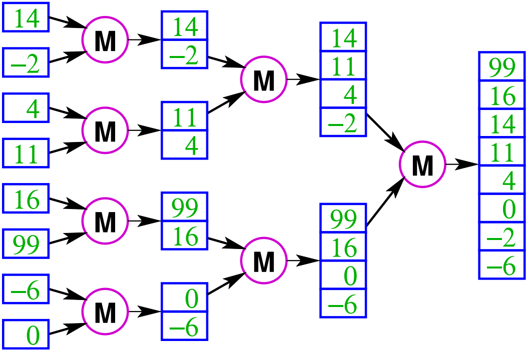
Bitonic Sort

Mark Greenstreet

CpSc 418 – Feb. 10, 2017

- I know that some of the links in the electronic version are broken. I know that it would be nice if I complete the final slides. I will post to piazza when this is done.

Parallelizing Mergesort



- We looked at this in the Feb. 8 lecture.
- The challenge is the merge step:
  - Can we make a parallel merge?

Merging and the 0-1 Principle

Easy cases

A	*B*	*A*	*B*	*A*	*B*
1	1	1	1	1	1
1	1	1	1	1	1
1	0	0	1	1	1
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

The main idea:

- Use divide-and-conquer.
  - Given two arrays, *A* and *B*, divide them into smaller arrays that we can merge, and then easily combine the results.
  - What criterion should we use for dividing the arrays?
- Observation:
  - It's easy to merge two arrays of the same size, if they both have the same number of **1s**.
  - If they have **nearly** the same number of **1s**, that's easy as well.

Dividing the problem (part 1)

- For simplicity, assume each array has an even number of elements.
  - As we go on, we'll assume that each array has an power-of-two number of elements.
  - That's the easiest way to explain bitonic sort.
  - Note: the algorithm works for arbitrary array sizes.
    - See the lecture slides from 2013.
- Divide each array in the middle?
  - If A has N elements and N_1 are ones,
  - How many ones are in $A[0, \ldots, (N/2) - 1]$?
  - How many ones are in $A[N/2, \ldots, N - 1]$?
- Taking every other element?
  - How many ones are in the $A[0, 2, \ldots, N - 2]$?
  - How many ones are in the $A[1, 3, \ldots, N - 1]$?
- Other schemes?

Dividing the problem (part 2)

- Let A and B be arrays that are sorted into ascending order.
  - Let A_0 be the odd-indexed element of A and A_1 be the odd-indexed.
  - Likewise for B_0 and B_1.
- Key observations:

$$\begin{array}{lclcl} \text{HowManyOnes}(A_0) & \leq & \text{HowManyOnes}(A_1) & \leq & \text{HowManyOnes}(A_0)+1 \\ \text{HowManyOnes}(B_0) & \leq & \text{HowManyOnes}(B_1) & \leq & \text{HowManyOnes}(B_0)+1 \end{array}$$

- With a bit of algebra, we get

$$\left|\text{HowManyOnes}(A_0 \mathbin{++} B_1) - \text{HowManyOnes}(A_1 \mathbin{++} B_0)\right| \quad \leq \quad 1$$

- In English that says that
  - If we merge A_0 with B_1 to get C_0,
  - and we merge A_1 with B_0 to get C_1,
  - then C_0 and C_1 differ by at most one in the number of ones that they have.
    - ★ This is an "easy" case from slide 3.

Merging

- Given N that is a power of 2, and arrays A and B that each have N elements and are sorted into ascending order, we can merge them with a sorting network.
- If $N = 1$, then just do `CompareAndSwap(`A`, `B`)`.
- Otherwise, let A_0 be the odd-indexed element of A and A_1 be the odd-indexed, and likewise for B_0 and B_1.
- Merge A_0 and B_1 into a single ascending sequence, C_0.
- Merge A_1 and B_0 into a single ascending sequence, C_1.
  - Note that the number of ones in C_0 and C_1 differ by at most one.
- Merge C_0 and C_1 into a single ascending sequence.
  - This is an "easy" case from slide 3.
  - We can perform this merge using $N/2$ compare-and-swap modules.
- Complexity:
  - Depth: $O(\log N)$ – logarithmic parallel time.
  - Number of compare-and-swap modules $O(N \log N)$.
- **Pause:** If you understand this, you've got all of the key ideas of bitonic sorting.
  - The bitonic approach just improves on this simple algorithm.

Bitonic Sequences

- A sequence is **bitonic** if it consists of a monotonically increasing sequence followed by a monotonically decreasing sequence.
  - Either of those sub-sequences can be empty.
  - We'll also consider a monotonically decreasing followed by monotonically increasing sequence to be bitonic.
- Properties of bitonic sequence
  - Any subsequence of a bitonic sequence is bitonic.
  - Let A be a bitonic sequence consisting of **0s** and **1s**. Let A_0 and A_1 be the even- and odd-indexed subsequences of A.
  - The number of **1s** in A_0 and A_1 differ by at most 1.
    - ★ We'll examine the number of **0s** on slide 10.

Bitonic Merge – big picture

- Bitonic merge produces a monotonic sequence from an bitonic input.
- Given two sorted sequences, A and B, note that

$$X = A \mathbin{++} \text{reverse}(B)$$

  is bitonic.
  - We don't require the lengths of A or B to be powers of two.
  - If fact, we don't even require that A and B have the same length.
- Divide X into X_0 and X_1, the even-indexed and odd-indexed subsequences.
  - X_0 and X_1 are both bitonic.
  - The number of **1s** in X_0 and X_1 differ by at most 1.
- Use bitonic merge (recursion) to sort X_0 and X_1 into ascending order to get Y_0 and Y_1.
  - $\text{HowManyOnes}(Y_0) = \text{HowManyOnes}(X_0)$, and $\text{HowManyOnes}(Y_1) = \text{HowManyOnes}(X_1)$.
  - Therefore, the number of **1s** in Y_0 and Y_1 differ by at most 1.
  - This is an "easy" case from slide 3.

Counting the 0s and 1s (even total length)

X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1
0	0	0	0	0	0	0	0	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	0	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1

- First, we'll look at the case when length(A ++ B) is even.
- Given two sorted sequences, A and B, let

$$\begin{aligned} X_0 &= \text{EvenIndexed}(A \text{ ++ reverse}(B)) \\ X_1 &= \text{OddIndexed}(A \text{ ++ reverse}(B)) \end{aligned}$$

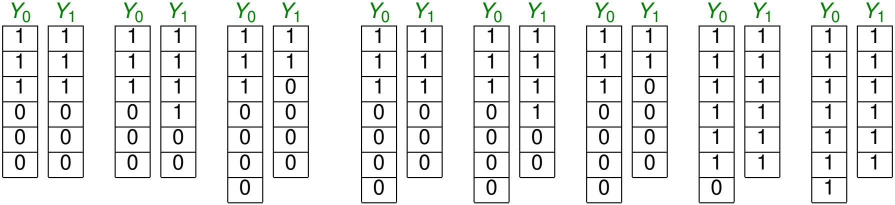
  - This means that $X[i] = X_{i \bmod 2}[i \text{ div } 2]$.
  - In English, the elements of X go left-to-right and then bottom-to-top in X_0 and X_1.
- The number of **1s** in X_0 and the number of ones in X_1 differ by at most 1.
- Likewise for the number of **0s**.

Counting the 0s and 1s (odd total length)

X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1	X_0	X_1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	0	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0		0		0		0		0		0		0		1	

- Let $N = \text{length}(A \mathbin{++} B)$, where N is odd.
- The number of **1s** in X_0 and the number of ones in X_1 differ by at most 1.
- The number of **0s** in $X_0[1, \ldots, \lfloor N/2 \rfloor]$ and the number of zeros in X_1 differ by at most 1.
- Either $X_0[0]$ or $X_0[\lfloor N/2 \rfloor]$ is the **least** element of $A \mathbin{++} B$.

After applying bitonic merge to X_0 and Y_0



- Let $N = \text{length}(A \mathbin{++} B)$.
-
- If N is even,
  - Any out of order elements are in the same row, i.e. $X_0[i] > X_1[i]$ for some $0 \leq i < N/2$.
- If N is odd
  - Any out of order elements are of the form $X_0[i+1] > X_1[i]$ for some $0 \leq i < N/2$.
  - $X_0[0]$ is the least element of X_0 and X_1.

The complexity of bitonic merge

- We'll count the compare-and-swap operations
  - Is it OK to ignore reversing one array, concatenating the arrays, separating the even- and odd-indexed elements, and recombining them later?
  - Yes. The number of these operations is proportional to the number of compare-and-swaps
  - Yes. Even better, in the next lecture, we'll show how to eliminate most of these data-shuffling operations.
- A bitonic merge of N elements requires:
  - two bitonic merges of $N/2$ items (if $N > 2$)
  - $\lfloor N/2 \rfloor$ compare-and-swap operations.
- The total number of compare and swap operations is $O(N \log N)$.

Bitonic-Sort, and it's complexity

Shuffle and unshuffle

- Shuffle is like what you can do with a deck of cards:
  - Divide the deck in half
  - Select cards alternately from the two halves.
  - Shuffle is a circular-right-shift of the index bits.
    - Assuming the number of cards in the deck is a power of two.
- Unshuffle is the inverse of shuffle.
  - Unshuffling a deck of cards is dealing to two players.
  - Unshuffle is a circular-left-shift of the index bits.