Pipelining Defined

- A “product” in a stage of partial completion, is moved along through a series of “stations”, becoming more complete at each station.
- The stations operate in parallel on multiple products, each working on a product in its respective stage of completion.
- There may or may not be parallelism within a given station.

Examples of Computational Products

<table>
<thead>
<tr>
<th>Product</th>
<th>Partially Complete Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>Approximation to a number</td>
</tr>
<tr>
<td>Sorted array</td>
<td>Partially sorted array</td>
</tr>
<tr>
<td>Image</td>
<td>Transformed image</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Raw image</th>
<th>Thinned image</th>
<th>Rotated &amp; thinned image</th>
<th>Rotated, thinned, and cropped image</th>
<th>Rotated, thinned, cropped, and inverted image</th>
</tr>
</thead>
</table>

Utility of Pipelines

- Pipelines can provide a modular approach to constructing functions, rather than trying to invent or manage one single comprehensive function.
- Unix users are used to this kind of thing:
  (thin < image) | rotate | crop | invert

Pipeline Timing from the Products’ View
Pipeline Timing from the processors’ view

Distributed Pipelining (my definition)

- In *distributed* pipelining, the result is not what comes out the last stage in one step, but rather the collective sequence of things coming out.

```
# 3 7 2 1 8 5 6 # 8 7 6 5 3 2 1
```

Example of Distributed Pipeline (adapted from PP): Sorting

Pipeline Sorting

- The method shown is $O(N)$ and requires $N$ processors.
- Also, the number of items to be sorted is limited by the number of processors.
- As soon as a sequence clears the first processor, the next sequence can be started, so $N-1$ processors can be kept busy.

Pipelinable, Parallel, Sorting Methods

- Sorting networks were first proposed by Kenneth Batcher (Goodyear Aerospace) in the 1960’s.
- They are networks constructed from a number of simple devices, called comparators.
- Sorting networks are discussed in PP 9.2.7 and 9.2.8. We discuss them here with pipelining.

Sorting Network Comparator

- A single comparator inputs two numbers and outputs them in sorted order.
If desired, an individual comparator can be pipelined at the bit-level, as a finite-state machine accepting inputs MSB first (assuming the same number of bits in both numbers):

- **Analysis**
  - Clearly the min
  - Whatever is left
  - Clearly the max

- **Exercises**
  - Construct a sorting network for 4 numbers
  - Construct a sorting network for 8 numbers

- **General Constructions**
  - **Odd-Even Merging**
    - Assume 2n elements to be sorted.
    - Split the elements into two groups.
    - Sort each half recursively.
    - Merge the odd-indexed components of the result.
    - Merge the even-indexed components of the result.
    - Merge the output of the merges.

- **Merging**
  - Can be done in 1 stage.

For synchronous sorting, we'd have to include some padding on the non-compared lines.
In general, how can one verify that a proposed scheme sorts?

- 0-1 Principle
- Symbolic analysis ("Boolean" operators)
- Partial-order analysis (Gale & Karp, Liu)

Timing Analysis of Sorting by Odd-Even Merging

- Let $S(n)$ = parallel time to sort $n$ elements
- Let $M(n)$ = time to merge $n/2$ with $n/2$ elements
- Recurrences:
  - $S(1) = 0$
  - $S(2n) = S(n) + M(2n)$
  - $M(2) = 1$
  - $M(2n) = M(n) + 1$
  - assuming we don’t charge for splitting into 2 groups

Solving Recurrences: Assume powers of 2

- Merging:
  - $M(2) = 1$
  - $M(2n) = M(n) + 1$
  - Therefore
    - $M(2^n) = n$
- Sorting:
  - $S(2^n) = S(2^{n-1}) + n$
  - Giving $S(2^n) = n + (n-1) + (n-2) + \ldots + 1 = n(n+1)/2$
  - i.e. $S(N) = O(\log^2 N)$

A Different Sort of Construction

- Bitonic sorting
- Define a bitonic sequence to be one in which is either:
  - a strictly ascending portion, followed by a strictly descending portion, OR
  - any cyclic shift of a bitonic sequence.

Bitonicity

[Diagram showing ascending-descending sequence]

[Diagram showing ascending-descending sequence]
Bitonicity

![Diagram of ascending and descending sequences]

Bitonic Sorting:
(This is the main reason for the definition of bitonic.)

- An already-bitonic sequence can be sorted by the following scheme:
  - Do pairwise compare-exchanges between elements $j$ and $j+n/2$ for $j = 1, 2, \ldots, n/2$
  - This gives two sequences that:
    - are bitonic
    - all elements of one are $\leq$ all elements of the other
  - Repeating this scheme recursively with the resulting two bitonic sequences gives a sorted sequence.

A Point that Needs Proof

- Example: $11100001 \quad 000101111$
  - If the number of 0's and 1's is the same, then we end up with the form:
    \[
    0 \ldots 0 1 \ldots 1
    \]
  - If there are more 0's, we end up with the form
    \[
    0 \ldots 0 0 \ldots 1
    \]
  - If there are more 1's, we end up with the form
    \[
    0 \ldots 1 \ldots 1 0 \ldots 1
    \]
  - all of which have the desired property.

A Point that Needs Proof, cont'd

- Start with: $[8, 9, 7, 4, 2, 1, 3, 5]$
  - Do compare-exchanges between elements $j$ and $j+n/2$ for $j = 1, 2, \ldots, n/2$:
    \[
    [8, 9, 7, 4, 2, 1, 3, 5] \quad [2, 1, 3, 4, 8, 9, 7, 5]
    \]
  - This gives two sequences that:
    - are bitonic
    - all elements of one are $\leq$ all elements of the other
  - Repeating this scheme recursively with the resulting two bitonic sequences gives a sorted sequence, just like in quicksort.
Bitonic Sorting

- Two already sorted sequences can be made into a bitonic sequence by reversing the second one and abutting them.
- This may seem like a counter-intuitive thing to do, since so arranging them is contrary to the desired ultimate sorted order.

Bitonic Sort Summary

- To create a bitonic sequence:
  - Sort the two halves of the sequence by any method, so that one half is ascending, the other descending.
- To sort a sequence known to be bitonic:
  - Compare-exchange elements $j$ and $j+n/2$, for $j = 0, 1, 2, \ldots, n/2-1$
  - It can be shown that the two halves are bitonic, and all of one half is ≤ all of the other.
  - Therefore, sort them as bitonic sequences, then concatenate the results.

Bitonic Sort Recursion

\[ B_n = \text{bitonic compare exchanges (CE)} \]

General Sort Recursion

\[ S = \text{sorter for arbitrary} \]
\[ B = \text{sorter for bitonic sequence} \]

Bitonic Sort Examples

- $B_2$
- $B_4$
- $B_8$

General Sort Examples

- $S_4$

$2S_2's$

$B_4$
General Sort Examples

- $S_8$

Exercises

- Analyze the time taken for bitonic sorting.
- Comment on the pipelinability of bitonic sorting.
- What does bitonic sorting have to do with hypercubes?

Other facets of sorting networks

- The number of comparators for odd-even merging or bitonic sorting are both $O(n \log^2 n)$, which is typical of the most accessible constructions.
- An upper bound of $O(n \log n)$ comparators has been shown, but the constants are large (in the 1000's).

Shear Fun

- ShearSort, invented by Isaac Scherson
- Non-pipelined, but similar idea to bitonic
- Suitable for rectangular mesh of PE's
- Scheme, for $n$ rows:
  - repeat $1+\log n$ times:
    - Sort the rows “zig-zag” (alternating increasing and decreasing).
    - Sort the columns, all increasing.
    - (The last set of column sorts can be omitted.)

ShearSort Example

<table>
<thead>
<tr>
<th>Start</th>
<th>9 3 15 1</th>
<th>1 3 9 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 14 6 8</td>
<td>14 8 6 0</td>
<td></td>
</tr>
<tr>
<td>2 5 13 11</td>
<td>2 5 11 13</td>
<td></td>
</tr>
<tr>
<td>12 7 4 10</td>
<td>12 10 7 4</td>
<td></td>
</tr>
<tr>
<td>1 3 6 0</td>
<td>1 3 9 6</td>
<td></td>
</tr>
<tr>
<td>2 5 7 4</td>
<td>2 5 7 4</td>
<td></td>
</tr>
<tr>
<td>12 8 9 13</td>
<td>12 8 9 13</td>
<td></td>
</tr>
<tr>
<td>14 10 11 15</td>
<td>14 10 11 15</td>
<td></td>
</tr>
<tr>
<td>0 1 3 2</td>
<td>0 1 3 2</td>
<td></td>
</tr>
<tr>
<td>7 5 4 6</td>
<td>7 5 4 6</td>
<td></td>
</tr>
<tr>
<td>8 9 11 10</td>
<td>8 9 10 11</td>
<td></td>
</tr>
<tr>
<td>15 14 12 13</td>
<td>15 14 13 12</td>
<td></td>
</tr>
<tr>
<td>Done</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Proof of Shearsort

- Use the 0-1 principle
- Suffices to show that any initial assignment of 0’s and 1’s gets sorted.
- A row is called
  - 1-row, if it consists of all 1’s
  - 0-row, if it consists of all 0’s
  - mixed row, otherwise
- Need to show that sorting leaves us with 0-rows, a mixed row, then 1-rows.

Proof of Shearsort

- Need to show that sorting leaves us with 0-rows, a mixed row, then 1-rows.
- Claim: Each iteration at least halves the number of mixed rows.
- Thus after log n iterations (n = #rows), there is only one mixed row.
- The final row sort sorts this mixed row, wherever it may be.

Proof of Claim

- Claim: Each iteration at least halves the number of mixed rows.
- Proof: Consider an adjacent pair of mixed rows after a row sort.
- There can only be three kinds of configurations:
  a) \( 0 \ldots 1 \ldots 1 \) equal 0’s and 1’s
     \( 1 \ldots 1 \ldots 0 \)
  b) \( 0 \ldots 0 1 \ldots 1 \) surplus of 0’s
     \( 1 \ldots 1 0 \ldots 0 \)
  c) \( 0 \ldots 0 1 \ldots 1 \) surplus of 1’s
     \( 1 \ldots 1 0 \ldots 0 \)

Proof of Claim, cont’d

- Transitions from the three kinds of configurations:
  a) \( \begin{array}{c} 0 \ldots 1 \ldots 1 \ \ 0 \ldots \ldots \ldots 0 \\ 1 \ldots 1 0 \ldots 0 \end{array} \)
  b) \( \begin{array}{c} 0 \ldots 0 1 \ldots 1 \\ 1 \ldots 1 0 \ldots 0 \end{array} \)
  c) \( \begin{array}{c} 0 \ldots 0 1 \ldots 1 \\ 1 \ldots 1 0 \ldots 0 \end{array} \)
- In each case, the number of mixed rows is at least halved, as desired.

Exercises

- Give an upper-bound for shared-memory implementation of shearsort.
- Give an upper-bound for distributed-memory implementation of shearsort, assuming a mesh-wise interconnection of PE’s.
- Extend shearsort to 3 dimensions, and analyze.