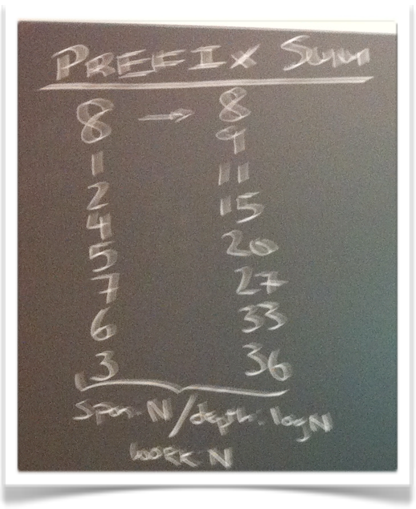
Parallel Sorting
February 22, 2011

I. Prefix Sum:



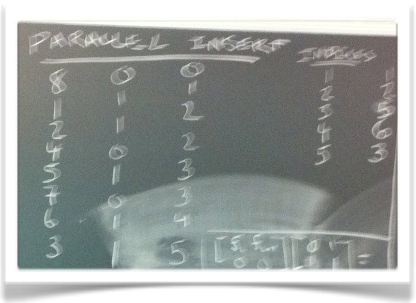
Prefix Sums allow us to convert a sequence:

$a_1, a_2, a_3, \ldots, a_N$

to the sequence of partial sums:

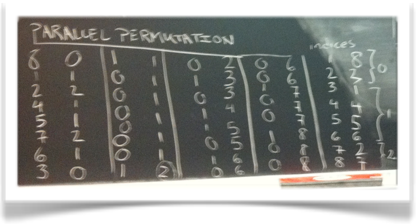
$a_1, a_1 + a_2, a_1 + a_2 + a_3, \ldots, a_1 + \ldots + a_N$

II. Parallel Insert:



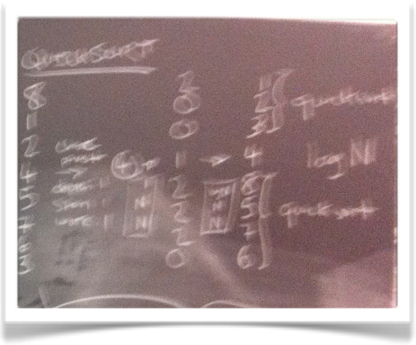
By performing prefix sums on a 0,1 mask, we can insert elements from a sequence into a shorter list in parallel.

III. Parallel Permutation:



By using labels from an index set (0, 1, ... k) we can use a similar idea to permute a set in parallel so that the elements labeled 0 appear before those labeled 1, which appear before those labeled 2, and so forth.

IV. QuickSort:



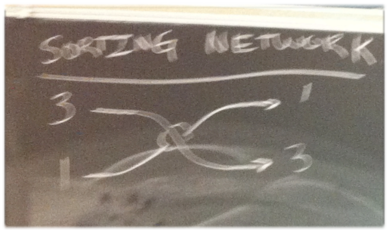
If we choose a pivot and label every element as follows:

0 - less than the pivot
1 - equal to the pivot
2 - greater than the pivot

then we can perform QuickSort in parallel.

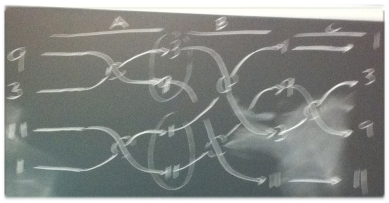
V. 2-input Sorting Network:

This sorting network takes any two numbers and returns them in sorted order.



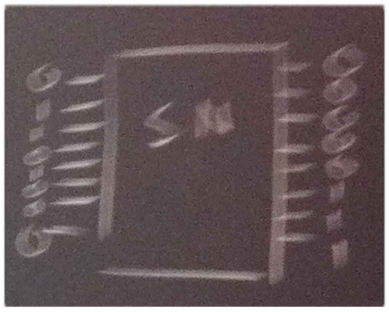
VI. 4-input Sorting Network:

This network similarly sorts 4 inputs.



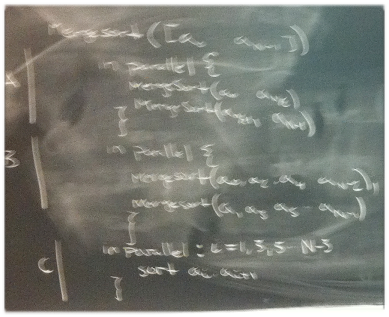
VII. Correctness:

The zero-one principle states that a sorting network is valid if it can sort all 2^n sequences of 0s and 1s.



VIII. MergeSort:

We can generalize the 4-input sorting network (VI) to create a parallel form of MergeSort that works for arbitrary sequences with power-of-two length.



```
MergeSort([a0 ... aN-1])
A   in parallel {
        MergeSort(a0 ... aN/2)
        MergeSort(aN/2+1 ... aN-1)
    }
B   in parallel {
        MergeSort(a0, a2, a4 ... aN-2),
        MergeSort(a1, a3, a5 ... aN-1)
    }
C   in parallel : i = 1, 3, 5 ... N-3
        sort ai, ai+1
    }
```

VIII. MergeSort Correctness:

An inductive proof of the correctness of MergeSort based on the zero-one principle: