MODELS OF COMPUTATION
(Chapter 2)

- Models
  - An abstract description of a real world entity
  - Attempts to capture the essential features while suppressing the less important details.
  - Important to have a model that is both precise and as simple as possible to support theoretical studies of the entity modeled.
  - If experiments or theoretical studies show the model does not capture the important aspects of the physical entity closely, then the model should be refined.
  - Many engineers will not accept an abstract model of entity being studied, but insist on a detailed model.
    - Often reject a model if it does not capture the lower level details of the physical entity.
- Model of Computation
  - Describes a class of computers
  - Allows algorithms to be written for a general model rather than for a specific computer.
  - Allows the advantages of various models to be studied and compared.
  - Important, since the life-time of specific computers is quite short (e.g., 10 years).

Some Additional Networks

- **References:** [2,Akl, Ch 2], [3, Quinn, Ch 2-3]
- Shuffle Exchange
  - Let n be a power of 2 and $P_0, P_1, \ldots, P_{n-1}$ denote the processors.
  - A *perfect-shuffle* connection is a one-way communication link that exists from
    - P_i to P_{2i} if $i < n/2$ and
    - P_i to P_{2i+1-n} if $i \geq n/2$
  - Alternately, a perfect-shuffle connection exists between P_i and P_k if a left one-digit circular rotation of i, expressed in binary, produces k.
  - Its name is due to fact that if a deck of cards were "shuffled perfectly", the shuffle link of i gives the final shuffled position of card i
    - Example: See Figure 2.15.
  - An *exchange* connection link is a two way link that exists between P_i and P_{i+1} when i is even.
  - Figure 2.14 illustrates the shuffle & exchange links for 8 processors.
  - The reverse of a perfect shuffle link is called an *unshuffle* link.
  - A network with the shuffle, unshuffle, and exchange connections is called a *shuffle-exchange network*.

- Cube-Connected Cycles (or CCC)
  - A problem with the hypercube network is the large number of links each processor must support when q is large.
  - The CCC solves this problem by replacing each node of the q-dimensional hypercube with a ring of q processors, each connected to 3 PEs:
    - its two neighbors in the ring
    - one processor in the ring of a neighboring hypercube node.
  - Example: See Figure 2.18 in [2]

- **Network Metrics:** Recall Metrics for comparing network topologies
  - Degree
    - The *degree of network* is the maximum number of links incident on any processor.
    - Each link uses a *port* on the processor, so the most economical network has the lowest degree
  - Diameter
    - The *distance* between two processors P and Q is the number of links on the shortest path from P to Q.

Comparison of Network Topologies (cont)

  - The *diameter of a network* is the maximum distance between pairs of processors.
  - The *bisection width of a network* is the minimum number of edges that must be cut to divide the network into two halves (within one).
- Table 2.21in [2] (reproduced below) compares the topologies of the networks we have discussed.

  - See Table 3-1 of Quinn for additional details.

Topology	Degree	Diameter	Bis. W.
Linear Array	2	O(n)	1
Mesh	4	$O(\sqrt{n})$	n
Tree	3	O(lg n)	1
Shuffle-Exchange	3	O(lg n)	$\sqrt{n}$
Hypercube	O(lg n)	O(lg n)	$2^{d-1}\sqrt{n}$
Cube-Con. Cycles	3	O(lg n)	2^{d-1}

PRAM: Parallel Random Access Machine

- **References:**[2, Ch 2], [3, Ch 2], and [7, Ch 30]
  - **[7]** "Intro to Algorithm", Cormen, et.al., 1990

- The RAM Model (Random Access Machine)
  - A *memory* with M locations. Size of M is as large as needed.
  - A *processor* operating under the control of a sequential program. It can
    - load data from memory
    - store date into memory
    - execute arithmetic & logical computations on data.
  - A *memory access unit (MAU)* that creates a path from the processor to an arbitrary memory location.
- Sequential Algorithm Steps
  - A *READ* phase in which the processor reads datum from a memory location and copies it into a register.
  - A *COMPUTE* phase in which a processor performs a basic operation on data from one or two of its registers.
  - A *WRITE* phase in which the processor copies the contents of an internal register into a memory location.

- PRAM Model Description
  - Let $P_1, P_2, \ldots, P_n$ be identical processors
  - Assume these processors have a common memory with M memory locations with $M \geq N$.
  - Each P_i has a MAU that allows it to access each of the M memory locations.
  - A processor P_i sends data to a processor P_k by storing it in a memory location that P_k can read at a later time.
  - The model allows each processor to have its own algorithm and to run asynchronously.
  - In many applications, all processors run the same algorithm synchronously.
    - Restricted model called *synchronous PRAM*
    - Algorithm steps have 3 or less phases
      - READ Phase: Up to n processors read up to n memory locations simultaneously.
      - COMPUTE Phase: Up to n processors perform basic arithmetic/logical operations on their local data.
      - WRITE phase: Up to n processors write simultaneously into up to n memory locations.

  - Each processor knows its own ID and algorithms can use processor IDs to control the actions of the processors. (True for all models.)
- PRAM Memory Access Methods
  - *Exclusive Read (ER):* Two or more processors can not simultaneously read the same memory location.
  - *Concurrent Read (CR):* Any number of processors can read the same memory location simultaneously.
  - *Exclusive Write (EW):* Two or more processors can not write to the same memory location simultaneously.
  - *Concurrent Write (CW):* Any number of processors can write to the same memory location simultaneously.
- Variants of Concurrent Write:
  - *Priority CW*: The processor with the highest priority writes its value into a memory location.
  - *Common CW:* Processors writing to a common memory location succeed only if they write the same value.
  - *Arbitrary CW:* When more than one value is written to the same location, any one of these values (e.g., one with lowest processor ID) is stored in memory

  - *Random CW:* One of the processors is selected by some random process to write its value into memory.
  - *Combining CW:* The values of all the processors trying to write to a memory location are combined into a single value and stored into the memory location.
    - Some possible functions for combining numerical values are SUM, PRODUCT, MAXIMUM, MINIMUM.
    - Some possible functions for combining boolean values are AND, INCLUSIVE-OR, EXCLUSIVE-OR, etc.

The RAM Model

- A *memory* with M locations. Size of M is as large as needed.
- A *processor* operating under the control of a sequential program. It can
  - load data from memory
  - store date into memory
  - execute arithmetic & logical computations on data.
- A *memory access unit (MAU)* that creates a path from the processor to an arbitrary memory location.
- Sequential Algorithm Steps
  - A *READ* phase in which the processor reads datum from a memory location and copies it into a register.
  - A

PRAM ALGORITHMS

- Reference: Chapter 4 of [2, Akl], Chapter 30 of [7, identified below], and Chapter 2 of [3, Quinn]
  - **[7]** "Introduction to Algorithms" by Cormen, Leisterson, and Rivest, First (older) edition, 1990, McGraw Hill and MIT Press.

- Prefix computation application considered first
- EREW PRAM Model is assumed.
- A *binary operation* on a set S is a function

$$\oplus: S \times S \rightarrow S.$$

- Traditionally, the element $\oplus(s_1, s_2)$ is denoted as

$$s_1 \oplus s_1.$$

- The binary operations considered for prefix computations will be assumed to be
  - *associative*: $(s_1 \oplus s_2) \oplus s_3 = s_1 \oplus (s_2 \oplus s_3)$
- Examples
  - Numbers: addition, multiplication, max, min.
  - Strings: concatentation for strings
  - Logical Operations: **and**, **or**, **xor**
- Note: $\oplus$ is not required to be commutative.
- Prefix Operations: Assume $s_0, s_1, \ldots, s_{n-1}$ are in S. The computation of $p_0, p_1, \ldots, p_{n-1}$ defined below is called prefix computation:

$$p_0 = s_0$$
$$p_1 = s_0 \oplus s_1$$
$$\vdots$$
$$p_{n-1} = s_0 \oplus s_1 \oplus \ldots \oplus s_{n-1}$$

- *Suffix* computation is similar, but proceeds from right to left.
- A binary operation is assumed to take constant time, unless stated otherwise.
- The number of steps to compute p_{n-1} has a lower bound of $\Omega(n)$ since n-1 operations are required.
- Previous prefix sum examples in reference [2]:
  - Example 1.6 solves the prefix sum problem using the combinational circuit in Figure 1.4.
  - Example 2.1 gives the usual RAM algorithm.
  - Example 2.5 solves the prefix sum problem using a hypercube, as shown in Figure 2.21.
- Prefix Computation on PRAM can simulate both
  - the hypercube prefix operation algorithm
  - the combinational circuit computation.

  with the same $O(\lg n)$ running time.
- Discuss visual algorithm in Figure 4.1 (for n=8)
  - Same algorithm as given for hypercube and combinational circuit earlier.

- **EREW PRAM Version**: Assume PRAM has n processors, $P_0, P_1, \ldots, P_{n-1}$, and n is a power of 2. Initially, P_i stores x_i in shared memory location s_i for i = 0,1, ... , n-1.


```
for j = 0 to (lg n) -1, do
    for i = 2^j to n-1 do
        h = i - 2^j
        s_i = s_h ⊕ s_i
    endfor
endfor
```

- **Analysis:**
  - Running time is $t(n) = O(\lg n)$
  - Cost is $c(n) = p(n) \times t(n) = O(n \lg n)$
  - Note not cost optimal, as RAM takes $O(n)$
- **Cost-Optimal EREW PRAM Prefix Algorithm**
  - In order to make the above algorithm optimal, we must reduce the cost by a factor of lg n.
  - In this case, it is easier to reduce the nr of processors by a factor of lg n.
  - Let $k = \lceil \lg n \rceil$ and $m = \lceil n/k \rceil$
  - The input sequence $X = (x_0, x_1, \ldots, x_{n-1})$ is partitioned into m subsequences $Y_0, Y_1, \ldots, Y_{m-1}$ with k items in each subsequence.
    - While Y_{m-1} may have fewer than k items, without loss of generality (WLOG) we may assume that it has k items here.
  - The subsequences then have the form,

$$Y_i = (x_{i*k}, x_{i*k+1}, \ldots, x_{i*k+k-1})$$

Algorithm PRAM Prefix Computation (X, $\oplus$,S)

- **Step 1**: Each processor P_i computes the prefix sum of the sequence Y_i using the RAM prefix algorithm, and stores these intermediate results in $s_{ik}, s_{ik+1}, \ldots, s_{(i+1)k-1}$.
- **Step 2:** All m PEs execute the preceding PRAM prefix algorithm on the sequence $(s_{k-1}, s_{2k-1}, \ldots, s_{n-1})$, replacing s_{ik-1} with $s_{k-1} \oplus \ldots \oplus s_{ik-1}$.
- **Step 3:** Finally, all P_i for $1 \le i \le m-1$ adjust their partial value sums for all but the final term in their partial sum subseqence by performing the computation

$$s_{ik+j} \leftarrow s_{ik+j} \oplus s_{ik-1}$$

  for $1 \le j \le k-1$.

- **Analysis:**
  - Step 1 takes $O(\lg n) = O(k)$ time.
  - Step 2 takes $O(\lg m) = O(\lg n/k)$
    $= O(\lg n - \lg k) = O(\lg n - \lg \lg n)$
    $= O(\lg n) = O(k)$
  - Step 3 takes O(k) time.
  - The overall time for this algorithm is $O(\lg n)$ and its cost is $O((\lg n) \times n/(\lg n)) = O(n)$
- See pseudocode version on pg 155 of [2].

§4.6 Array Packing

- **Problem:** Assume that we have
  - an array of n elements, $X = \{x_1, x_2, \ldots, x_n\}$
  - Some array elements are *marked* (or *distinguished*).

  The requirements of this problem are to
  - pack the marked elements in the front part of the array.
  - maintain the original order between the marked elements.
  - place the remaining elements in the back of the array.
  - also, maintain the original order between the unmarked elements.
- **Sequential solution**:
  - Uses a technique similar to quicksort.
  - Use two pointers q (initially 1) and r (initially n).
  - Pointer q advances to the right until it hits an unmarked element.
  - Next, r advances to the left until it hits a marked element.
  - The elements at position q and r are switched and the process continues.

  - This process terminates when $q \geq r$.
  - The $\mathbf{O}(n)$ time is optimal.

- **An EREW PRAM Algorithm for Array Packing**
  - Set s_i in P_i to 1 if x_i is marked and set $s_i = 0$ otherwise.

  2. Perform a prefix sum on S to obtain the destination $d_i = s_i$ for each marked x_i .
  3. All PEs set $m = s_n$, the nr of marked elements.
  4. Reset $s_i = 0$ if x_i is marked and $s_i = 1$ otherwise.
  5. Perform a prefix sum on S and set $d_i = s_i + m$ for each unmarked x_i .
  6. Each P_i copies array element x_i into address d_i in X.
- **Algorithm analysis**:
  - Assume n/lg(n) processors are used above.
  - Each prefix sum required O(lg n) time.
  - The broadcast in Step 3 requires O(lg n) time, using a binary tree (in memory) or prefix sum.
    (e.g., prefix sum on b's with $b_1 = a_n$ and $b_i = 0$ for $1 < i \leq n$)
  - All and other steps require constant time.
  - Runs in $\mathbf{O}$(lg n) time and is cost optimal.
- **Note:** There many applications for this algorithm.

An Optimal PRAM Sort

- Two references are listed below. The book by JaJa may be referenced in the future and is a well-known textbook devoted to PRAM algorithm.

  [8] Joseph JaJa, An Introduction to Parallel Algorithms, Addison Wesley, pgs 160-173.

  [9]. R. Cole, Parallel Merge Sort, SIAM Journal on Computing, Vol. 17, 1988, pp. 770-785.
- Cole's Merge Sort (for PRAM)
  - Cole's Merge Sort runs in O(lg n) and requires O(n) processors, so it is cost optimal.
  - The Cole sort is significantly more efficient than most (if not all) other PRAM sorts.
  - A complete presentation for CREW PRAM is given in [8].
    - JaJa states that the algorithm he presents can be modified to run on EREW, but that the details are non-trivial.
    - Akl calls this sort PRAM SORT in [2] and gives a very high level presentation of the EREW version of this algorithm in Ch. 4.
  - Currently, this sort is the best-known PRAM sort is usually the one cited when a cost-optimal PRAM sort using O(n) PEs is needed.

- Comments about some other sorts for PRAM
  - A work-optimal CREW PRAM algorithm that runs in O((lg n) lg lg n) time and uses O(n) processors which is much simpler is given in JaJa's book (pg 158-160).
  - Also, JaJa gives an O(lg n) time randomized sort for CREW PRAM on pages 465-473.
    - With high probability, this algorithm terminates in O(lg n) time and requires O(n lg n) operations
      - i.e., with high-probability, is work-optimal.
  - Sorting is sometimes called the "queen of the algorithms":
    - A speedup in the best-known sort for a parallel model usually results in a similar speedup other algorithms that use sorting.

- **A Divide & Conquer or Simulation Algorithm**
  - To be added from [2,Ch 5], [3,Ch 2], [7,Ch 30].
  - Possible Candidates
    - Merging two sorted lists [2,Ch 5] or [3]
    - Searching an unsorted list
    - Selection algorithm

Symbol Bar -- omit on printing

- ⊕ × s_1 " $ '0 W → ← ≥ ≤ ∧ ∨ ∈ ∉ ⌈ ⌉ ⌊ ⌋