Lesson 2-5 Distributed Memory Sorting

Bitonic Sort

The pseudo code for generating a bitonic sequence:

```
genBitonic(A[0:n-1])
  if n ≥ 2 then
    // assume 2|n
    spawn genBitonic(A[0:n/2-1])
    genBitonic(A[n/2:n-1])
    sync
    spawn bitonicMerge+(A[0:n/2-1])
    bitonicMerge-(A[n/2:n-1])
```

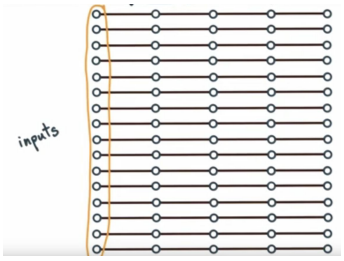
```
bitonicSort(A[0:n-1])
  genBitonic(A[:])
  bitonicMerge+(A[:])
```

1. create two bitonic subsequences
2. Make one an increasing sequence
3. Make one into a decreasing sequence

Work for a bitonic sort: $W_{bs}(n) = \theta(n \log^2(n))$ -- this algorithm is not work optimal
Span for a bitonic sort: $D_{bs}(n) = \theta(\log^2(n))$

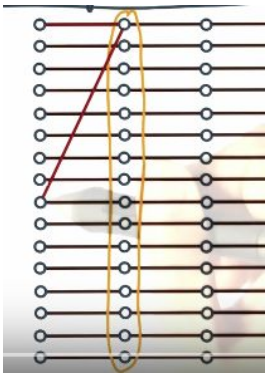
Distributed Bitonic Merge via Binary Exchange

With regards to bitonic sorts, everything boils down to doing bitonic sorts efficiently.

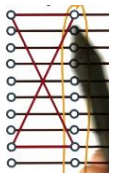


Step 1: a bitonic split, performed in place
At the end of stage 1 there are two bitonic subsequences

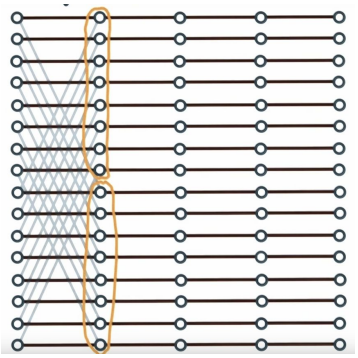
For a distributed algorithm, divide the processes between the nodes.



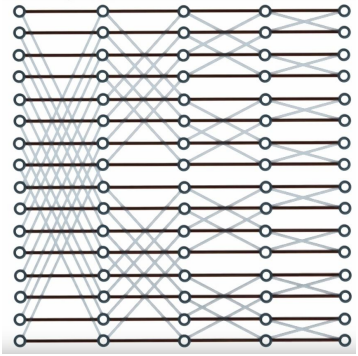
This shows the minimum between the two inputs. (The start of the split.)



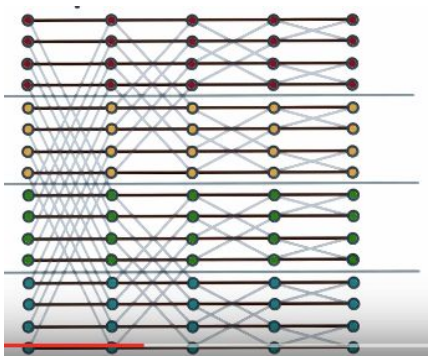
The second red line show the maximum between the two inputs. The edges indicate the dependence between the two inputs and outputs.



The end result is a bitonic split.

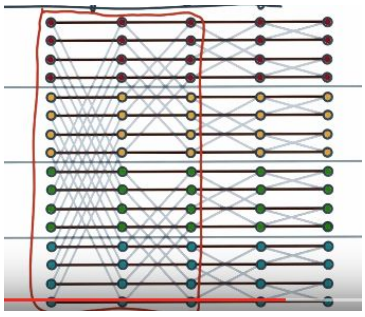


This shows the inputs, the outputs, and the patterns of dependencies.



To look at the distributed version of the this:
Using 4 processing nodes.

In this case, communication
Communication occurs anywhere a dependence edge crosses a process boundary.
Binary exchange= two processes swapping data with each other.



Note that all communication occurs in the first log(p) stages.

There are only log(p) stages that require communication.
In the other log(n/P) stages there is no need for communication between processes.

In a cyclic distribution the rows of the network are assigned to different processes in a round-robin fashion. The communication is similar to the block scheme.

Pick a Network

Which network would allow for fully concurrent exchanges without congestion?
Hypercube and fully connected topologies.
This is because you need a network with a linear or better bisection width. The fully connected network is more than necessary.

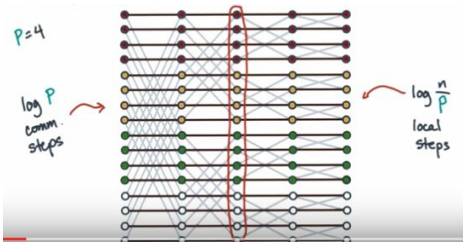
Communication Cost of a Bitonic Merge

What is the communication time of a bitonic merge, assuming a block-distributed, binary-exchange scheme on a hypercube.

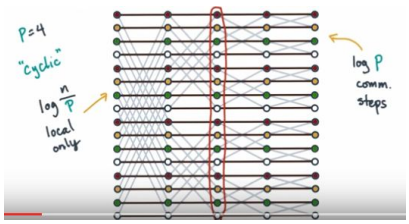
Communication time = (a + (b *(n/p))* log(p))
Recall, the binary exchange scheme only communicates during the first log(P) stages.
Each process has to send n/P words at each stage.

Bitonic Merge via Transposes

Two distributed bitonic merge schemes have been discussed:



Block distribution scheme: log(P) stages of communication and log(n/P) stages of computation.



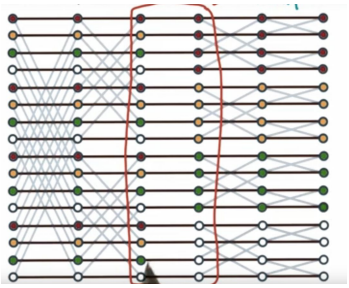
Cyclic scheme:log(n/P) stages of computation and log(P) stages of computation

The running time for the two are the same: $T_{net}(n; P) = \alpha \log(P) + \beta (n/P)\log(P)$

Is there something that will reduce the β term, even at the cost of increasing the α term? YES

Start with a cyclic topology:

This means there is no communication initially. Then switch to block, which means no communication at the end. To make this work the data will have to be transposed (or shuffled).



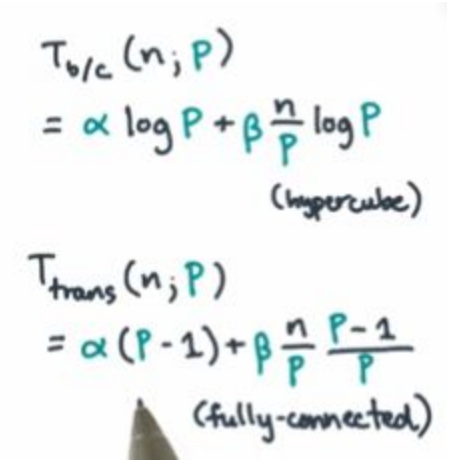
The transpose can be seen as an all-to-all exchange or as a matrix transpose.

If we look at one process note: it needs to send P-1 messages, each of size n/P^2, to all the other processes.

To determine the process time:

1. Assume the network is fully connected.

$T_{trans}(n;P) = \alpha (P-1) + \beta (n/P)((P-1)/P)$ Fully connected network.



There is a latency bandwidth tradeoff between the two schemes.
In practice is it very hard to for the block or cyclic scheme to beat the transpose scheme.

Butterfly Trivia
Name another famous algorithm that follows the same computational pattern. Fast Fourier Transform

Bitonic Sort Cost Computation

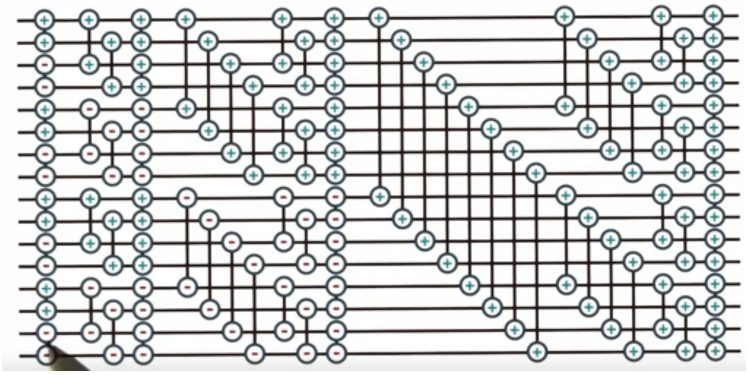
$\tau = cost/compare$

$\tau(n/P)K =$ total time to do the comparisons, at merging stage 'K'

There are log n merging stages, so the total cost for computation is:

$$\sum_{k=1}^{\log n} \tau \frac{n}{p} k = O\left(\tau \frac{n \log^2 n}{p}\right)$$

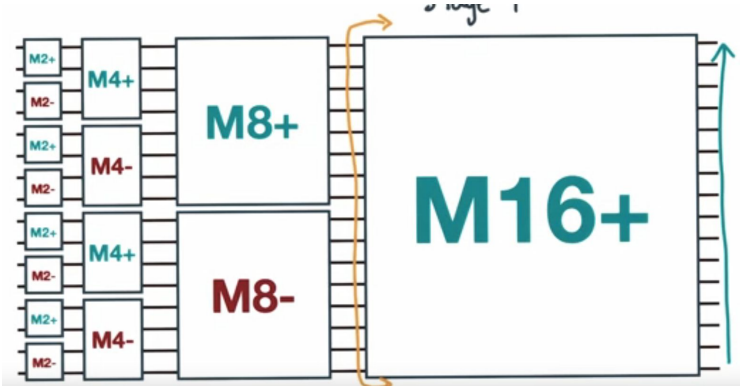
Bitonic Sort Cost Communication



Assume: n, P are powers of n; P/n

$T_{msg}(m) = \alpha + \beta m$

What is the communication time?
The key question: where does communication happen?
For each stage K, the size of the bitonic merge is $n_K = 2^K$



Stage 4 - a single bitonic merge of size 16.

Assume a block distribution with ‘P’ processes.
Communication only starts when K > log(n/P)

$P_K = 2^{K-\log(n/P)}$ … when k == log(n) it is simplified to P

Bitonic Merge in P processes of n size requires

$$T_{merge}(m; P) = \left(\alpha + \beta \frac{m}{P}\right) \log P$$

Time to communicate

So the time for the entire sort is:

$$T_{sort}(n;P) = \sum_{k=\log\frac{n}{P}+1}^{\log n} T_{merge}(n_k;P_k)$$

The simplified version is:

$$O(\alpha \log(P) + \beta(n/P)(\log^2 P))$$

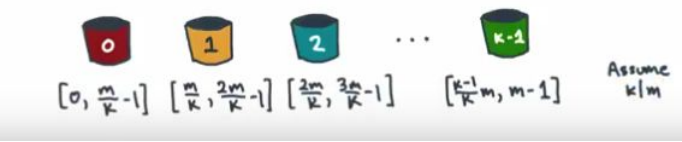
Linear Time Distributed Sort, Part 1

Any comparison based algorithm for sorting scales to: O(n log n)

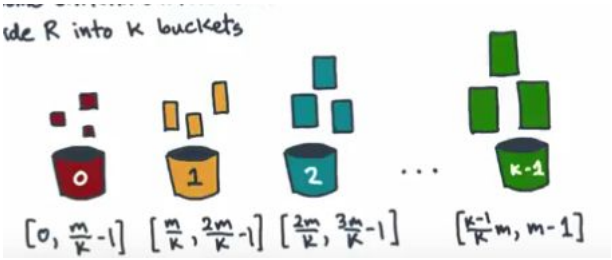
For the bucket sort … O(n)

To do bucket sort:

1. Start by assuming you know the possible values. R = {0,1,2,3,...,m-1}
2. The values to be sorted are uniformly distributed over the range.
3. Divide R into k buckets



4. The bucket sort first figures out which bucket each value belongs.



5. Sort within in each bucket and concatenate the results.

How is this a linear time scheme?
How many elements are in each bucket? the expected number of elements in each bucket will be n/k. (Assume the elements are evenly distributed across the range)

The time to sort each bucket is: O(n/k log (n/k))

$E[\#\text{ elems / bucket}] = \Theta(\frac{n}{k})$ (n = # elems to sort)

$E[\text{time to sort bucket}] = O(\frac{n}{k}\log\frac{n}{k})$

$E[\text{total time}] = O(n \log\frac{n}{k}) = O(n)$ if $k = \Theta(n)$

Distributed Bucket Sort

It is to distribute -- assign each bucket to a compute node.

Make the following assumptions:

- k= P
- elements per node ~n/P elements per table
- Assume all nodes know bucket ranges
- Assume the network is fully connected

The running time is: O(b * (n/P) + t * (n/P) + (a * P))

The steps to get this:

1. Each process needs to scan its list of local elements and decide which elements go where. The nodes can do this in parallel and the work is linear.
2. The nodes need to exchange elements (an all-to-all operation). Each node has ~n/P elements. So expect to send n/P^2 elements to every node. Assume the network is fully connected.
3. Now each bucket must do a local sort, of cost O(n/P)

Steps:

1. Local scan = $\Theta(\tau\frac{n}{P})$
2. All-to-all = $\Theta(\alpha P + \beta\frac{n}{P})$
3. Local sort = $O(n/P)$

Linear-time Distributed Sort, Part 2: Sample Sort

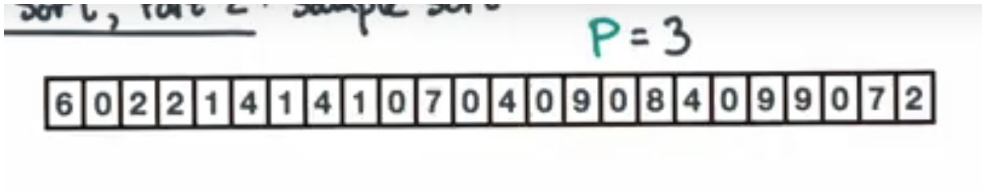
The bucket sort has a major flaw … the assumption of a uniform distribution of values across the buckets. If you do not have uniform dist., you will not get an equal number of elements in each bucket.

So use Sampling….

Do bucket sort, but the intervals vary according to the data. To decide the size of the intervals, use sampling.

To do Sample Sort:

1. Begin with data and, in this case, 3 processes.



2. Assume the elements are equally dist. among the 3 processes:

3. Sort them locally

4. Each process will choose a sample of elements from their list. Each should choose the same equally spaced elements.

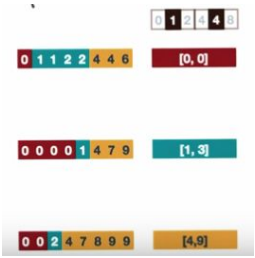
5. Gather the samples in the root.
6. Sort the samples on the root.

7. Select P-1 splitters (in this case 2)

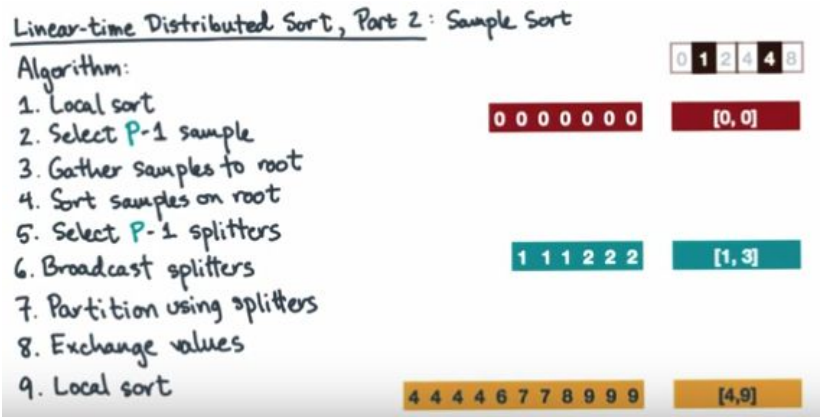
8. The splitters define the global bucket boundaries.
9. For this example:

> The first bucket will get the first ‘split’, the ‘0’ elements. The second bucket will get the second split, 1-3. The third bucket will get the last split, 4 - end.

10. Now the splitters will need to be broadcast.
11. Each node can partition its elements using the splitters.



12. Then the nodes exchange values. Each node will get only the values in its range.
13. Then each node will do a local sort.



In the running time for this sample sort, what is the largest asymptotic function of P? (Assume P processes)

$O(P^2)$ or $O(P^2 \log P)$ …...The root has to sort P^2 samples.

If the system is truly massive, this could be a delimited to scalability.