RANDOMIZATION IN COMPUTING

CSE 6512

LECTURE 16, 20th Oct 2011.

NOTES BY UJWAL POTLURI

Parallel algorithms:

A parallel algorithm is an algorithm which is executed using more than one machine at a time.

To solve a problem using a parallel algorithm, the problem is divided into smaller sub-problems each of which is executed on different machines.

Let S be the best sequential run time to solve a problem.

Let T be the parallel run time using P processors.

FACT:

$T \geq S/P$

This can be proven by contradiction. Assume that $T<S/P$. We can simulate each step of the parallel algorithm using a single processor in $\leq P$ steps. Thus the entire parallel algorithm can be simulated in $\leq PT$ steps. If $PT < S$, we have a sequential algorithm whose run time is $< S$, which is a contradiction. □

In sequential computing, the Random Access Machine (RAM) is a universally accepted model of computing. In a RAM, any basic operation takes one unit of time. The time and space complexities are expressed as functions of the input size. For instance, the time complexity is the total number of basic operations performed by the algorithm.

When it comes to parallel computing, there are numerous models of computing. These models differ in how inter-processor communications are enabled. These models are broadly categorized into two:

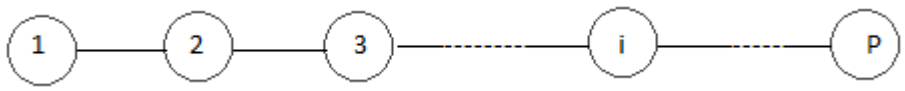
- Fixed connection machines
- Parallel Random Access Machines (PRAMs)

Fixed Connection Machines:

Definition: a fixed connection machine is a directed graph G(V,E) where V corresponds to the processors and E corresponds to the communication links between them.

Example:

Linear Array:



Degree = 2.

Diameter = P-1.

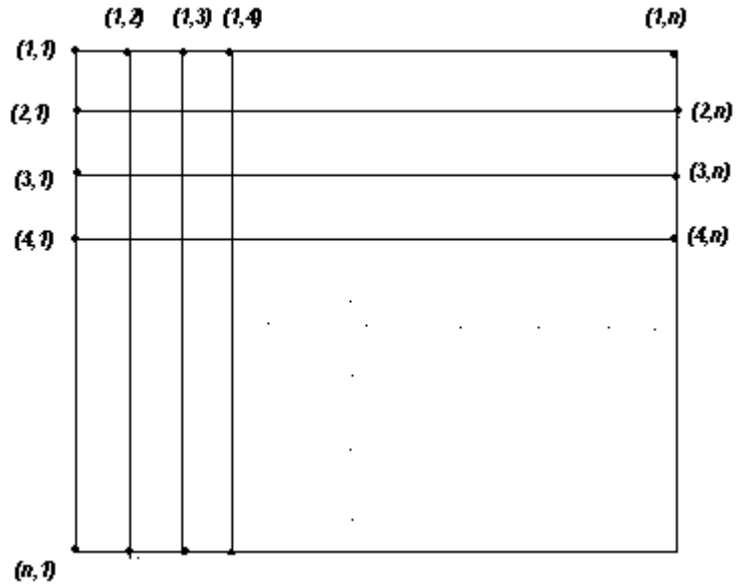
Communication between adjacent nodes is done in one unit of time.

Communication between any two nodes is done using a path (preferably the shortest path) between the two.

There are 3 crucial parameters in any fixed connection machine.

- **Degree**: is defined as the largest degree of any node. It should be as small as possible.
- **Diameter**: is defined to be the maximum of the shortest path between any two nodes i and j. It should be as small as possible.
- **Bisection width**: is defined to be the minimum size of any cut that partitions the network into two equal halves.

A MESH:



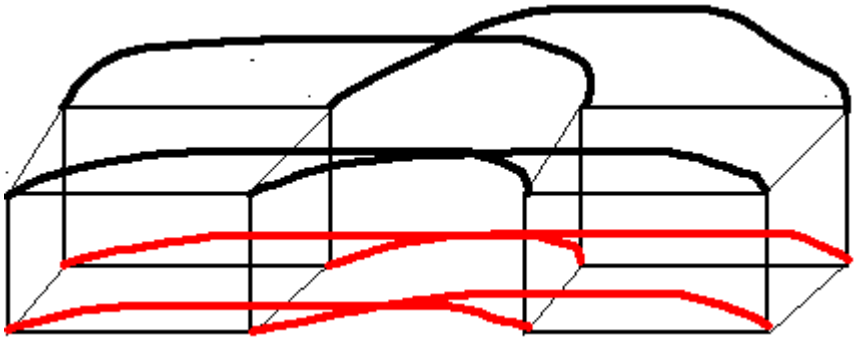
Degree = 4.

Diameter = 2(n-1).

The diameter is proportional to the square root of the number of processors.

Lower bound for sorting is equal to the diameter i.e., 2(n-1).

Hypercube:

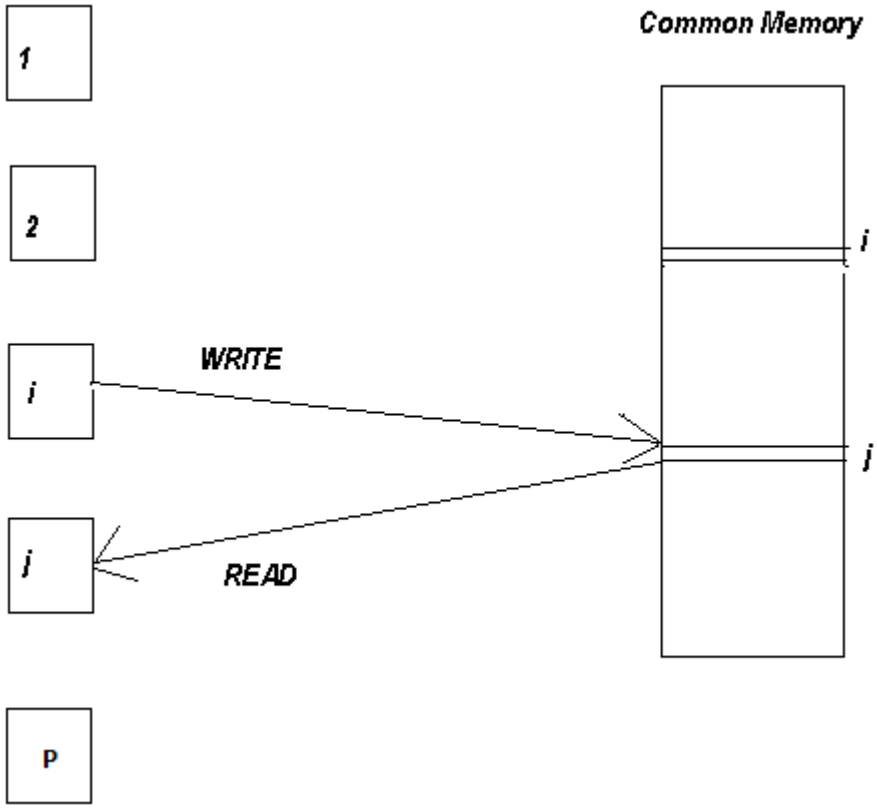


A hypercube of dimension n has 2^n nodes.

Diameter = n.

Degree = n.

PRAMs:



The PRAMs communicate with each other by writing into and reading from the common memory.

It takes two steps to communicate- one for writing and another for reading.

But it is not scalable in practice.

A PRAM is a simple and nice model. Often, the ideas developed for PRAMs can be applied to the other models as well. Depending on how read and write conflicts are resolved, we can conceive of different versions of the PRAM:

- EREW (Exclusive Read Exclusive Write)
- CREW (Concurrent Read Exclusive Write)
- CRCW (Concurrent Read Concurrent Write)

We need a special mechanism to resolve write conflicts since processors that are trying to write in the same cell at the same time might have different data to write. Some possible write conflict resolution mechanisms are:

- **COMMON**: Concurrent writes are permitted only if all the conflicting processors have the same message to write.
- **ARBITRARY**: In this model when there is a write conflict, an arbitrary one of the conflicting processors gets to write.
- **PRIORITY**: The processors are statically assigned priorities at the beginning. The write conflicts are resolved using the priorities.

The following is a hierarchy of the PRAMs in increasing order of computational power: EREW PRAM, CREW PRAM, Common CRCW PRAM, Arbitrary CRCW PRAM, and Priority CRCW PRAM. An algorithm designed for any version of the PRAM can be run on a more powerful PRAM model without any degradation in either processor bound or time bound. For example, an algorithm that runs on the EREW PRAM can be run on the CREW PRAM without increasing the number of processors or the run time.

Definition:

The work done by a parallel algorithm = PT.

Definition:

A parallel algorithm is asymptotically optimal if

$$PT = O(S).$$

The above equality is a sufficient condition for a parallel algorithm to be asymptotically optimal. For asymptotic optimality this condition may not be necessary.

Example:

INPUT: n bits $a_0, a_1, a_2, \ldots, a_n$

OUTPUT: $a_0 \wedge a_1 \wedge a_2 \wedge \ldots \wedge a_n$

FACT:

The above problem can be solved in constant time given n COMMON CRCW PRAM processors.

Step 1: Processor 1 sets: Result := 1;

Step 2: **for** i := 1 to n **in parallel do**

Processor i tries to write a 0 in Result **if** $a_i = 0$;

TOTAL TIME = O(1).

Work Done = O(n).

For this problem, S = n-1.

⇒ The algorithm is asymptotically optimal.

FACT:

We can find the maximum of n arbitrary elements in O(1) time using n^2 COMMON CRCW PRAM processors.

Proof:

Let the processors be $P_{11}, P_{12}, \ldots, P_{1n}, \ldots, P_{2n}, \ldots, P_{nn}$

Form n groups $G_1, G_2, \ldots, G_n$ with n processors in each group.

Let $X = k_1, k_2, \ldots, k_n$ be the input

1) **for** $1 \leq i, j \leq n$ **in parallel do**
   Processor P_{ij} computes
   b_{ij} := "Is $k_i \geq k_j$? "
2) **for** $1 \leq i \leq n$ **in parallel do**

Processors in G_i compute

$Q_i = b_{i1} \wedge b_{i2} \wedge \ldots \wedge b_{in}$ using the previous algorithm;

3) **for** $1 \leq i \leq n$ **in parallel do**
if $Q_i = 1$ **then** P_{i1} writes k_i in Result;

TOTAL RUN TIME = O(1).

Work Done = $O(n^2)$.

S = O(n).

⇒ The algorithm is not asymptotically optimal!