consists of 2 scrollable planes. Each plane is made up of tiles. Each tile is an 8×8 pixel square with 4 bits per pixel. Each pixel can thus have 16 colors. Each tile can use 1 of 4 color tables, so on the screen there can be 64 colors at once, but only 16 different colors can be in any specific tile. Tiles require 32 bytes. There are 64 KB of graphics memory, which allows for 2048 unique tiles if memory is used for nothing else.

Each plane can be scrolled independently in various ways. Planes consist of tables of words, in which each word describes a tile. A word contains 11 bits for identifying the tile, 2 bits for “flip x” and “flip y,” 2 bits for the selection of the color table, and 1 bit for a depth selector. Sprites are also composed of tiles. A sprite can be up to 4 tiles wide by four tiles high. Since the size of each tile is 8×8, this means sprites can be anywhere from 8×8 pixels to 32×32 pixels in size. There can be 80 sprites on the screen at one time. On a single scan line there can be 10 32-pixel wide sprites or 20 16-pixel wide sprites. Each sprite can only have 16 colors taken from the 4 different color tables. Colors are allocated 3 bits for each gun, and so 512 colors are possible. (Color 0=transparent.)

There is a memory copier program that is resident in hardware that performs fast copies from the 68000 RAM into the graphics RAM. The Z80 also has 8KB of RAM. The Z80 can access the graphics chip or the sound chips, but usually these chips are controlled by the 68000.

The process of creating a game cartridge involves (1) writing the game program, (2) translating the program into object code (compiling, assembling, and linking the code into an executable object module; some parts of the program may be written in a high level language, and other parts, directly in assembly language), (3) testing the program on a reprogrammable development card (if a reprogrammable development card is available), and (4) burning the program into a blank game cartridge.

See Further Reading below for more information on programming the Sega Genesis.

10.3 Network Architecture: The Internet

In the early days of computing, computers were centralized facilities that contained most or all of the resources used by the populations they serviced. Data was transferred between computers via media (punched paper cards, paper tapes, magnetic tapes, and magnetic disks), hand-carried by an operator.

As the number of computers increased, and costs shifted away from hardware and more toward labor, it became economical to directly link computers so that resources could be shared. This is what networking is about. We briefly explored local area networks in Chapter 8, in the context of the traditional 7-layer ISO model. Here, we take a deeper look at architectural aspects of computer networks in the context of the Internet model.

10.3.1 THE INTERNET MODEL

In a telecommunication system there may be many sources and many destinations. An example of this form of communication is a long distance telephone network. For every telephone to be reachable from every other telephone, there must be a path, or **channel**, between each source and destination. If there are 10^7 telephones in New York City and 10^7 telephones in Chicago, then for everyone in one city to be able to call everyone in the other city, $10^7 \times 10^7 = 10^{14}$ channels must exist between the cities. Fortunately, not everyone in New York City wants to talk with everyone in Chicago at the same time, and a smaller number of channels between New York City and Chicago can be shared among all telephones in those cities. On the other hand, there must be at least one line from each telephone to the telephone company's central office, and there must be sufficient lines between central offices to handle the maximum number of simultaneously held conversations.

A small number of physical connections, on the order of a few to a few thousand depending on whether fibers or wires are used, are all that are needed to connect the cities because it is never the case that everyone in one city wants to call someone in the other city at the same time. The information carrying capacity of the connections (called **bandwidth**) is shared among all of the users so that a dramatic reduction in cost is realized. A control mechanism must be created, however, so that the bandwidth can be shared properly.

Layering in the TCP/IP Protocol Suite

An “internet” is a collection of interconnected networks. The “Internet” is prob-

ably the most well-known internet, using the TCP/IP protocol and IP addresses in what is known as the TCP/IP protocol suite (more on this below). The 7-layer OSI model has been simplified somewhat in the Internet, which can be thought of as having only 4 layers, as illustrated by the protocol stack shown in Figure 10-23. At the bottom of the protocol stack is the Link layer, which is made up of

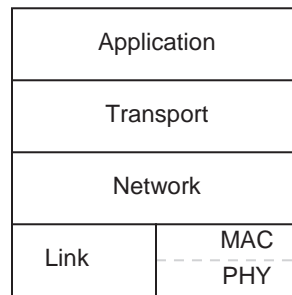


Figure 10-23 Internet protocol stack.

the medium access control (MAC) and physical (PHY) sublayers. The Link layer resolves contention for the medium when more than one device wants to transmit, manages the logical grouping of bits into frames, and implements error protection.

The Link layer is responsible for simply getting a frame of bits from one machine to a directly connected machine. This is fine for point-to-point communication between two cooperating processes on different machines. In order for multiple processes to share the same link, however, a protocol is needed to coordinate which data goes to what process. This is the responsibility of the Network layer, which is implemented with the Internet Protocol (IP) for the Internet.

The network layer deals with hop-by-hop communication. The Transport layer deals with end-to-end communication, in which there may be a number of intervening systems between the sender and receiver. The Transport layer deals with retransmission (for errors, or packets dropped due to congestion), sequencing (packets may arrive out-of-order, flow control (applying back-pressure to the source to relieve congestion) and error protection (the Link layer does not do enough error protection on its own.) For the Internet, the Transport layer is implemented with the Transmission Control Protocol (TCP). The TCP/IP combination at the Network and Transport layers is what defines the Internet. Any other protocols can be used at the Link and Application layers.

At the Application layer, a process can exchange data with another process anywhere on the Internet and treat the connection as if it is a file on the local system, reading and writing bytes with ordinary read and write system calls, frequently implemented by **sockets**, which are pathways to the network through the operating system.

Internet Addresses

Every interface on the Internet has a unique IP address. Version 4 of the IP protocol, known as IPv4, is still widely used but is gradually being replaced by IPv6 which uses addresses that are four times larger, and has several enhancements and simplifications to IPv4. An example of an IPv4 address, shown in “dotted decimal notation” is shown below:

165.230.140.67

Each number that is delimited by a dot is an unsigned byte in the range from 0 through 255. The equivalent bit pattern for the IPv4 address shown above is then:

10100101.11100110.10001100.10000011

The leftmost bits determine the class of the address. Figure 10-24 shows the five

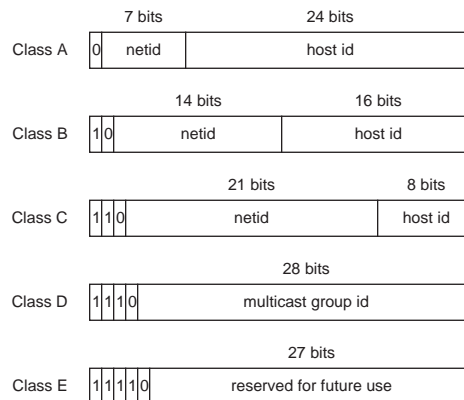


Figure 10-24 Five classes of IPv4 addresses.

IPv4 classes. Class A has 7 bits for the network identification (ID) and 24 bits for the host ID. There can thus be at most 2^7 class A networks and 2^{24} hosts on each class A network. A number of these addresses are reserved, and so the number of

addresses that can be assigned to hosts is fewer than the number of possible addresses.

Class B addresses use 14 bits for the network ID and 16 bits for the host ID. Class C addresses use 21 bits for the network ID and 8 bits for the host ID. Class D addresses are used for **multicast** groups, to which an end-system that has a class A, B, or C address subscribes, and thereby receives all network traffic intended for that group. This is an efficient mechanism for sending the same packets to multiple subscribers, without flooding the network with broadcasts, and without the sender needing to keep track of all of the current subscribers.

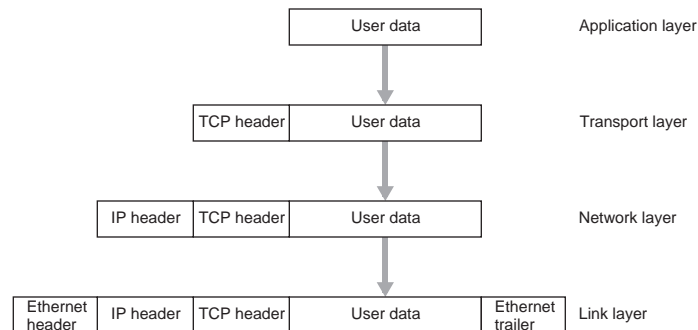


Figure 10-25 Encapsulation in the TCP/IP protocol suite.

adds a TCP header that identifies the source and destination ports, forming a TCP segment. The TCP segment is passed down to the network layer, where the TCP segment is repackaged into IP datagrams, each with an IP header identifying the source and destination systems. The IP datagrams are sent to the Link layer, where the datagrams are encapsulated into Ethernet frames (for this example). The reverse process takes place on the receiving system.

A single TCP segment may be decomposed into a number of IP datagrams, that are independently routed through the Internet. Each IP datagram contains the source and destination IP addresses (in the IP header), the source and destination ports (in the TCP header), and the protocol (in the IP header – TCP is only one of the transport layer protocols used in the Internet.) Collectively, these five parameters uniquely identify each IP datagram as it traverses the Internet, which helps ensure that the datagrams arrive at the correct receiving process.

The Domain Name System

The Domain Name Systems (DNS) is a distributed database that maps between hostnames and IP addresses, and provides mail routing information. For example, `cereal.rutgers.edu` maps to `165.230.140.67` (and vice versa), and all three names: `internet.rutgers.edu`, `www.internet.rutgers.edu`, and `mulder.rutgers.edu` map to `165.230.44.67`. The DNS is responsible for interacting with programs that need to map between names and addresses.

Each domain (like `rutgers.edu`) maintains its own database of information, and runs a server that other systems across the Internet can query. Access to the DNS is provided through a **resolver** which is embodied in library routines that are silently linked into high-level programs that access the network.

The Network Information Center (NIC, also known as the InterNIC) manages the top-level domains, and delegates authority for second level domains. Within a **zone**, a local administrator maintains the name server database. There must be a primary name server, which loads its database from a file, and secondary name servers, which get their information from the primary name server. Caching is used, so that a query that causes other servers to be contacted does not cause future queries to cause additional contacts to other servers.

The World Wide Web

The World Wide Web (or simply, the “Web”) is made up of client processes (Web browsers) and Web servers running the HyperText Transport Protocol (HTTP), at the Application layer of the Internet. As distinctions get blurred in everyday usage, it is important to keep in mind that the Web is built on top of the Internet – the Web is not the Internet itself.

In 1989, Tim Berners-Lee at CERN (the European high-energy physics facility) developed a text based Web, for exchanging technical documents among colleagues. In February 1993, the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign released a graphical version of the Mosaic Web browser, as well as an HTTP server, both free of charge, and the Web exploded to where it is today.

10.3.2 TRANSMISSION MEDIA

In a geographically close environment, computers can be networked with private cables in a number of configurations, such as the bus, star, ring, and mesh configurations, as shown in Figure 10-2. For geographically distant systems, the public switched telephone network (PSTN) can be used, which takes the form of an incomplete mesh.

Users connect to the PSTN with modems (see Chapter 8) that convert logical bits into audible sounds. People can hear at frequencies up to about 20 KHz, but only speak at frequencies of about 4 KHz, which is approximately the bandwidth that traditional telephony will pass on a voice-grade line. An analog signal (such as voice) that is approximated with a digital signal needs to be sampled at least twice per cycle (to capture the high and low values), and so a sampling rate of 8 KHz is needed to digitize a voice-grade line. At 8 bits per sample, that gives a bit rate of $8 \text{ bits/cycle} \times 8 \text{ KHz} = 64 \text{ Kbits/sec}$ which is what is available on an ordinary phone line. One sample out of every 8 is used by the telephone company to

administer the line, and so the maximum bit rate possible on a voice-grade line is 56 Kbits/sec.

A transmitted binary sequence is converted into high/low values, but the waveform gets attenuated and distorted, more so at high frequencies and long distances. Figure 10-26 illustrates the sampling problem. The binary pattern

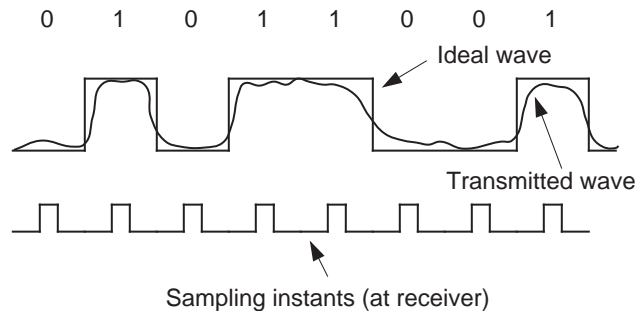


Figure 10-26 Ideal vs. transmitted waves.

01011001 is represented by an ideal wave, which is only approximated by a transmitted wave. The ideal wave contains discontinuities, which are difficult to produce with a real wave. In terms of analysis, we can think of the ideal wave as being approximated with a superposition of sinusoidal waves, with sharper edges achieved at higher frequencies.

Unfortunately, high frequencies are attenuated more greatly than low frequencies in most media, and different frequencies propagate at different rates, which leads to distortions of the wave as it propagates. The degree of distortion varies with the transmission medium, several of which are described here.

Two-Wire Open Lines

In one of the simplest scenarios, a pair of wires, open to free space, carries a signal and a return (the “ground”). The two-wire open line configuration is shown in Figure 10-27a. The lines emit electromagnetic radiation, and they also pick up noise, not necessarily the same amount of noise for each line, which distorts the difference signal. The lines are also vulnerable to “capacitive coupling” which means they pick up unwanted signals from neighboring wires. The speed and distance for reliable transmission is limited to about 19.2 Kbps and 50 m.

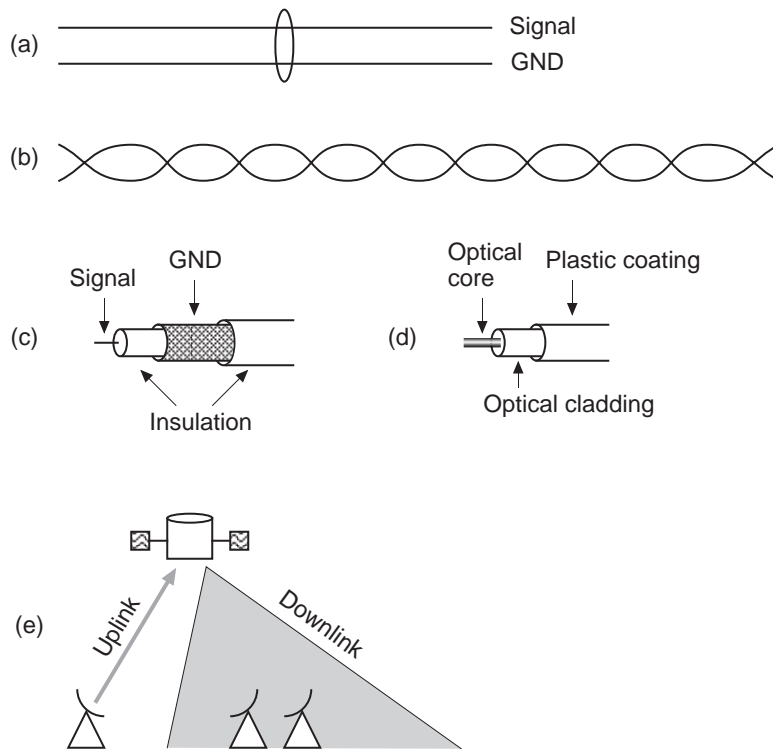


Figure 10-27 Transmission media. (a) Two-wire open lines; (b) twisted-pair lines; (c) coaxial cable; (d) optical fiber; (e) satellites.

Twisted-Pair Lines

If we twist the pair of lines in the two-wire open line configuration, then any spurious external noise that is introduced to the line affects both the signal and ground (reference) wires in the same way. Figure 10-27b shows the twisted-wire configuration. The difference signal is thus unaffected, and we can transmit up to 1 Mbps over 100 m.

Coaxial Cable

For higher speeds (10 Mbps) and longer distances (hundreds of meters), the signal wire is placed inside of the reference conductor (coaxially) with an insulator between the two, as shown in Figure 10-27c. The braiding of the outer conductor makes the cable more flexible. The idea is that the center conductor is effec-

tively shielded from external interference, and is also shielded from losses from electromagnetic radiation.

Optical Fiber

Optical communication is immune to electromagnetic interference and crosstalk, and supports a much wider bandwidth. There is a need for optoelectronic conversions on each end, which is commercially available up to a few Gbps (using laser diodes.) Optical fiber consists of the optical core, optical cladding, and a plastic coating as shown in Figure 10-27d.

A light emitting diode (LED) is less expensive light source than a laser diode, but it emits light at various angles, and so a **multimode stepped index** fiber is used that reflects light less than the critical angle back into the core. Because the path lengths differ, the received pulse is wider, and only modest bit rates can be supported. The LEDs are inexpensive, however, and bending tolerances are a less significant issue than for laser diodes.

With a **multimode graded index** fiber, light is refracted more greatly as it moves away from the core, which narrows the pulse and reduces losses.

Single mode (monomode) fiber reduces the core diameter to a single wavelength so that light travels along a single dispersionless path. Laser diodes are commonly used as sources for single mode fiber, and can operate up to several Gbps over tens of kilometers.

Satellites

Manmade satellites that are launched into orbit around the Earth are used for communication when a broad area of coverage is needed at a lower cost than a wireline network (including optical fibers). Natural satellites, inside and outside of Earth orbits (such as the Moon and asteroids), can also be used for communication, but are not generally in use for such purposes.

In satellite communication, a collimated microwave beam is transmitted from the ground to a satellite, where a transponder that covers a certain band of frequencies retransmits the signal to an area of coverage on the Earth. The satellite configuration is shown in Figure 10-27e.

A typical satellite has several transponders at 500 MHz per channel. A small area of coverage means that the transmitted signal is stronger and the receiving dishes can be smaller. This is typical for direct broadcast satellite (DBS) television, in which very small receiving dishes are used. The DBS satellites orbit the Earth at a low orbit, approximately 700 Km, and so a smaller collecting area is needed than for satellites that are placed in geosynchronous orbit (23,000 miles above the surface of the Earth), where the Earth's attractive gravitational force and the repelling centrifugal force are balanced, so that the satellite appears stationary over the ground when the orbit collocates with the Equator. This is why large satellite dishes are aimed in the direction of the Equator.

For two-way satellite network communication, the delay between the end-user and the satellite needs to be tolerable. The uplink to the satellite is generally slower than the downlink. This matches the typical mode of operation for an end-user on the Internet, since less than 10% of the network traffic goes from the end-user to the Internet, and over 90% of the network traffic goes from the Internet to the end-user. The speed of communication is limited by c (the speed of light in a vacuum) which is approximately 1 ns per foot, or 5 μ s per mile (5280 feet per mile). Over a distance of 23,000 miles, the free space delay is more than 100 ms to the satellite and another 100 ms back to the Earth, plus a processing delay. This is more than the acceptable average delay of 100 ms for a keystroke response. Low Earth orbit (LEO) is only at 700 Km, and introduces a much smaller delay, on the order of spanning the distance of a few states in the United States, and is therefore better suited for interactive networking.

Terrestrial Microwave

Ground based line-of-sight links are effective up to 50 Km, particularly for crossing difficult terrain, although they are prone to atmospheric disturbances, flocks of geese, *etc.*

Radio

In cellular radio communication, a radio base station is placed in the middle of **cell**, which is generally less than 20 Km in diameter. A restricted band of frequencies is used within a cell, for communication between roaming cellular devices and the base station. Neighboring cells use a different band of frequencies, so that there is no confusion at the cell boundary when a **handoff** is made as a roaming end-user transits from one cell to another, which normally involves a frequency change.

Total available bandwidth in a cell is small, on the order of 2 MB/s, which is subdivided over the number of channels in use. In congested areas, cell sizes are smaller than in less densely populated areas, sometimes extending no farther than a single building.

10.3.3 BRIDGES AND ROUTERS REVISITED, AND SWITCHES

A **hub** is a central connection point for end systems. A hub is also known as a **bridge** when an end system is another hub. A hub simply copies packets from one network interface to all of the others, as illustrated in the configuration shown in Figure 10-28a. Hubs and bridges have modest intelligence these days,

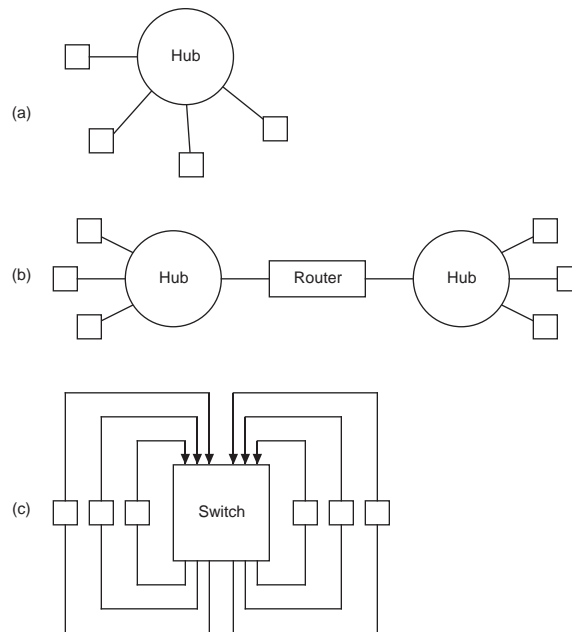


Figure 10-28 Configurations shown for (a) a hub; (b) a router; and (c) a switch.

by isolating collisions on single network links (that is, if two packets collide on a span of the network, which is a normal but unwanted condition, the collision signal is not propagated to the other network links), and by limiting certain types of traffic from being sent to all other interfaces.

A **router** connects one network to another (see Figure 10-28b), and makes decisions with respect to forwarding packets across its boundaries. A router by definition has more than one network interface and forwards packets between

interfaces. The network protocols used on either side of a router can differ.

A router forwards packets based on the protocol, whereas a **switch** forwards packets based only on the destination address. A switch is a high speed hub with no shared bandwidth, as illustrated in Figure 10-28c. A switch eliminates media access conflicts because there is no contention for the media.

We see an example of a switch in the 3-stage Clos network discussed in Section 10.1.2. This type of network requires an external controller that sets up the source-to-destination paths. An enhancement is a self-routing network, that sets up source-to-destination connections on-the-fly, based on the destination addresses in the headers of packets traversing the network.

As an example, consider designing a 4-input, 4-output self-routing switch. We can accomplish this using the **bubblesort** algorithm, in which packets with the smallest addresses are bubbled to the top, by making pairwise exchanges starting from the top and working toward the bottom, dropping the packet with the largest address to the bottom on each pass. For n channels, there are $n(n-1)/2$ comparisons that need to be made. For this case, $n=4$, and so $4(4-1)/2 = 6$ comparisons need to be made, which means that the switch needs 6 crosspoints.

The 4×4 self-routing switch is shown in Figure 10-29.

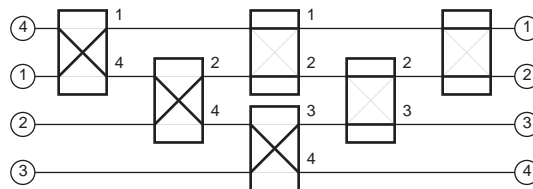


Figure 10-29 A 4×4 self-routing switch based on the bubblesort algorithm.

SUMMARY

Parallel architectures can be classified as MISD, SIMD, or MIMD. The MISD approach is used for systolic array processing, and is the least general architecture of the three. In a SIMD architecture, all PEs carry out the same operations on different data sets, in an “army of ants” approach to parallel processing. The MIMD approach can be characterized as “herd of elephants,” because there are a small

number of powerful processors, each with their own data and instruction streams.

The current trend is moving away from the fine grain parallelism that is exemplified by the MISD and SIMD approaches, and toward the MIMD approach. This trend is due to the high time cost of communicating among PEs, and the economy of using networks of workstations over tightly coupled parallel processors. The goal of the MIMD approach is to better balance the time spent in computation with the time spent in communication.

The Internet is based on the TCP/IP protocol suite. User data is encapsulated at the Application, Transport, Network, and Link layers, and is sent through the Internet and de-encapsulated (de-muxed) on the receiving system. As it is passed through the Internet, the data traverses various transmission media, that varies in bandwidth and distance capabilities.

■ FURTHER READING

(Quinn, 1987) and (Hwang, 1993) overview the field of parallel processing in terms of architectures and algorithms. (Flynn, 1972) covers the Flynn taxonomy of architectures. (Gerasoulis and Yang, 1991) argue for maintaining a ratio of communication time to computation time of less than 1. (Hillis, 1985) and (Hillis, 1993) describe the architectures of the CM-1 and CM-5, respectively. (Hui, 1990) covers interconnection networks, and (Leighton, 1992) covers routing algorithms for a few types of interconnection networks. (Wu and Feng, 1981) covers routing on a shuffle-exchange network. (Halsall, 1996) gives a thorough and readable treatment of network media types.

Additional information can be found on programming the Sega Genesis at <http://hiwaay.net/~jfrohwei/sega/genesis.html>.

Flynn, M. J., "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, vol. 30, no. 7, pp. 948-960, (1972).

Yang, T. and A. Gerasoulis, "A Fast Static Scheduling Algorithm for DAGs on an Unbounded Number of Processors," *Proceedings of Supercomputing '91*, Albuquerque, New Mexico, (Nov. 1991).

Halsall, F., *Data Communications, Computer Networks, and Open Systems*,

4/e, Addison-Wesley, (1996).

Hillis, W. D., *The Connection Machine*, The MIT Press, (1985).

Hillis, W. D. and L. W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," *Communications of the ACM*, vol. 36, no. 11, pp. 31-40, (Nov., 1993).

Hui, J. Y., *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, (1990).

Hwang, K., *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, (1993).

Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, (1992).

Quinn, M. J., *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, (1987).

Stone, H. S. and J. Cocke, "Computer Architecture in the 1990s," *IEEE Computer*, vol. 24, no. 9, pp. 30-38, (Sept., 1991).

Wu, C.-L. and T.-Y. Feng, "The Universality of the Shuffle-Exchange Network," *IEEE Transactions on Computers*, vol. C-30, no. 5, pp. 324-, (1981).

■ PROBLEMS

10.1 Create a dependency graph for the following expression:

$$f(x, y) = x^2 + 2xy + y^2.$$

10.2 Given 100 processors for a computation with 5% of the code that cannot be parallelized, compute speedup and efficiency.

10.3 What is the diameter of a 16-space hypercube?

10.4 For the EXAMPLE in Section 10.1.2, compute the total crosspoint complexity over all three stages.

- 10.5** To which IPv4 class does address 165.230.140.67 belong?
- 10.6** How many networks (not hosts) can the IPv4 class A, B, and C addresses support? That is, how many distinct class A, B, and C network addresses can there be? Do not consider reserved addresses.
- 10.7** Network media always carries data in bit-serial fashion, and never in parallel. That is not to say that data could not be carried over a network in byte-parallel or word-parallel fashion; there simply is no advantage to doing it this way. To see why this is the case, calculate the time required to transmit a 32-bit word between two computers over a 32-foot network. The network speed is 1 Gbps per channel. The delay imposed by the distance is 1 ns per foot. Calculate the time to transmit the 32-bit word using a single channel (bit-serial fashion) and using 32 channels (word-parallel fashion).