EE 382V: Parallel Algorithms **Summer 2017**

Lecture 8: June 16

Lecturer: Vijay Garg *Scribe: Andy Yang*

8.1 Introduction

In this section, we will be discussing sorting algorithms. Sorting can be separated into two categories: comparison-based, and non-comparison-based. Comparison-based algorithms use method such as $compare(A[i], A[j])$ where element at i location is compared with element at j location of the same array A.

The following describes time complexity of insertion sort, quicksort, and mergesort.

	worst-case time	average-case time
Insertion sort	$O(n^2)$	$O(n^2)$
Quicksort	$O(n^2)$	$O(n \times \log(n))$
Mergesort	$O(n \times \log(n))$	$O(n \times \log(n))$

Typically, quicksort is faster than mergesort. There are no comparison-based algorithms that can do better than $O(n \times \log(n))$ time, which is the lower bound for both worst-case and average-case scenarios.

If you have an unsorted array, the input to a sort has a permutation of $n!$. Each comparison eliminates half of the permutations. Therefore, the number of comparisons or the height of the tree would be

$$height \geq O(log(n!)) = O(n \times \log(n))$$

For parallel algorithms, we can do a lot of comparison at a same time.

8.2 Odd-even Sort

Odd-even sort is sometimes called brick sort. The idea is compare elements in pairwise. During the first pass of the comparisons, the elements at odd indices are compared with their successors and are switched accordingly. During second pass of the comparisons, the elements at even indices are compared with their successors and are switched accordingly. Repeat the above process until after a set of odd and even comparisons, there are no more switching of elements in the array and is considered done. Notice that odd and even comparisons are considered as one step. It requires CRCW PRAM, but what we want is CREW or EREW. What if the smallest number in the array is at the most right side? We will run n^2 times, and be done.

Given an unsorted array:
3 11 2 9 6 1 5 8

After odd entries comparisons:
3 11 2 9 1 6 5 8

After even entries comparisons:
3 2 11 1 9 5 6 8

After odd entries comparisons:
2 3 1 11 5 9 6 8

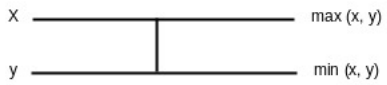
After even entries comparisons:
2 1 3 5 11 6 9 8

...

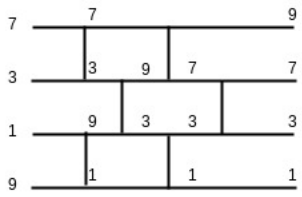
	Time	Work
Odd-even sort	$O(n)$	$O(n^2)$

8.2.1 Oblivious Sorting Algorithm

Oblivious sorting algorithm makes the exact same comparisons. The same entries in the array will be compared irrespective of the actual entries. Have a comparitor takes two inputs x, y, and generates $max(x, y)$ and $min(x, y)$. Even the entries are sorted, they still need to go through the comparitors.



Odd-even sort is an oblivious sorting algorithm. It can be represented graphically as below:



Time is corresponding to the depth of the circuit, or length of the wall. Work is corresponding to the number of comparitors.

8.3 Bitonic Sort (Batcher Sort)

In mergesort, we divide the array into two parts, and sort them individually and merge, but merging is not easy. Instead of sorting them in the same direction, bitonic sort sorts them one in ascending direction and the other in descending direction, and sort them recursively. The first half would be in ascending order, and the second half would be in descending order.

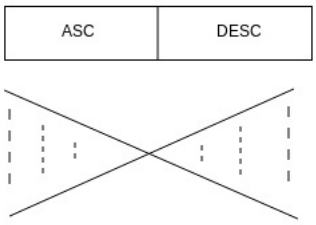
Mergesort sorts sequentially merge one things at a time, but we want to compare multiple elements at the time.

We will do left bitonic sort, right bitonic sort, and do bitonic merge. The bigger element go up, and the smaller element go down. We would have upper and lower part, and the upper part is all bigger than elements in the lower part.

In $O(logn)$ step, we will be done since we recursively divide up the array.

8.3.1 Bitonic Sequence

Monotonic sequence is either in descreasing order or increasing order. Bitonic sequence is either first part is increasing and the second part is descreasing or the first part is descreasing and the second part is increasing. The following shows graphical representation of the bitonic sequence and its element comparisons.



Think of bitonic sequence as two separate arrays. Given a bitonic sequence A, where h indicates the location of the largest element and lo indicates the location of the smallest element.

			h		lo	
3	4	5	7	6	1	2

There exist two monotonic sequences $A[h..lo]$ and $A[lo..h]$ with the second array rotating to get the bitonic sequence.

Definition: A sequence $A[0]...A[n-1]$ is bitonic if there exist two indices l and h such that $A[l..h]$ is an ascending sequence and $A[h..l]$ is descending sequence.

8.3.2 Zero-one Principle

Theorem: If a comparator network correctly sorts all sequences 0 and 1, then it correctly sorts all integer sequences.

Proof: By contradiction, if a comparator network correctly sorts all $0-1$ sequence but did not correctly sort some of the integer sequence.

Bitonic $0-1$ sequences are shown as the follows, given that $j, k, l \geq 0$

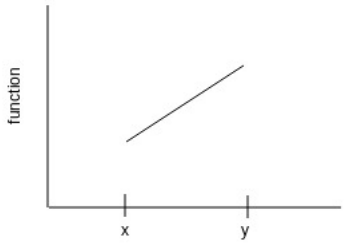
$0^j \quad 1^k \quad 0^l$

$1^j \quad 0^k \quad 1^l$

8.3.3 Monotonic Function

A monotonic function is a function that preserves the concept of order such as $3 \leq 5$. Instead of using x_1 as input, apply $f(x_1)$, the resulting ordering would be same as if input were x_1.

A function is monotonic iff $x \leq y -> f(x) \leq f(y)$.

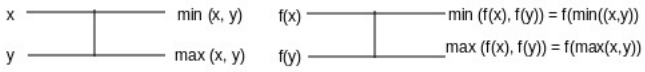


$f(x) = -6x$ would not be monotonic, whereas $f(x) = 6x$ would be a monotonic function.

Claim: Given a function $f(x) : x- > x$ is monotonic, it preserves miniumum and maximum operator.

$$f(min(x, y)) = min(f(x), f(y))$$

$$f(max(x, y)) = max(f(x), f(y))$$



Instead of give x,y as the inputs, give $f(x)$, $f(y)$.

Suppose for all $x_1..x_n$ such that the network does not sort it correctly, the result would be that for all x_i,$x_j : x_i < x_j$, the output puts x_j before x_i or output is faulty.

8.3.4 Bitonic Merge Lemma

Bitonic merge lemma states that given a bitonic sequence A of length $2n$. Let

$B = min(A[i], A[i + n])|0 \leq i < n$
$C = max(A[i], A[i + n])|0 \leq i < n$

Then B and C are both bitonic and all values in B are less than or equal to all values in C.

Let $A = 2 \quad 9 \quad 11 \quad 16 \quad 14 \quad 12 \quad 2 \quad 1$

$B = min(2, 14), min(9, 12), min(11, 2), min(16, 1)$

$C = max(2, 14), max(9, 12), max(11, 2), max(16, 1)$

$B = [2, 9, 2, 1]$, which is bitonic
$C = [14, 12, 11, 16]$, which is bitonic

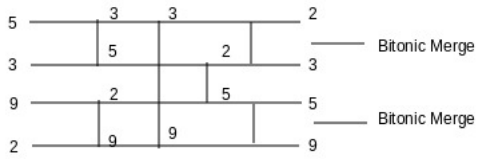
Case analysis of using 0-1 sequence can be used to verify the correctness of bitonic merge lemma.

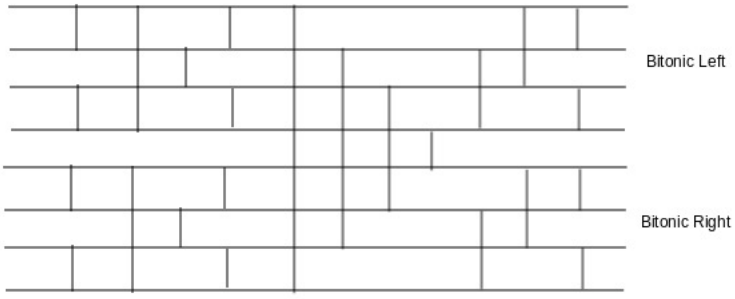
8.3.5 Bitonic Sort Algorithm

Bitonic sort recursively divide the array into two halves, and in each bitonic sort function call bitonicMerge to recursively sort the first half in ascending order, and sort the second half in descending order. Each bitonicMerge call would result in a bitonic sequence, which allows bitonic merge at a higher call level to recursively breaks the bitonic sequence into two halves and compares the elements and swap when necessary. The algorithm implementations are shown below:

```
1  void bitonicSort (int lo, int n, boolean dir) <- n is power of 2
2   if (n>1) {
3    int m = n/2
4    bitonicSort (lo, m, ASC)
5    ||
6    bitonicSort (lo+m, m, DESC)
7    bitonicMerge (lo, n, dir)
8   }
9  }
10
11 void bitonicMerge (int lo, int n, boolean dir){
12  if (n>1) {
13   int m = n/2;
14   for (int i = lo; i < lo + m; i++) in parallel do
15    compare (i, i + m, dir);//minimum on one side, maximum on the other side, double
          recursion
16    bitonicMerge(lo, m, dir)
17    ||
18    bitonicMerge(lo+m, m, dir)
19  }
20 }
21
22 //Assume i < j, guaranteed by the caller
23 void compare (int i, int j, boolean dir) {
24  if (dir == (a[i] > a[j] {
25   exchange (i,j);
26  }
27 }
```

The followings are graphical representation of bitonic sort algorithms for four inputs and eight inputs respectively.





8.3.6 Bitonic Sort Complexity Analysis

Bitonic sort: $T(n) = T(n/2) + O(logn)$ (bitonic merge)
$T(1) = O(1)$;
$T(n) = logn + (logn - 1) + (logn - 2) + ...$ (at each level)
$T(n) = O(logn)^2$

$W(n) = O(n \times log^2 n)$

There exists AKS network that runs $O(logn)$ and has work $O(n \times logn)$,but it is not fast enough to bitonic sort.