Mapping Normal Algorithms to Meshes

- Normal algorithms work well on hypercubes.
- Because hypercube machines are expensive to build, we map normal algorithms to 1-D and 2-D arrays, which are inexpensive.
- Shuffle and unshuffle operations are key to the translation of normal algorithms to meshes.
- When a deck of card is split in half and the two sets of cards are interlaced, a shuffle operation is performed.

Example on $8 = 2^3$ elements:

Original  0 1 2 3 4 5 6 7
Shuffled  0 4 1 5 2 6 3 7

Shuffle Operations on Linear Arrays

- Consider shuffle of $2^d$ items on 1-D arrays. It can be done by a sequence of swaps of adjacent elements on the array, as shown.

<table>
<thead>
<tr>
<th>Step</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 3</td>
<td>0 4 1 5 2 6 3 7</td>
</tr>
<tr>
<td>Step 2</td>
<td>0 1 4 2 5 3 6 7</td>
</tr>
<tr>
<td>Step 1</td>
<td>0 1 2 4 3 5 6 7</td>
</tr>
</tbody>
</table>

- Could you write a small program for each array processor using the integer it contains as well as the cell number to decide when to swap and when to terminate?
- Number of steps: $2^{d-1} - 1$
- Unshuffle: reverse shuffle steps

Importance of Shuffle Operations

- Represent 8 integers in binary

Original  000 001 010 011 100 101 110 111
Shuffled  000 100 001 101 010 110 011 111

- In original, elements $(i, i+1)$ for $i$ even differ in the lsb. When shuffled, they differ in msb
- A normal algorithm on a hypercube swaps values in elements whose indices differ in one component of their representation.
Ascending Normal Algs. on 1-D Array

• Lowest dimen.swap: swap \((i, i+1)\) for \(i\) even.

• Second lowest dim. swap: shuffle contiguous blocks of 4 elements, swap \((i, i+1)\) for \(i\) even.

• Here is shuffle on blocks of 4 elements:

<table>
<thead>
<tr>
<th>Original</th>
<th>000 001 010 011</th>
<th>100 101 110 111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shuffled</td>
<td>000 010 001 111</td>
<td>100 110 101 111</td>
</tr>
</tbody>
</table>

• Since not first step in shuffle on 8, unshuffle

• Third lowest dim. swap, shuffle contiguous blocks of 8 elements, swap \((i, i+1)\) for \(i\) even.

• Repeat until done: a) reverse last shuffle on blocks of \(2^k\) elements, b) shuffle on blocks of \(2^{k+1}\) elements, swap \((i, i+1)\) for \(i\) even.

• No. of steps in shuffles: \(2(2^{k-1}-1)\) for \(k= 2, 3, ..., d\) or \(2(1+3+7+15+...+2^d-1)-1 = 2^d -d -1\)

Normal Algorithms on 2-D Arrays

• Consider \(m \times m\) array. Index elements \((r, c)\), for \(0 \leq r, c \leq m-1\) in row major order, that is, the column indices increase from left to right as in 1-D array. Row indices also increase from top to bottom, as shown.

\[
\begin{align*}
(0,0) & (0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7) \\
(1,0) & (1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) \\
(2,0) & (2,1) (2,2) (2,3) (2,4) (2,5) (2,6) (2,7) \\
(3,0) & (3,1) (3,2) (3,3) (3,4) (3,5) (3,6) (3,7) \\
(4,0) & (4,1) (4,2) (4,3) (4,4) (4,5) (4,6) (4,7) \\
(5,0) & (5,1) (5,2) (5,3) (5,4) (5,5) (5,6) (5,7) \\
(6,0) & (6,1) (6,2) (6,3) (6,4) (6,5) (6,6) (6,7) \\
(7,0) & (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7) \\
\end{align*}
\]

• Treat an index as \(cm+r\). All normal alg. exchanges can be done by doing shuffles on rows followed by shuffles on columns.

• Ascend normal alg. uses \(O(\sqrt{n})\) steps, \(n=m^2\).

• See book for normal alg. on CCC

The PRAM Model

- The PRAM is really an abstract progr model
- Four types: EREW, ERCW, CREW, CRCW
- Can Boolean functions be computed quickly?

How to represent function?
Can we use concurrency to good advantage?
Is this use of concurrency realistic?

Simulating CRCW on EREW PRAM

- The CRCW PRAM can be simulated on the EREW PRAM in time that is larger by a factor of the time to sort a list of \(p\) items.

- Consider reading cycle. Processor \(p_j\) puts into location \(M_j\) value \((a_j, j)\) indicating it wishes to read from location \(a_j\). Sort pairs. \(p_i\) reads location \(M_i\) on first step and loc. \(i-1\) on second. If they hold different addresses, \(p_i\) reads from address given in \(M_i\) and puts it into \(M_i\).

- Using a segmented copy-right prefix computation (see book), copy value at location \(a\) to all locations \(M_k\) that request it. This step takes \(O(\log p)\) time for \(p\)-processor PRAM. Sorting takes more time.

- To write, repeat except let \(p_j\) store value \(v\) to be written in \(M_j\) and let first (arbitrary) value be written.