Preliminary Comments

The initial results of this chapter were covered in the Chapter on Combinational Circuits & Sorting Networks. In particular, the 0-1 principle (see CLR pg 42) and Transposition Sort (See CLR pg 44) were covered at the end of Combinational Circuits Chapter.

In particular, the 0-1 principle was covered for a circuit in above chapter, but the argument given here for a linear array of processors is very similar to the one given in the previous set of slides for a circuit.

Likewise, the Transposition sort in the

previous chapter was for a circuit, but almost the same proof works here. In fact, the argument given in this set of slides show that the running time of the transposition sort is exactly n.

Given that we have only a short time left, it seems a better use of our time to not go through very similar proofs, but instead to skip to the 2-D mesh sort algorithm, which is very well-known.

Mesh Models

(Chapter 8)

1. Overview of Mesh and Related models.

a. Diameter:

- The linear array is $O(n)$, which is large.
- The mesh as diameter $O(\sqrt{n})$, which is significantly smaller.

b. The size of the diameter is significant for problems requiring frequent long-range data transfers.

c. Some advantages of 2-D Mesh.

Maximum degree is 4.

Has a regular topology (i.e., is same at all points except for boundaries).

Easily extended by row or column additions.

d. Disadvantages of the 2-D Mesh.

- Diameter is still large.

e. Mesh of Trees and Pyramids.

- Combines mesh and tree models
- Both have a diameter of $O(\lg n)$.
- These models will not be covered in this course.

2. Row-Major Sort

a. Suppose we are given a 2-D mesh with m rows and n columns.

b. Assume the $N = n \times m$ processors are indexed by row-major ordering:

$$\begin{array}{lllll}
P_0 & P_1 & \bullet & \bullet & P_{n-1} \\
P_n & P_{n+1} & \bullet & \bullet & P_{2n-1} \\
P_{2n} & \bullet & \bullet & \bullet & P_{3n-1} \\
\bullet & \bullet & \bullet & \bullet & \bullet \\
P_{n^2-n} & P_{n^2-n+1} & \bullet & \bullet & P_{n^2-1}
\end{array}$$

- Note that processor P_i is in

row j and column k if and only if $i = jn + k$, where $0 \leq k < n$.

- **c.** A sequence $\{x_1, x_2, \ldots, x_{n-1}\}$ of values in a 2-D mesh with x_i in P_i is said to be sorted if $x_1 \leq x_2 \leq \ldots \leq x_{n-1}$.

3. The 0-1 Principle

- **a.** Let A be an algorithm that performs a *predetermined sequence* of comparison-exchanges on a set of N numbers.
- **b.** Each comparison-exchange compares two numbers and determines whether to exchange them, based on the outcome of the comparison.
- **c.** The 0-1 **principle** states that if A correctly sorts all 2^N sequences of length N of 0’s and 1’s, then it correctly sorts any sequence of N arbitrary numbers.
- **d.** The 0-1 principle occurred earlier in text as Problem 3.2.

e. Examples of sorts satisfying this predetermined condition include

- Batcher’s odd-even merge sorting circuit
- linear array sort of last chapter.

f. Examples of sorts not satisfying this condition include

- Quick Sort (comparisons made depends upon values)
- Bubble Sort (Stopping depends upon comparisons)

g. Proof: (0-1 Principle)

- Let $T = \{x_1, x_2, \ldots, x_n\}$ be an unsorted sequence.
- Let $S = \{y_1, y_2, \ldots, y_n\}$ be a sorted version of T.
- Suppose A is an algorithm that sorts all sequences of 0’s and 1’s correctly.
- However, assume that A applied to T incorrectly produces $T' = \{y_1^{'}, y_2^{'}, \ldots, y_n^{'}\}$.

- Let j be the smallest index such that $y_j^{'} \neq y_j$.
- Then, we have the following:
  - $y_i^{'} = y_i \leq y_j$ for $0 \leq i < j$
  - $y_j^{'} > y_j$
  - $y_k^{'} = y_j$ for some $k > j$.
- We create a sequence Z of 0's and 1's from T (using y_j as a spitting value) as follows: For $i = 0, 1, \ldots, n - 1$ let
  - $z_i = 0$ if $x_i \leq y_j$
  - $z_i = 1$ if $x_i > y_j$
- Then for each pair of indices i and m,

$$x_i \leq x_m \text{ implies that } z_i \leq z_m$$

- When Algorithm A is applied to seqence Z, the comparison results are the same as when it is applied to T, so the same action is taken at each step.
- If Algorithm A produces $Z^{'}$ from Z, then the corresponding

values of Z' and T' are

$$Z' = \{ \; 0 \quad \ldots \quad 0 \quad 1 \quad \ldots \quad 0 \quad \ldots$$
$$T' = \{ \; y_0' \quad \ldots \quad y_{j-1}' \quad y_j' \quad \ldots \quad y_k' \quad \ldots$$

- This establishes that Algorithm A also does not sort sequences of 0's and 1's correctly, which is a contradiction.

4. Transposition Sort:

a. The transposition sort is really a sort for linear arrays. It is used here to sort columns and rows of the 2D mesh.

b. Unlike sorts in last chapter, it assumes the data to be sorted is initially located in the PEs and sort does not involve any I/O.

c. Assume that $P_0, P_1, \ldots, P_{N-1}$ is a linear array of PEs with x_i in P_i for each i. This sort must sort a sequence $S = (x_0, x_1, \ldots, x_{N-1})$ into a sequence $S' = (y_0, y_1, \ldots, y_{N-1})$ with y_i in P_i so that $y_i \leq y_k$ when $i \leq k$.

d. **Linear Array Transposition Sort:**

```
i.   For $j = 0$ to $N - 1$ do
ii.      For $i = 0$ to $N - 2$ do
iii.         if $i \bmod 2 = j \bmod 2$
iv.             then compare-exchange($P_i, P_{i+1}$)
v.           endif
vi.      endfor
vii. endfor
```

e. The table below illustrates the initial action of this algorithm when S is the sequence $(1, 1, 1, 1, 0, 0, 0, 0)$.

time	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
u=0	1	1	1	1	0	0	0	0
u=1	1	1	1	1	0	0	0	0
u=2	1	1	1	0	1	0	0	0
u=3	1	1	0	1	0	1	0	0
u=4	1	0	1	0	1	0	1	0

- Notice in the 1^{st} pass, $(even, even+1)$ exchanges are made, while in the 2^{nd} pass, $(odd, odd+1)$ exchanges occur.
- In this example, once a 1 moves right, it continues to move right at each step until it reaches its destination.
- Likewise, once a 0 moves left, it continues to move left at each step until it is in place

f. Correctness is established using the 0-1 principle.

- Assume a sequence Z of 0's

and 1’s are stored in $P_0, P_1, \ldots, P_{N-1}$ with one element per PE.

- As in above example, the algorithm moves the 1’s only to the right and the 0’s only to the left.
- Suppose 0’s occurs q times in the sequence and 1’s occur $N - q$ times.
- Assume the worst case, in which all 1’s initially lie to the left and $N - q$ (i.e., the number of 1’s) is even.
- Then, the rightmost 1 (in P_{N-q-1}) moves right during the second iteration, or when $j = 1$ in the algorithm.
- This allows the second rightmost 1 to move right when $j = 2$.
- This continues until the 1 in P_0 moves right when $j = N - q$ (or

the $N - q + 1$ step, as j is initially 0).

- This leftmost 1 travels right at each iteration afterwards and reaches its destination P_q in $q - 1$ steps.
- Since $j = 0$ initially, in the worst case

$$(N - q + 1) + (q - 1) = N$$

iterations are needed.

5. ***Mesh Sort (Thomas Leighton): Preliminaries***

   a. *Alternate Reference*: F. Thomas Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann, 1992, pg 139-153

   b. Initial Agreements:

   - The 0-1 Principle allows us to restrict our attention to sorting only 0's and1's.

- The Linear Array Transportation Sort (called ”Sort” here) will be used for sorting rows and columns in Mesh Sort.
- The presentation is simpler if we assume the matrix has m-row and n-column mesh, where
  - $m = 2^s$
  - $n = \sqrt{n} \times \sqrt{n} = 2^r \times 2^r = 2^{2r}$
  - $s \geq r$
- *Observe:*
  - $N = m \times n = 2^{2r+s}$
  - $\sqrt{n} = 2^r \leq 2^s = m$
  - $m/\sqrt{n} = 2^{s-r} \geq 1$ and this value is an integer, so $\sqrt{n}$ divides m evenly
- Above assumptions allow us to partition the matrix into submatrices of size $\sqrt{n} \times \sqrt{n}$

c. Region Definitions

- **Horizonal slice:** As shown in Figure 8.4(a), the m rows can be partitioned evenly into horizonal strips, each with $\sqrt{n}$ rows, since

$$m/\sqrt{n} = 2^{s-r} \geq 1$$

- **Vertical Slice:** As shown in Figure 8.4(b), a vertical slice is a submesh with m rows and $\sqrt{n}$ columns.
  - There are $\sqrt{n}$ of these vertical slices.
- **Block:** As shown in Figure 8.4(c), a block is the intersection of a vertical slice with a horizonal slice.
  - Each block is a $\sqrt{n} \times \sqrt{n}$ submesh.

d. Illustration:

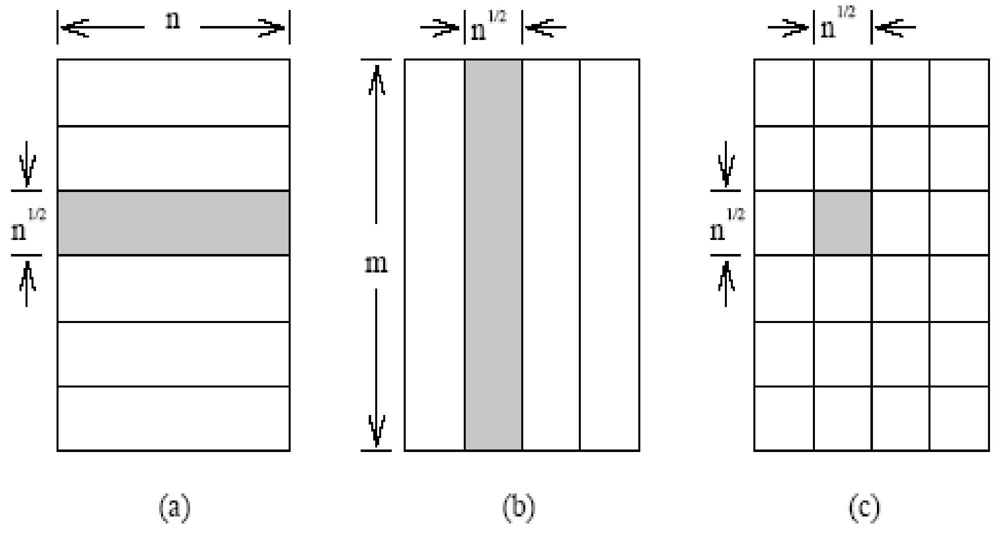

Figure 8.4: Dividing a mesh into submeshes: (a) Horizontal slice; (b) Vertical Slic
(c) Block.

e. **Uniformity**
   - **Uniform Region:** A row, horizonal slice, vertical slice, or block consisting either of all 0’s or all 1’s.
   - **Non-uniform Region:** A row, horizonal slice, vertical slice, or block containing a mixture of 0’s and 1’s.

f. *Observation:* When the sorting algorithm terminates, the mesh consists of zero or more uniform rows filled with 0's, followed by at most one non-uniform row, followed by zero or more uniform rows filled with 1's.

6. ***Three Basic Operations***
   a. **Operation BALANCE:**
      - Applied to a horizonal or vertical slice.
      - *Effect of BALANCE:* In a $v \times w$ mesh, the number of 0's and 1's are balanced among the w columns, leaving at most $\min\{v, w\}$ non-uniform rows after the columns are sorted.
        - Note this is obviously true if $v < w$. In this case, we normally will apply BALANCE to the $w \times v$ mesh of w rows and v columns instead.
        - We discuss the $v \times w$ mesh case where $v > w$ below.

- **Three Steps** of BALANCE Operation:
  - **i.** Sort each column in nondecreasing order using SORT.
  - **ii.** Shift i^{th} row of submesh cyclically $i \bmod w$ positions right.
  - **iii.** Sort each column in nondecreasing order using SORT.
- Step (i) pushes all 0’s to the top and all 1’s to the bottom in each of the w columns.
- Effect of Cyclic Shift in Step (ii) on first element of each row:

$$\begin{array}{ccc}
a_{1,1} & \bullet & \bullet \\
\bullet & a_{2,1} & \bullet \\
\bullet & \bullet & a_{3,1} \\
a_{4,1} & \bullet & \bullet \\
\bullet & a_{5,1} & \bullet
\end{array}$$

- Overall effect of Steps (i-ii) is to spread the 0’s and 1’s from each column across all w columns.
- Suppose i and j are distinct columns and k is an arbitrary column in the submesh.
  - Step (ii) spreads the elements of column k among all columns.
  - The number of 0’s received from column k by columns i and j differ at most by 1.
  - Likewise, the number of

1's that columns i and j receive from column k differ at most by 1.

- *Summary*: After Step (ii), the number of 0's (respectively, the number of 1's) in columns i and j can differ at most by w.
- **Combined Effect after Step (iii) on $v \times w$ submatrix**:
  - at most $v = \min\{v, w\}$ rows are non-uniform
  - the non-uniform rows are consecutive and separate uniform rows of 0's from uniform rows of 1's.
- **Example:** If the height of the box in Figure 8.5 is increased to about 3 times its width, it illustrates the effect of applying BALANCE alone to a vertical slice of the original mesh.

b. Operation **UNBLOCK**

- Applied to a block (i.e., a

$\sqrt{n} \times \sqrt{n}$ submesh)

- Two Steps of the UNBLOCK Operation
  - **i.** Cyclically shift the elements in each row i to the right $i\sqrt{n} \bmod n$ positions.
  - **ii.** Sort each column in nondecreasing order using SORT.
- *Effect of UNBLOCK:* Distributes one element in each block to each column in the mesh, so that
  - ▪ each uniform block produces a uniform row.
  - ▪ each non-uniform block produces at most one non-uniform row.
- *Justification of preceding claim:*
  - ▪ Step 1 transfers each of the n elements of a block

to a different column.

- ■ Example: Mesh before and after Step1. (Here $m = 2^2 = 4$, $n = 2^{2\times2} = 16$, and $\sqrt{n} = 4$.

7. Example:

$$\left|\begin{array}{cccccccccccccccc}
. & . & . & . & 0 & 0 & 0 & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & 0 & 0 & 1 & 0 & . & . & . & . & . & . & . & . \\
. & . & . & . & 0 & 1 & 0 & 0 & . & . & . & . & . & . & . & . \\
. & . & . & . & 1 & 0 & 0 & 0 & . & . & . & . & . & . & . & .
\end{array}\right|$$

$$\left|\begin{array}{cccccccccccccccc}
. & . & . & . & 0 & 0 & 0 & 1 & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & 0 & 0 & 1 & 0 & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right.$$

1. Assume there are b non-uniform blocks before executing UNBLOCK.

a. • ■ After Step (i), the

difference in the number of 0's of two columns is at most b.

- After the column-sort in Step (ii), at most b non-uniform rows remain in the mesh.
- The non-uniform rows are consecutive and separate the uniform rows of 0's from the uniform rows of 1's.

c Operation **SHEAR**

- *Steps of SHEAR*
  - **i.** Sort all even numbered (*odd numbered*) rows in increasing (*decreasing, respectively*) order using SORT.
  - **ii.** Sort each column in increasing order using SORT.
- *Effect of SHEAR*: If there are b

consecutive non-uniform rows initially, then after operation SHEAR, there are at most $\lceil b/2 \rceil$ consecutive non-uniform rows.

- *Justification of above Claim:*
    - ■ Let mesh have b consecutive non-uniform rows initially.
    - ■ Consider a *pair* of adjacent non-uniform rows.
    - ■ Step (i) places the 0's of the pair of adjacent rows at opposite ends.
    - ■ Then a column may get at most one more 0 or 1 than any other column from one pair of rows.

```
←0/1→|←0's→|←—0/1—-→
 0  0  0  0  1  1  1  1
 1  1  0  0  0  0  0  0
```

- Since there are $\lceil b/2 \rceil$ pairs of adjacent non-uniform rows, the difference in the number of 0's in any two columns is at most $\lceil b/2 \rceil$.
- Sorting the columns in Step (ii) causes at most $\lceil b/2 \rceil$ non-uniform rows to remain.
- Again, the non-uniform rows separate the uniform rows of 0's from the uniform rows of 1's.

7 Algorithm MESH SORT

The number of basic row/col opns for each step is given after the step.

Step 1: For all vertical slices, do in parallel

- ▪ BALANCE (3)

Step 2: UNBLOCK (2)

Step 3: For all horizonal slices, do in parallel

- ■ BALANCE (3)

Step 4: UNBLOCK (2)

Step 5: For $i = 1$ to 3, do (sequentially)

- ■ SHEAR (2 each loop)

Step 6: SORT each row (1)

Total row or column operations: 17

8 Correctness of MESH SORT

a. After Step 1, the entire mesh has at most $2\sqrt{n}$ nonuniform blocks.

- BALANCE leaves at most $\sqrt{n}$ nonuniform rows in each *vertical* (i.e., $m \times \sqrt{n}$) slice.
- Since the nonuniform rows are consecutive, there are at most **two** nonuniform blocks in each *vertical* slice.
- See Figure 8.7 below

b. After Step 2, UNBLOCK leaves at most $2\sqrt{n}$ nonuniform rows, which are consecutive.

- Now there are at most **three** nonuniform *horizonal* slices in entire mesh.

c. In Step 3, BALANCE is applied (in parallel) to all the $\sqrt{n} \times n$ *horizonal* strips in parallel

- In effect, applied to rotated $n \times \sqrt{n}$ mesh strips.
- BALANCE applied to *one nonuniform horizonal* slice produces at most 2 nonuniform blocks in this slice (as in Step 1).
- Since only 3 horizonal slices were nonuniform (after Step 2), *at most* 6 nonuniform blocks remain after Step 3.

d. Figure 8.7 shows action after ”balance” operations in Step 1 and Step 3.

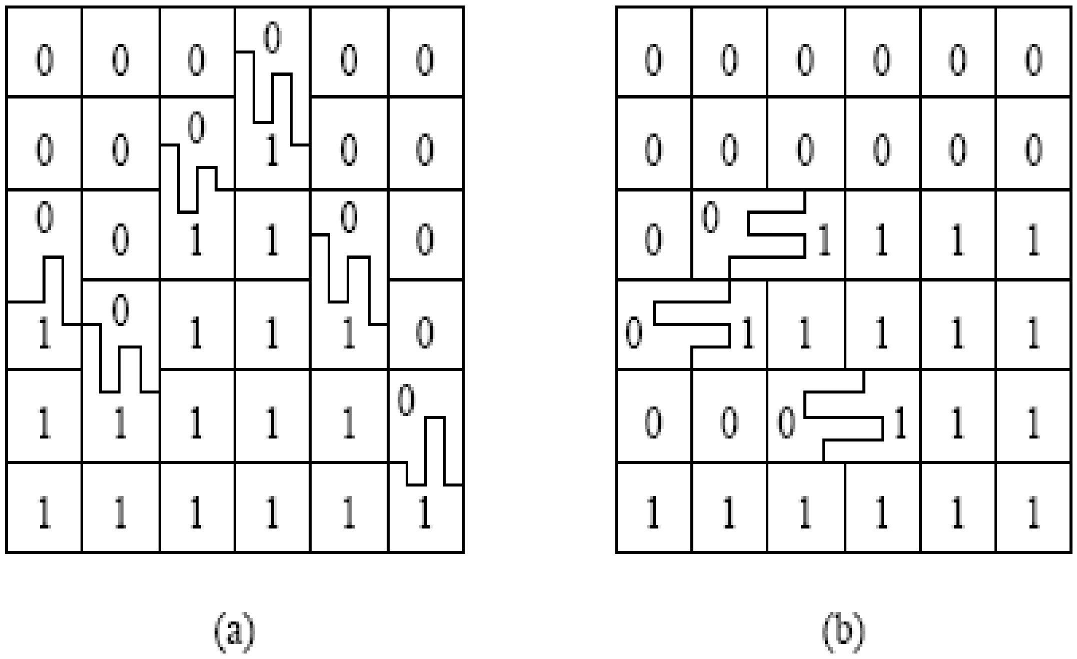

Figure 8.7: Proving the correctness of MESH SORT: (a) After Step 1; (b)
Step 3.

1. **a.** Step 4: Since only 6 blocks are nonuniform, UNBLOCK produces at most 6 nonuniform rows.
   b. In Step 5, SHEAR reduces the 6 nonuniform rows to
   - ■ $6/2 = 3$ after iteration 1.
     ■ $\lceil 3/2 \rceil = 2$ after iteration 2.
     ■ $2/2 = 1$ after iteration 3.

   c. In Step 6, a sort of all rows will sort the (possibly) one non-uniform row.

9 Analysis of MESH SORT

a. There are 17 basic row/column operations in all, when the substeps of BALANCE, UNBLOCK, and SHEAR are counted.

b. Each step above is a sort of a row or column or a cyclic shifting of a row by at most $n - 1$ positions.

c. Using the Linear Transportation Sort, each sorting step requires $O(n)$ or $O(m)$ time, depending on whether a row or column is sorted.

d. Each cyclic shift of a row takes $O(n)$ time, since at most $n - 1$ parallel moves are required to transfer items to their new row location.

□ ⇆ □ ⇆ □ ⇆ □ ⇆ □ ⇆ □ ⇆ □ ⇆ □

e. Alternately, above step can be

done by row sorts on the row-designation address of each item.

f. ***Running Time:*** $O(n+m)$, or $O(n)$ if we assume that m is $O(n)$.
   - This time is **best possible** on the 2D mesh, since an item may have to be moved from $P(0,0)$ to $P(m-1,n-1)$.

g. ***Cost:*** Assume that $m = n = \sqrt{N}$.
   - The running time is $t(N) = O(\sqrt{N})$
   - The cost is $c(N) = O(N^{3/2})$
   - The cost is not optimal, since an $O(N \lg N)$ cost is possible for a sequential sort of N items.
   - Note: For the case where $n = m$,
     - ***If*** this algorithm could be adjusted to allow each processor to handle

$$O(\frac{N^{3/2}}{N \lg N}) = O(\frac{\sqrt{N}}{\lg N}) = O(\frac{n}{\lg n})$$

nodes **without** changing its $O(n)$ running time,

- ***then*** the resulting algorithm would be optimal.

Note: ”Applying balance in Step 3 to a rotated strip” is easier to do if you transpose matrix so rows are in column position, do this step, then take transpose again. Rotating paper so rows are in column position can help, but the actual columns are in reverse order.