CS411
Parallel Processor Organization

As described earlier, the interconnection architecture among processors in a parallel machine has a large impact on how algorithms are implemented and also on how fast they will run. There are a couple of factors to consider:

- Degree. The degree of a processor is the number of communication links attached to it. It is desirable to have low degree from a cost standpoint and also from a scalability standpoint.

- Diameter. The diameter of a network is the maximum of the minimum distance between any pair of processors. That is, it is the longest path between any two processors, assuming that a shortest path is always chosen. A low diameter is desirable to allow efficient communication.

- Bisection width. The bisection width is the minimum number of wires that need to be removed in order to disconnect the network into two approximately equal sized subnetworks. A high bisection width implies higher bandwidth if there is a lot of traffic, but is more costly to build.

Linear Array

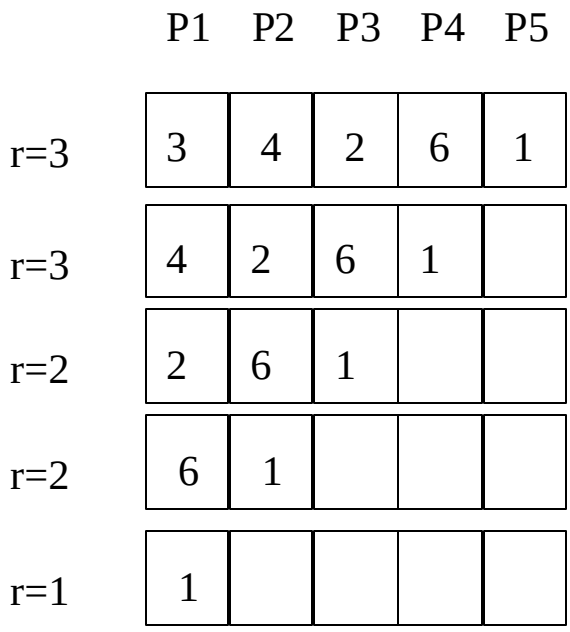
Perhaps the simplest architecture we could have is a string of n processors, P[1] to P[n], where each processor is connected to its two neighbors. P[I] is connected to P[I-1] and P[I+1]. P[1] and P[n] are only connected to one neighbor.

P[1] ←→ P[2] ←→ P[3] ←→ P[4] ←→ …. P[n]

The degree of this configuration is 2. The diameter is n-2 or O(n) if P[1] has a message to send to P[n]. Note however, that the minimum time for a message to travel is n/2, if P[1] and P[n] both sent their messages to P[n/2] who matches the information together. However, this is still O(n) communication time.

Although this architecture might seem rather limited, a related architecture, the ring, is sometimes employed (the Cray SV1 uses a type of ring). The ring has the same basic properties as the linear array, except the diameter is halved. Additionally, the linear array is a sub-component of other architectures we will see soon.

Let's look at a few simple problems we can apply to the linear array. First, consider the problem of finding the minimum. Assume that we have an array X[1..n] where X[I] is stored on P[I]. To determine the minimum, one method is for all data to march to the left in lockstep fashion. P[1] on the left sets running-min = min(running-min, x[I+1]). After n-1 steps, the minimum of X is stored in P[1].



This algorithm obviously takes O(n) time, no improvement over a sequential processor.

Now consider a related problem, where initially each processor contains no data. However, input is given one element at a time on the left to P[1]. Given n input numbers, we would like the data to be sorted where P[1] has the minimum, P[2] has the second smallest value, etc.

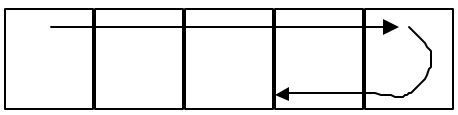
We can solve this problem by having each processor pass numbers to the right. However, each processor *keeps* the minimum data item that it has seen. That is, processor P[2] would store the minimum of all items with the exception of the smallest item, because P[1] would not pass it along. At the end of the algorithm, each processor has the values in ascending order:

	P1	P2	P3	P4	P5
3,2,4,1,5					
3,2,4,1	5				
3,2,4	1,5				
3,2	1,4	5			
3	1,2	4,5			
	1,3	4,2	5		
	1	2,3	4,5		
	1	2	4,3	5	
	1	2	3	4,5	
	1	2	3	4	5

The runtime for this 2n, which is O(n). This essentially gives us an O(n) sorting algorithm.

Finally, consider the case where each processor has some data that it wants to send to every other processor. We can use the tractor-tread algorithm, where each processor sends data to its neighbor. The end processors 'bounce' data back in the other direction. In O(n) steps, each processor can get data from every other processor.

P1 P2 P3 P4 P5

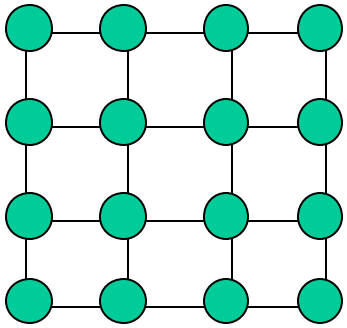


Once again, note that the linear array architecture is basically identical to a ring architecture, except we now cut the diameter in half with the option of sending data the opposite direction.

Mesh Architecture

A mesh architecture is quite common. It is implemented in many parallel machines, such as the Connection Machine, because it is relatively simple, has better attributes than the linear array, and is scalable.

The mesh is essentially a 2-dimensional, checkerboard arrangement. Some meshes use hexagonal neighbors or a vertical component in addition to the North , South, East, West arrangement. Processors can only communicate with their neighbors.



The square mesh is referred to as a mesh of size N. Note that there are $n^{1/2}$ rows and columns. The communication diameter is then $2n^{1/2}$-2 if we travel from the bottom left corner to the upper right corner. This is just $O(n^{1/2})$. This is an improvement over the linear array.

Let's look at some simple algorithms on the mesh.

Broadcast: Broadcast the data value x, initially stored in processor P[I,J] where I=row and J=column, to all processors in the mesh.

1. Rotate the value to all processors in P's row in $O(n^{1/2})$ time.
2. Each processor in P's row now rotates the value to each processor in its column in $O(n^{1/2})$ time.

The overall runtime is then $O(2n^{1/2})$ or just $O(n^{1/2})$.

Minimum: Each processor has a value, and we want the minimum value contained in all processors to be stored in processor P[1,1].

1. Each processor performs the linear array minimum algorithm within its column, with the minimum being stored in P[1, _].
2. Each processor in row 1 now has the minimum of each processor in their column. The processors in row 1 now perform the linear array minimum algorithm within row 1. The absolute minimum is now in P[1,1].

The overall runtime is also $O(2n^{1/2})$ or just $O(n^{1/2})$.

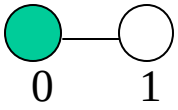
In general, we can perform the linear array algorithm on the mesh twice, once for the columns and once for the rows.

Hypercube Architecture

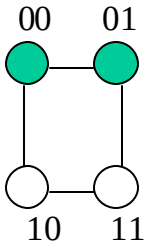
Let's look at one final architecture, the hypercube. This arrangement is used in the nCube, Paragon, CM I-II, Mark II-III, and several other machines. It is an attractive topology because it provides a low communication diameter and a high bisection width. The communication diameter is logarithmic in the number of processors, which allows for many fast operations. The main disadvantage is that it becomes expensive to scale up to a large number of processors.

Formally, a hypercube of size n consists of n processors indexed by the integers {0, 1, … n-1} where n is an integral power of 2. Processors A and B are connected iff their nique $\log_2$ n-bit strings differ in exactly one position.

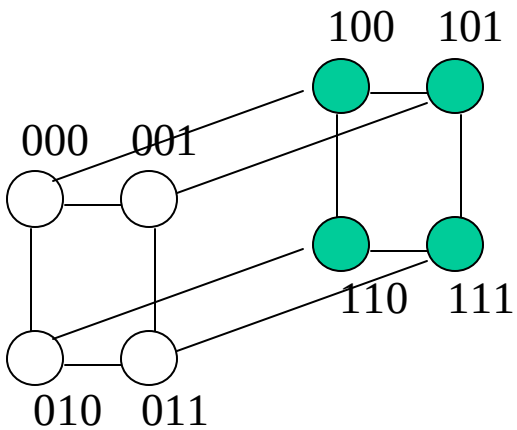
The easiest way to construct a hypercube is in a recursive fashion. First, start with a 1-cube:



To connect a 2-cube, we connect two 1-cubes. We connect the duplicate 1-cube by their corresponding nodes:



Next, we construct a 3-cube by connection two 2-cubes:



Based on this construction scheme, note that the number of communication links affiliated with each processor must increase as the size of the network increases. So unlike the mesh and linear array, this is a variable-degree network. The degree of a hypercube of size n is $\log_2 n$. In similar fashion, the communication diameter is also O(lgn). This is because each processor number differs by one bit. To send a message to the most distant processors, we would have to send it to an intermediate processor that differs by one bit. For example P0 to P7 traverses 000 → 001 → 011 → 111

This is an appealing property of the hypercube, because it has the promise of avoiding some of the communication bottlenecks that occur with the other architectures.

As an example algorithm, let's consider again the problem of finding the minimum in a hypercube of n=16. Each processor contains a single value, and we want the min of all these values.

In the first step, we send entries from all processors with a 1 in the most significant bit to their neighbors that have a 0 in the most significant bit. The processors that receive information compute the minimum of the received value and their element and store this result as a running minimum. In the next step, we send running minima from all processors with a 1 in their next most significant bit that received data from the previous step, to their neighbors with a 0 in that bit position. These processors also compute the running minima. The process continues until processor 0001 sends to processor 0000 which computes the final result. This requires a total of O(lgn) steps.

We can use the same process if we want to broadcast the result to all processors. The first processor sends data to its neighbor on the least significant bit, then both processors send data to their neighbors using the next significant bit, and so on, in another total of O(lgn) steps to broadcast the data to all processors.

Other Architectures

There are many other architectures out there. One is the pyramid architecture, which is essentially a tree on top of a mesh. This has the benefit of performing tree-like operations in lg time using the hierarchical portion, and mesh-like operations using the base of the pyramid.

Another scheme is the torus. A torus is formed by starting with a 2D mesh. If the leftmost and rightmost processors in a row are connected, and the topmost and bottommost processors are also connected, then this is called a torus architecture. This is even better connected than the mesh, but not quite as well connected as the hypercube. However, it is easier to scale up than a hypercube. The Intel Paragon used a 2D torus, and the Cray T3D is a 3D torus, hence its name. The Arctic Region Supercomputing Center has a Cray T3D with 272 processors and 69.6 GB of distributed memory. The memory is distributed with 256 MB of memory at each processor but is globally addressable.