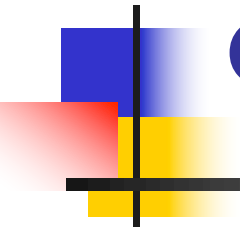
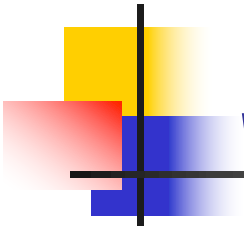


CS475 Parallel Programming

Sorting

Wim Bohm, Colorado State University

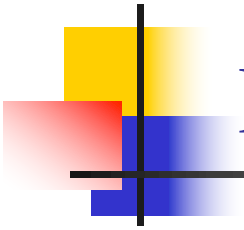


Sorting Problem

- Sorting
  - Input: sequence $S = (a_0, a_1, \ldots, a_{n-1})$
  - Output: $(b_0, b_1, \ldots, b_{n-1})$ = permutation of S s.t. $b_i <= b_{i+1}$
- Sorting Algorithm Categories
  - Internal sorting: S is small enough to fit in memory/network
    - We concentrate on this
  - External sorting: S partly stored on external device (disk)
  - Comparison sorting: Uses compares and exchanges
    - $\Omega(n \log(n))$ work
  - Non comparison sorting: Uses extra information about input data
    - Values lie in a small range (Radix Sort)
    - S is permutation of (1 .. N) (Pigeon hole sort)
    - Sometimes $\Omega(n)$ work

Storage for Input and Output

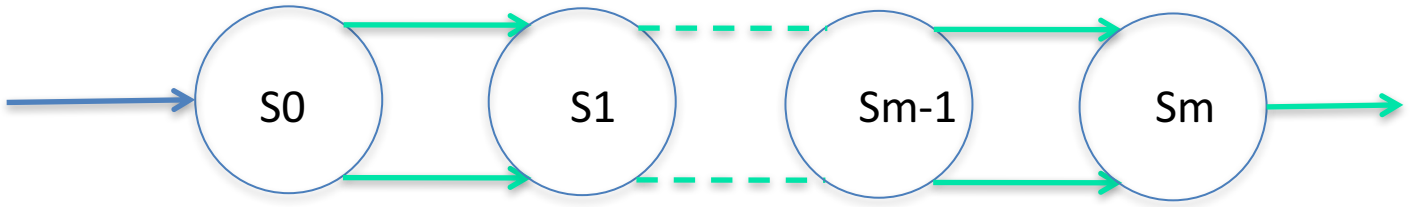
- Input sequence
  - Distributed in equal blocks over p processors (PEs) unless specified otherwise
- Output sequence
  - Distributed over p PEs unless specified otherwise
  - Ordering of blocks S_1 and S_2
    - $S_1 \leq S_2$ iff $\forall\, s_1 \in S_1,\ \forall\, s_2 \in S_2 : s_1 \leq s_2$
    - requires enumeration of processors
  - Sorted output
    - $PE_i < PE_j$ (according to some enumeration) $\Rightarrow$ block(PE_i) $\leq$ block(PE_j)



Discussion: Parallel Merge Sort

- A pipeline of sorters $S_0, S_1 \ldots S_n$
- S_0:
  - One input stream, two output streams
  - reads input stream and creates "sorted" subsequences of size 1
  - sends the subsequences to its outputs (alternating between the two)
- S_i: $(i = 1 .. n-1)$
  - Two input streams, two output streams
  - merges sorted subsequences from two input streams
  - sends double-sized, merged subsequences to its outputs (again alternating)
- S_n:
  - Two input streams, one output stream
  - merges sorted subsequences from two inputs into one result

Pipeline Mergesort

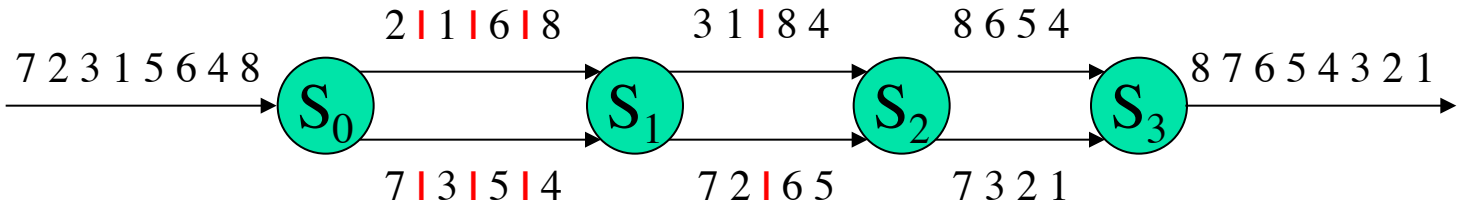


S0 reads the input stream, creates "sorted" sub-sequences of size one and sends these intermittently to one of its two output streams. **Si** repeatedly reads two sorted sub-sequences, one from each input stream, merges them, and writes the double sized sorted sub-sequences intermittently two one of its output streams. **Sm** reads two sorted sub-sequences, one from each input stream, merges these and produces the resulting output sequence.

If a read and a write each take one time step and internal computation takes 0 time:

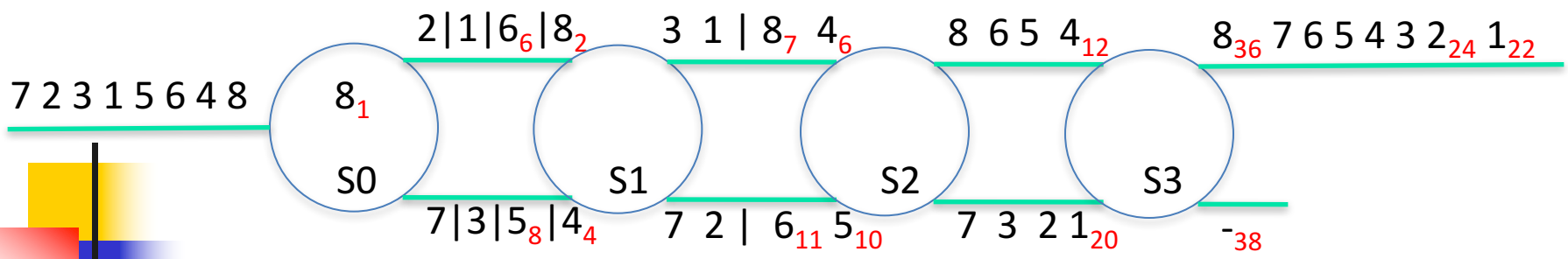
1. **How many sorters are needed to sort n numbers?**
2. **How many time steps does it take to sort n numbers?**
3. **Is this algorithm cost optimal?**

Pipeline Merge Sort (cont.)

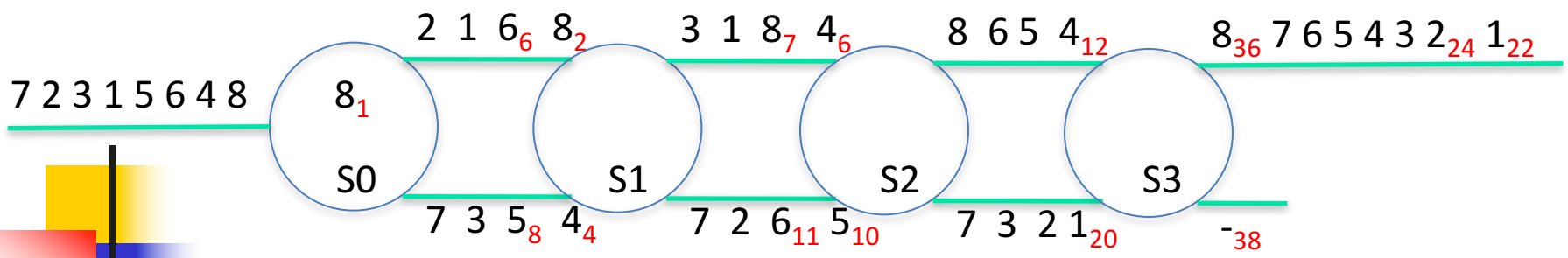


Questions:

1. Given $n = 2^m$ input numbers, how many sorters are needed?
2. If a sorter can read one number in one time step, write one number in one time step, and store and compare in zero time steps, how many time steps does it take to sort n numbers?
3. Is this algorithm cost optimal?



- Assumptions
  - sorter processes are internally sequential
  - only inter process //ism
  - channels are sequential streams
- Questions
  - what triggers S_{i+1} to start?
  - when does it start writing to its bottom channel?



- what triggers S_{i+1} to start?

  Si writing its first number to the bottom channel (BSi)

- when does it start writing to its bottom channel?

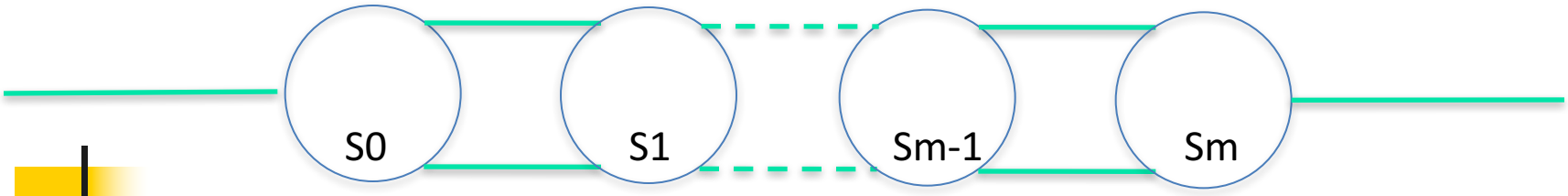
  $O(2^i)$ time steps later

BS_i: start times S_i writing to bottom channel:

BS_0: 4, BS_1:10, BS_2:20, BS_3:38 (and that's the end)

Writing all numbers to the upper channel takes $O(2^i)$

$BS_i = BS_{i-1} + 2^{i+1} + 2 \rightarrow BS_m = \sum_{i=0}^{m} 2^{i+1} + 2 = O(2^m + m) = O(2^m)$



1. How many sorters are needed to sort n numbers?

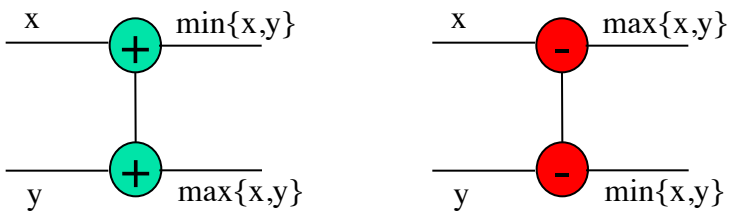
   $m = \log(n)+1 = O(\log(n))$

2. How many time steps does it take to sort n numbers?

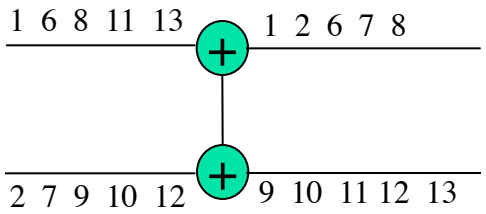
   $BSm = O(n)$

Sorting networks: Compare Exchange, Compare Split

- Compare exchange: n = p
  - Ascending (+)
  - descending (-)



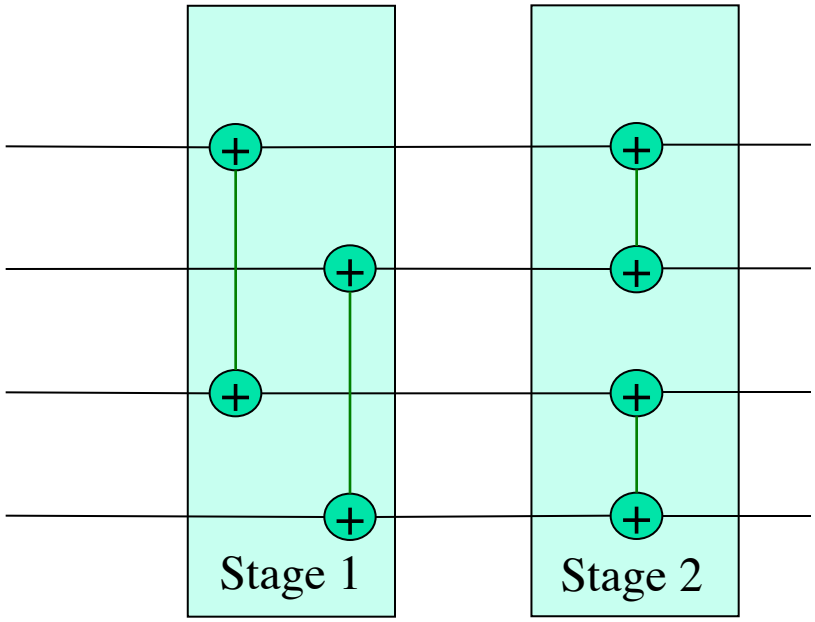
- Compare split: n > p
  - P_i and P_j have blocks of data
  - Merge the two blocks
  - P_i gets lower half, P_j gets upper half

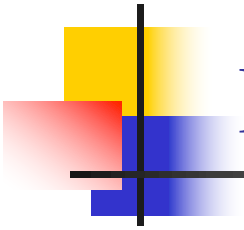


Sorting networks

- n numbers, n lines, M stages
- Each stage:
  - <= n/2 compare-exchanges
  - Each compare exchange computes in O(1) time
- time complexity: M
- Cost: M*n
- does this network sort?

  (1,3,4,2) (2,3,4,1) (3,4,2,1)…





Bitonic Sequence

- A sequence A = $a_0, a_1, \ldots, a_{n-1}$ is bitonic iff

1. There is an index i, $0 < i < n$, s.t.

$a_0 .. a_i$ is increasing

and

$a_i .. a_{n-1}$ is decreasing

or 2. There is a cyclic shift of A for which 1 holds.

Why is it called BI-tonic?

Bitonic Split

- A bitonic split divides a bitonic sequence in two:

BitSplit(BS)= $\nearrow$ S1= ($\min(bs_0, bs_{n/2})$, $\min(bs_1, bs_{n/2+1})$, .. , $\min(bs_{n/2-1}, bs_{n-1})$)

BitSplit(BS)= $\searrow$ S2 = ($\max(bs_0, bs_{n/2})$, $\max(bs_1, bs_{n/2+1})$, .. , $\max(bs_{n/2-1}, bs_{n-1})$)

- Theorem :

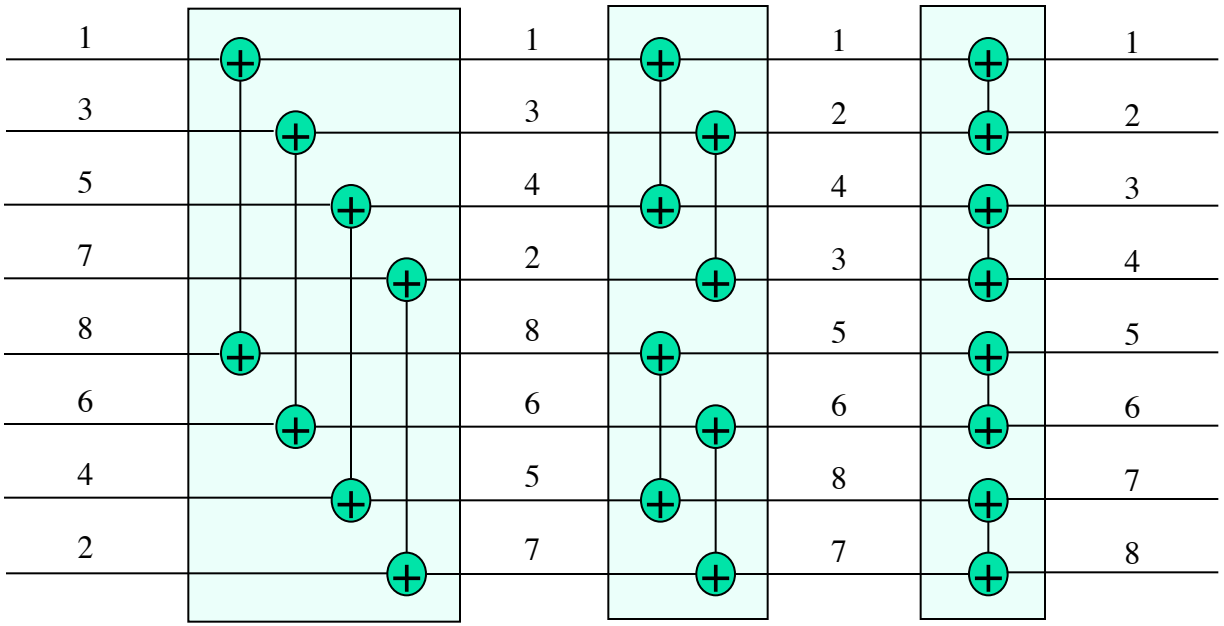
  S1 and S2 are both bitonic and S1 < S2

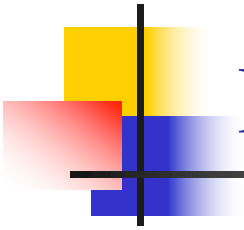
  Proof:

  By intimidation ☺ i.e. consideration of all cases

Bitonic Merge

- Given: a Bitonic Sequence BS of size $n = 2^m$
- Sort BS using m (=log n) Bitonic Split stages

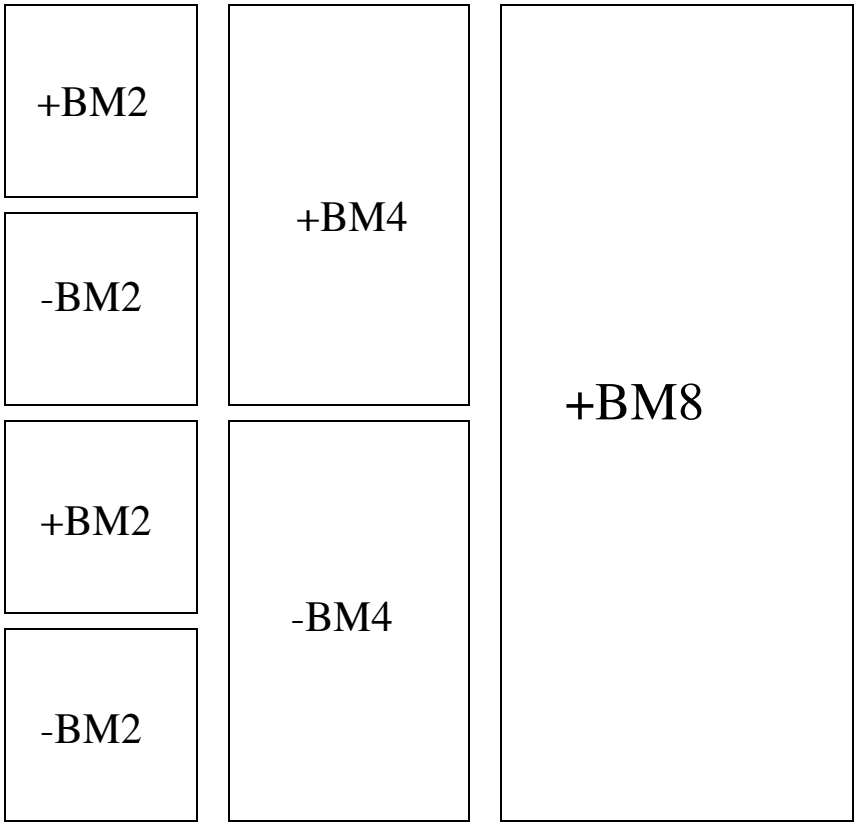




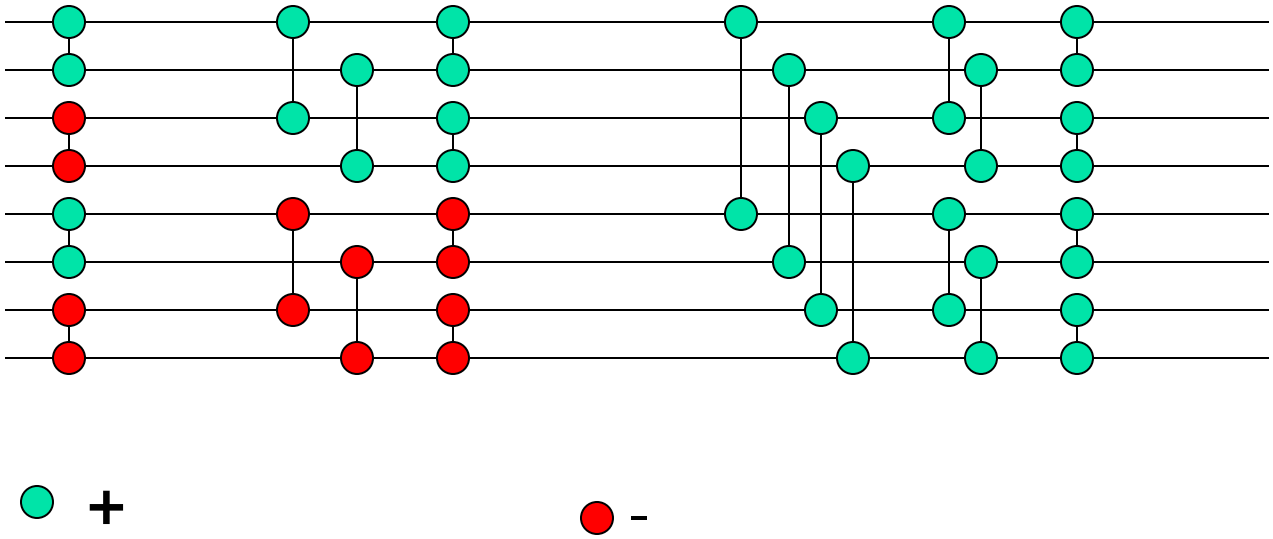
Recursive leap: Bitonic Sort

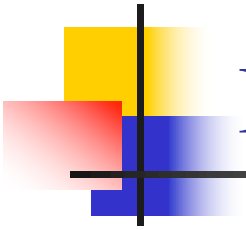
- Each 2 element subsequence is bitonic
- Merging 2 element subsequences, up and down, creates bitonic subsequences of size 4
  - Merging 2 elements up: +BM2
  - Merging 2 elements down: -BM2
- Merging these 4 sized subsequences up (+BM4) and down (-BM4) creates bitonic subsequences of size 8
- and so on.....

Bitonic sort = log(n) bitonic merge stages



Bitonic Sort network





Bitonic sort: time and work

- Time: $O(\log^2(n))$

  Number of stages:

  $B2+B4+B8+\ldots+B2^m = 1 + 2 + 3 + .. + m$

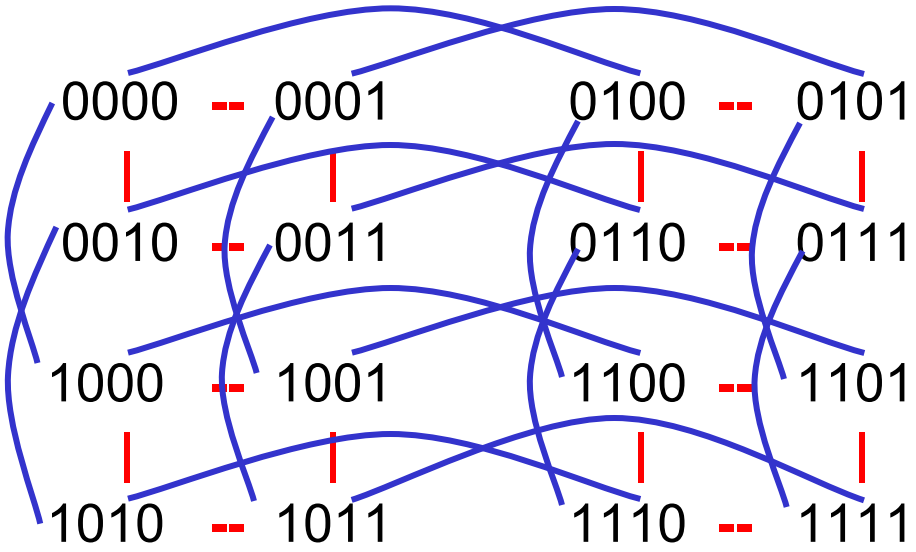
  where $m = \log(n)$

- Work: $O(n \log^2(n))$

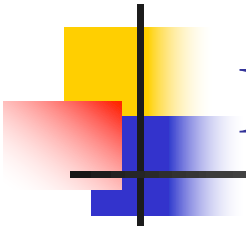
  $O(n)$ per stage

Bitonic Sort on Mesh

- No ideal mapping; best: nearest = most used

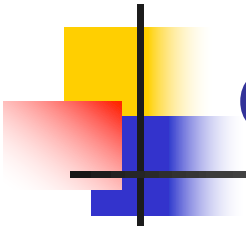


Distance 1: used 7 times, Distance 2: used 3 times



Bitonic Sort n > p

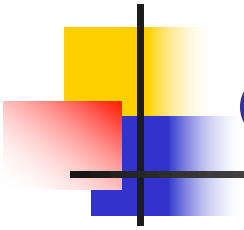
- n/p elements per PE
- Do local sorts at the beginning
- Use compare-split instead of compare-exchange
- Perfect load balance



Count/Radix/Bucket family

- Enumeration Sort
  - Determine rank of every element
  - Sort A[0..n-1], using counters C[0..n-1]

```
forall i in 0..n-1 C[i]=0
forall i in 0..n-1, forall j in 0..n-1
    if A[i]<A[j] or (A[i]==A[j] and i<j) C[j]++
forall i in 0..n-1 S[C[i]]=A[i]
```



Count sort: large number of small numbers

- n numbers in range 0..r-1
  - n >> r

```
forall i in 0..r  C[i]=0
forall i in 0..n-1 C[A[i]+1]++
PPC = ParallelPrefixSum( C )
forall i in 0..n-1 S[ PPC[A[i]]++ ]=A[i]
```

- One of the fastest sorts for this case

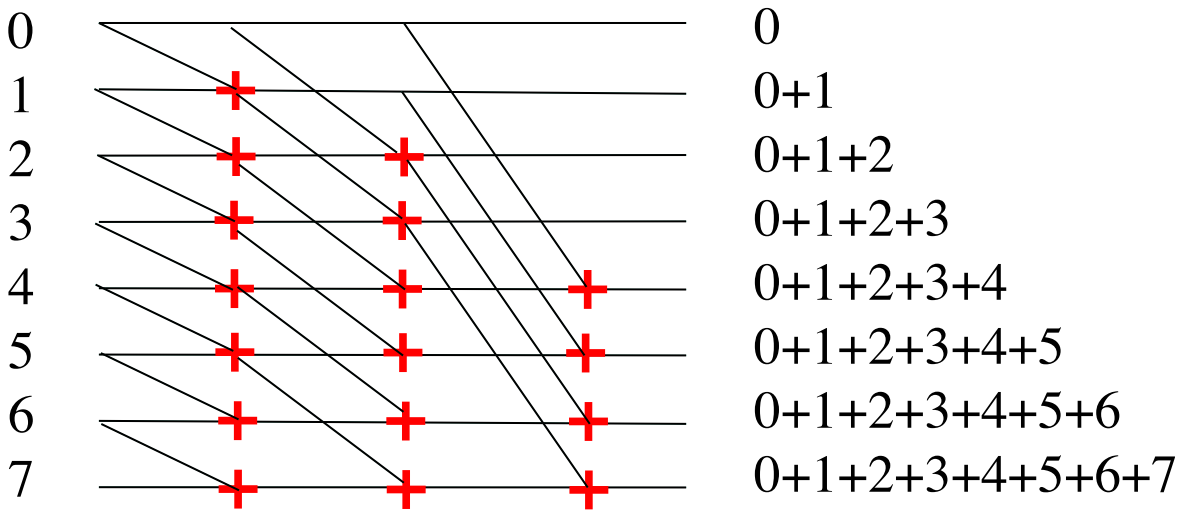
Partial sums, or Parallel Prefix

N numbers V_0 to V_{n-1} stored in A[1] to A[n]

Compute all partial sums (V_1+..+V_k)

```
d = 1
do log(n) times
    for all i in 0..n-1:
        if (i-d) >= 0 A[i] = A[i]+A[i-d]
    d *= 2
```

Parallel Prefix



See: Parallel Prefix with CUDA
https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html