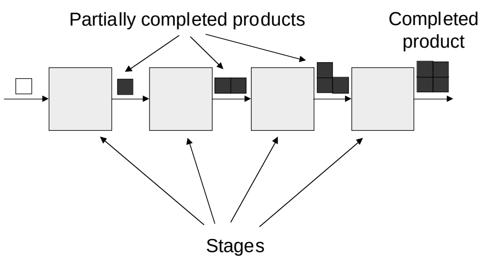
Pipelining

PP Ch. 5

Pipelining Defined

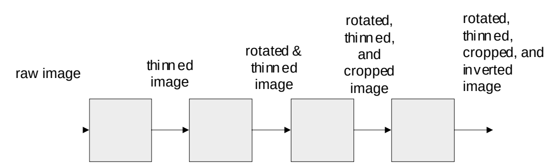
- A “product” in a stage of partial completion, is moved along through a series of “stations”, becoming more complete at each station.
- The stations operate in parallel on multiple products, each working on a product in its respective stage of completion.
- There may or may not be parallelism *within*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

Pipelining



Examples of Computational Products

Product	Partially Complete Product
Number	Approximation to a number
Sorted array	Partially sorted array
Image	Transformed image

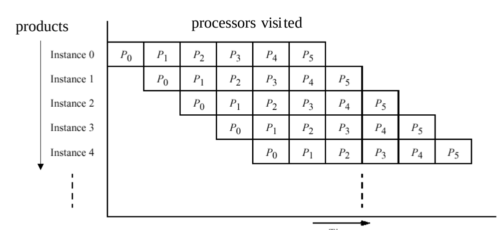


Utility of Pipelines

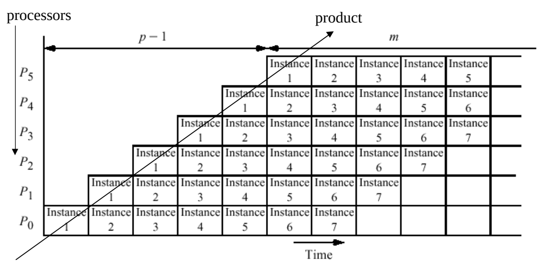
- Pipelines can provide a modular approach to constructing functions, rather than trying to invent or manage one single comprehensive function.
- Unix users are used to this kind of thing:

  (thin < image) | rotate | crop | invert

Pipeline Timing from the products’ view

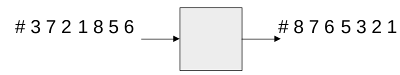


Pipeline Timing from the processors' view

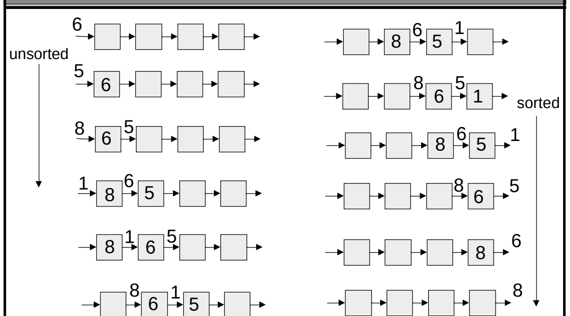


Distributed Pipelining
(my definition)

- 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.



Example of Distributed Pipeline (adapted from PP): Sorting



Pipeline Sorting

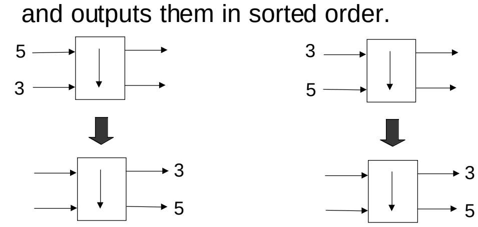
- The method shown is O(N) and requires N processors.
- Also, the number of items to be sorted is limited by the number of processors.
- As soon as a sequence clears the first processor, the next sequence can be started, so N-1 processors can be kept busy.

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.

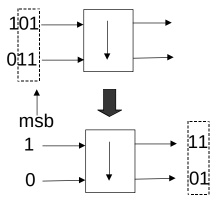
Sorting Network Comparator

- A single comparator inputs two numbers and outputs them in sorted order.



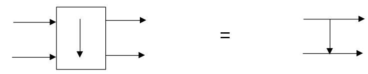
Sorting Network Comparator

- 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:

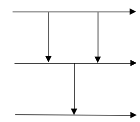


Sorting Networks

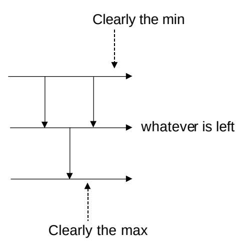
- Comparator abstraction (for drawing networks)



- A network that sorts 3 numbers



Analysis



Exercises

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

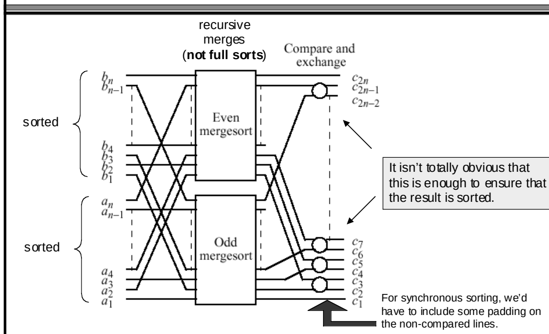
General Constructions

- 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.

parallel

can be done in 1 stage!!

Merging



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

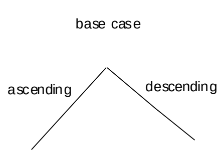
- Merging:
  - $M(2) = 1$
  - $M(2n) = M(n) + 1$
- Therefore
  - $M(2^n) = n$
- Sorting:
  - $S(2^n) = S(2^{n-1}) + n$
- Giving $S(2^n) = n + (n-1) + (n-2) + \ldots + 1 = n*(n+1)/2$

i.e. $\boxed{S(N) = O(\log^2 N)}$

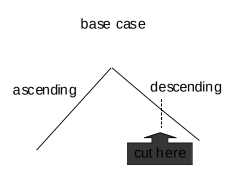
A Different Sort of Construction

- 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.

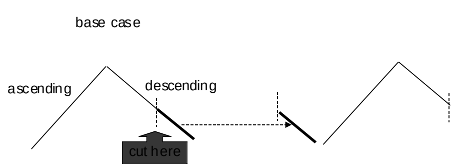
Bitonicity



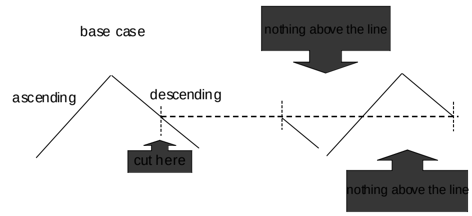
Bitonicity



Bitonicity



Bitonicity



Bitonic Sorting:

(This is the main *reason* for the definition of bitonic.)

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

A Point that Needs Proof

- all elements of one are ≤ all elements of the other
- In order to show this, we could use the 0-1 principle.
- Map an arbitrary numeric sequence into 0's and 1's by choosing an arbitrary threshold θ:
  $x > \theta$ maps x to 1, otherwise x is mapped to 0.
- For a bitonic sequence, the corresponding 0-1 sequence is 000…01…1 or ***some rotation*** thereof.
- When we do the compare-exchange step, the result will have the desired property.
- Example: $\underbrace{1\,1\,1\,1\,0}\,\underbrace{0\,0\,0\,1\,1} \Rightarrow \underbrace{0\,0\,0\,1\,0}\ \le\ \underbrace{1\,1\,1\,1\,1}$

A Point that Needs Proof, cont'd

- Example: $\underbrace{1\,1\,1\,1\,0}\,\underbrace{0\,0\,0\,1\,1} \Rightarrow \underbrace{0\,0\,0\,1\,0}\ \le\ \underbrace{1\,1\,1\,1\,1}$

If the number of 0's and 1's is the same, then we end up with the form:

$\underbrace{0\,.\,.\,.\,.\,.\,0}\ \underbrace{1\,.\,.\,.\,.\,.\,1}$

If there are more 0's, we end up with the form

$\underbrace{0\,.\,.\,.\,.\,0}\ \underbrace{0\,.\,.1.\,.1.\,.0}$

If there are more 1's, we end up with the form

$\underbrace{0\,.\,.\,1.\,.1\,.\,0}\ \underbrace{1.\,.\,.\,.\,.\,1}$

all of which have the desired property.

Bitonic Sorting Example

- Start with: [8, 9, 7, 4, 2, 1, 3, 5]
- Do compare-exchanges between elements j and j+n/2 for j = 1, 2, …, n/2:

braces indicate bitonic

$\overbrace{[8, 9, 7, 4, 2, 1, 3, 5]} \qquad [\overbrace{2, 1, 3, 4}, \overbrace{8, 9, 7, 5}]$

This gives two sequences that:
- are bitonic
- **all elements of one are ≤ 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

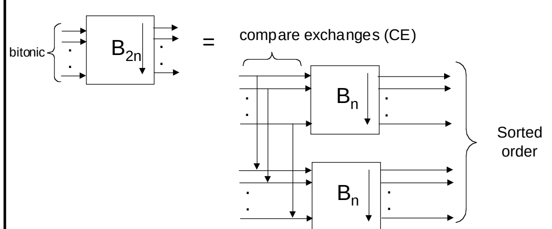
- Two already *sorted* sequences can be made into a bitonic sequence by reversing the second one and abutting them.

- 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

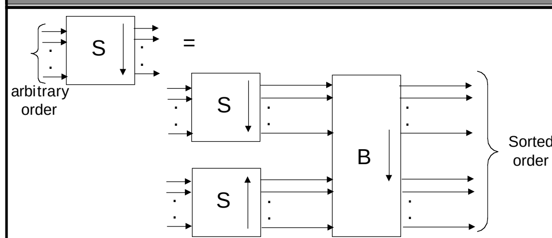
- To create a bitonic sequence:
  - Sort the two halves of the sequence by any method, so that one half is ascending, the other descending.
- To sort a sequence known to be bitonic:
  - Compare-exchange elements j and j+n/2, for j = 0, 1, 2, …, n/2-1
  - It can be shown that the two halves are bitonic,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

Bitonic Sort Recursion



B = sorter for bitonic sequence

General Sort Recursion

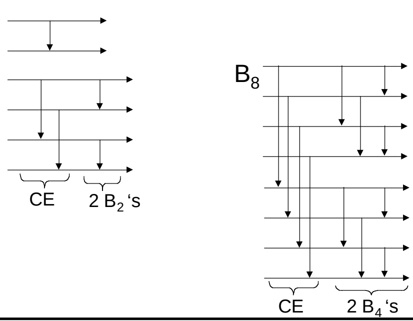


S = sorter for arbitrary

B = sorter for bitonic sequence

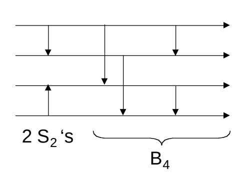
Bitonic Sort Examples

- B_2
- B_4



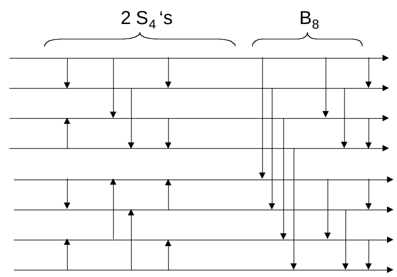
General Sort Examples

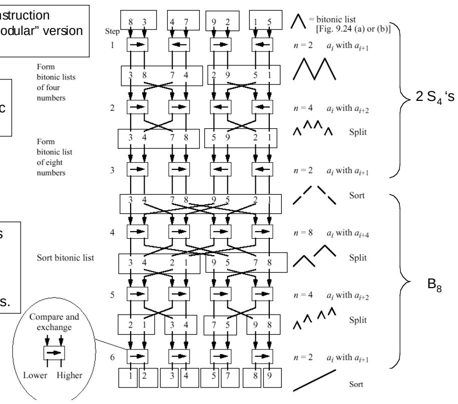
- S_4



General Sort Examples

- S_8





The overall construction for 8 inputs, "modular" version (from PP).

Red blocks show bitonic sequences.

Green blocks show bitonic sequences, ordered among blocks.

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

Prof. Isaac Scherson, UCI (aka The Schark)

- 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

Start

9	3	15	1
0	14	6	8
2	5	13	11
12	7	4	10

1	3	9	15
14	8	6	0
2	5	11	13
12	10	7	4

1	3	6	0
2	5	7	4
12	8	9	13
14	10	11	15

0	1	3	6
7	5	4	2
8	9	12	13
15	14	11	10

0	1	3	2
7	5	4	6
8	9	11	10
15	14	12	13

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

Done

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 . . . . 1. . . . 1 equal 0's and 1's
   1 . . . 1 0 . . . 0

b) 0 . . . . 0 1 . . 1 surplus of 0's
   1 . . 1 0 0

c) 0 . . 0 1 . . . . 1 surplus of 1's
   1 . . . . .1 0. . 0

Proof of Claim, cont'd

- Transitions from the three kinds of configurations:

a) 0 . . . . 1. . . . 1 ⇒ 0 0
   1 . . . 1 0 . . . 0 ⇒ 1 1

b) 0 . . . . 0 1 . . 1 ⇒ 0 0
   1 . . 1 0 . . . . 0 ⇒ 1 . . . . 1 0 . . 0

c) 0 . . 0 1 . . . . 1 ⇒ 0 . . . . 0 1 . . 1
   1 . . . . .1 0. . 0 ⇒ 1 1

- 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.