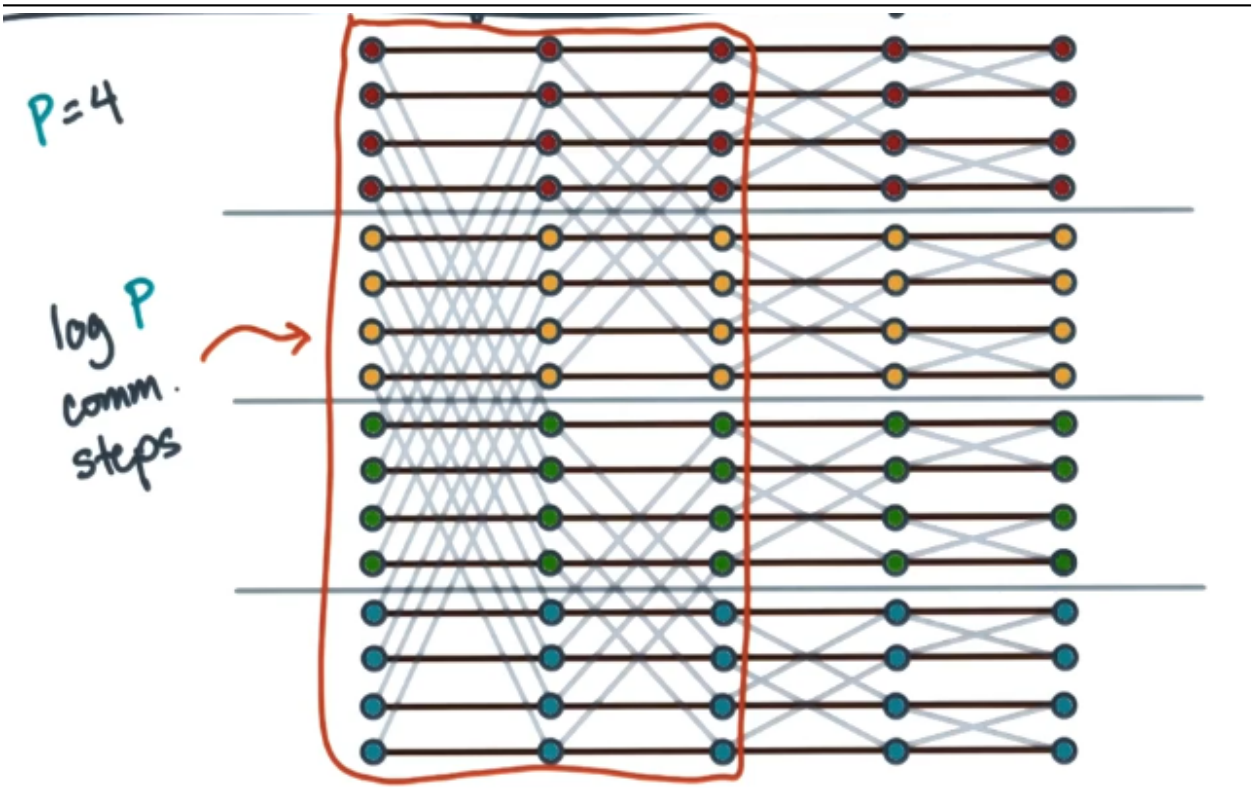
Distributed Memory Sorting

Introduction

1. Covers sorting in a distributed memory setting
   - Sorting is a fundamental primitive
   - Good practice for learning how to reason about algorithms in general

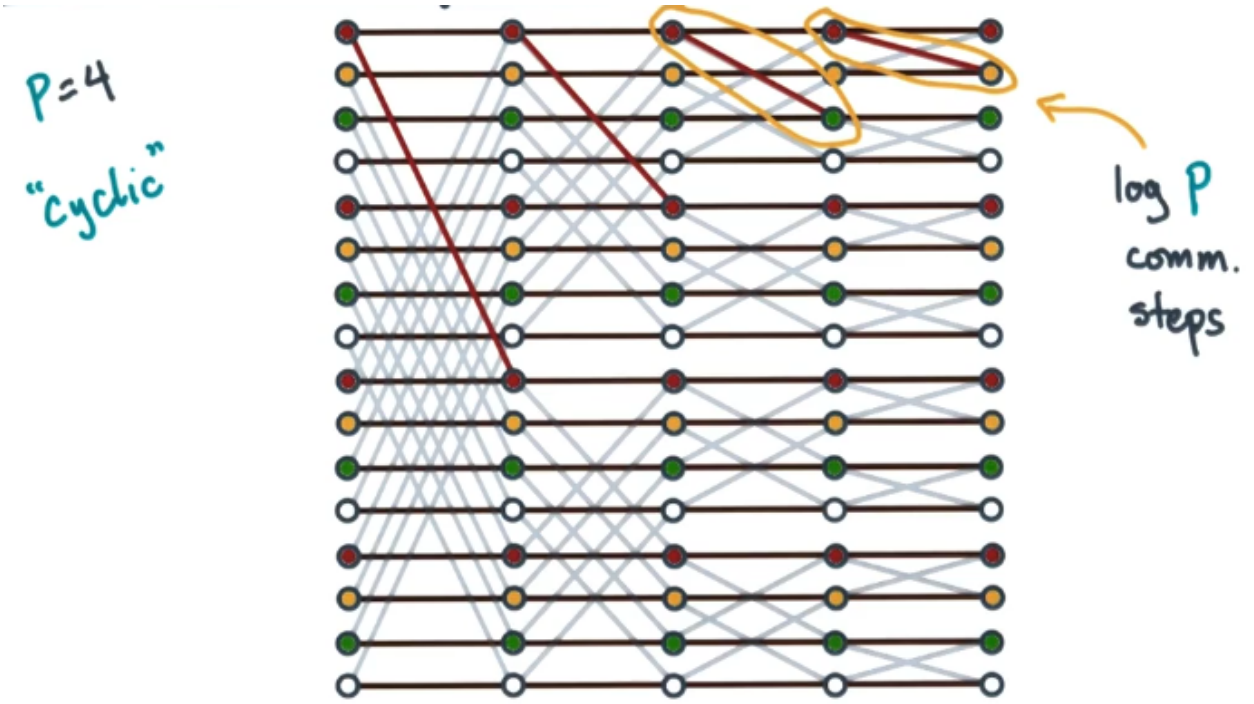
Distributed Bitonic Merge via Binary Exchange

1. Figure below shows the inputs, outputs, and pattern of dependencies in a bitonic merge with a block distribution
   - Divide across processing nodes; communication happens anywhere a dependence edge crosses a process boundary
   - There are log(P) communication steps and log(n/P) local steps



Bitonic Merge (Block)

2. Figure below shows the inputs, outputs, and pattern of dependencies in a bitonic merge with a cyclic distribution
   - In a cyclic distribution, each process gets non-adjacent inputs, but there are still log(P) communication steps and log(n/P) local steps



Bitonic Merge (Cyclic)

Pick a Network

1. Which of these topologies would allow for fully-concurrent exchanges without congestion for a block-distributed bitonic merge? Assume P = n
   - 1D ring
   - 3D torus
   - Hypercube (true)
   - Fully-connected (true)
2. Because the first 8 nodes communicate with the second 8 nodes, we need a network with a linear (or greater) bisection width
   - Fully-connected is overkill because there are additional links we won't use
   - Hypercube is a good fit
   - Butterfly network is exactly this type of topology

Communication Cost of a Bitonic Merge

1. What is the communication time of a bitonic merge, assuming a block-distributed, binary-exchange scheme on a hypercube?
2. (a + Bn/P) * log(P)
   - Each process sends n/P words at each stange
   - Communication only occurs in the first log(P) stages

Bitonic Merge via Transposes

1. Cyclic-distributed and block-distributed bitonic merge have high bandwidth requirements (sending n/P words log(P) times)
2. If we start using the cyclic algorithm (no communication at the beginning) then switch to block (no communication at the end) we can reduce the bandwidth
   - Need to reshuffle the data in between (matrix transpose)

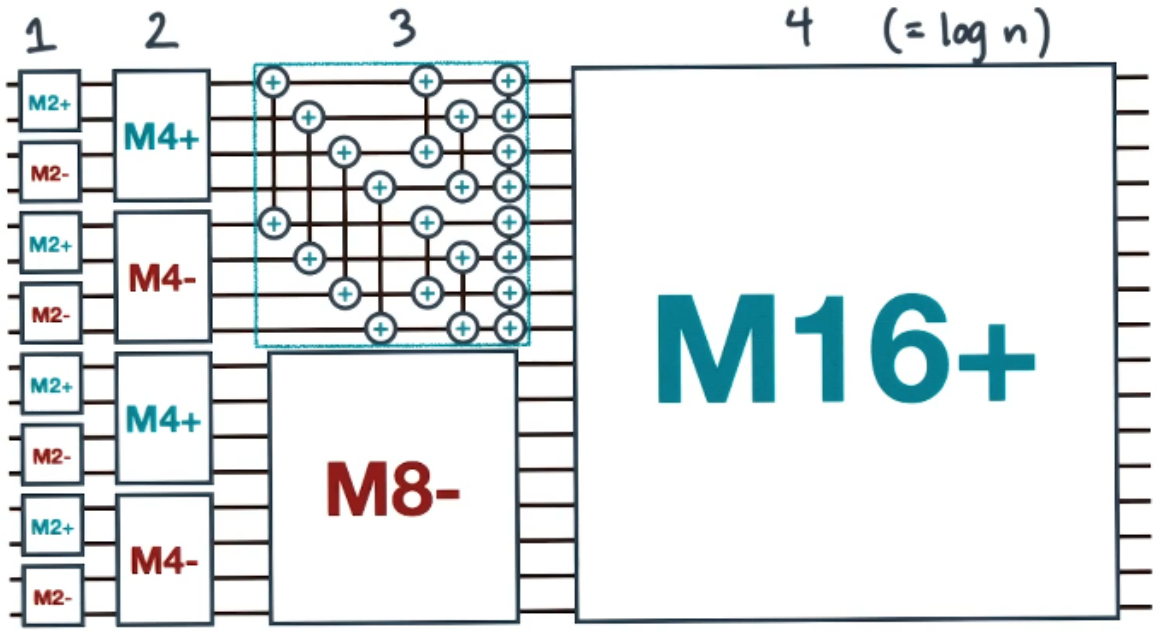
3. If the topology is a hypercube for block or cyclic scheme
   - T(n;P) = alog(P) + Bnlog(P)/P
4. If the topology is fully-connected for transpose scheme
   - T(n;P) = a(P-1) + Bn(P-1)/P^2
5. In practice, it's very difficult for the block or cyclic scheme to outperform the transpose scheme

Butterfly Trivia

1. Name another algorithm that follows the same computational pattern as the bitonic merge transpose scheme

Bitonic Sort Cost Computation

1. In a bitonic merge, there are log(P) stages, each performing a bitonic merge



Bitonic Merge (Cyclic)

2. Suppose there are P processes and a block-distribution scheme. In this case, P=4. Consider stage k
   - 2^k simultaneous merges occur
   - Each process own n/P elements
   - kth merging stage performs kn/P comparisons costing tau units
     - tau * nk/P
   - Sum this cost over log(n) merging stages
     - O(tau * nlog^2(n)/P)
     - Not work-optimal, but perfectly parallelizable

Bitonic Sort Cost Communication

1. Tcomp = O(tau * nlog^2(n)/P)
   - Tmsg(m) = a + Bm
   - Assume n, P are powers of 2; P | n, block-distributed scheme
2. What is the communication time?
   - O(alog(P) + Bn/P * log^2(P))

- nk = 2^k
- P processes
- Communicate only when k > log(n/P)
- Pk = 2^(k-log(n/P))

Linear Time Distributed Sort - Part 1

1. Comparison-based sort is O(nlog(n))
   - Can't go faster if your only primitive is a simple comparison
2. Bucket sort: O(n)
   - Assume a range of possible values: R = {0, 1, 2, . . . , m-1}
   - Assume the values are uniformly distributed
   - Divide R into k buckets, assuming k | m
   - Then, sort the elements into each bucket and sort each bucket
     - Expect n/k elements per bucket
   - E[#elems/bucket] = O(n/k)
   - E[time to sort bucket] = O(n/k * log(n/k))
   - E[total time] = O(n * log(n/k)) = O(n) if k == O(n)

Distributed Bucket Sort

1. What is the running time of bucket sort? Assume tau, a, B costs. Also assume k = P, ~n/P elements per node, and all nodes know bucket ranges
2. Steps:
   - Each node scans its list of local elements and decides which element goes where
     - O(tau * n/P)
   - Each node sends the values to the node with the correct bucket (all-to-all)
     - This means n/P^2 elements to every node
     - Assume fully-connected network
     - O(aP + Bn/P)
   - Local sort
     - O(n/P)
   - Total: tau * n/P + aP + Bn/P

Linear Time Distributed Sort - Part 2

1. Bucket sort is a neat idea, but its assumption of an underlying uniform distribution is a critical flaw
2. All linear sorting algorithms that run at scale use some sort of sample sort
   - Instead of linear-width buckets, let the width of the buckets vary according to the data
     - Use sampling to decide the widths
3. Assume the algorithms are equally distributed across the processes
   - Locally sort the elements
   - Select a sample of P-1 elements; choose these to be equally spaced across the sorted list
   - Gather the samples to the root process
   - Sort the samples on the root proccess
   - Select P-1 splitters
   - Broadcast the splitters
   - Partition using splitters
   - Exchange values
   - Do another local sort

Cost of Distributed Sample Sort

1. In the running time for this sample sort, what is the largest asymptotic function of P?
   - O(1)

- O(log(P))
- O(logˆ2(P))
- O(P)
- O(Plog(P))
- O(Plogˆ2(P))
- O(Pˆ2) (true)
- O(Pˆ2 * log(P)) (true)

2. The root must perform a local sort of the samples; this is either Pˆ2 or Pˆ2 * log(P) depending on the underlying sorting algorithm

Conclusion

1. Annual contest for implementing the fastest sort (sortbenchmark.org)
   - In 2014, fastest sort sorted 7TB in a minute
   - In 1995, fastest sort sorted 1GB in a minute
   - 7000x improvement over 20 years