Sorting Networks

Indranil Banerjee

George Mason University

ibanerje@gmu.edu

March 3, 2016
There are mainly two approaches to sorting in parallel:

1. **Non-oblivious**: Comparisons are data dependent
   
   Example: Parallel Quicksort, Parallel Merge Sort etc.

2. **Oblivious**: Comparisons are precomputed and does not depend on the results of previous comparisons.
   
   Example: Sorting Networks
Since the comparisons are not data dependent, we can precompute the comparisons and directly implement them inside a hardware.

An oblivious sorting algorithm proceeds in stages.

Each stage consists of a number of comparisons which occur concurrently.

We will look at one such algorithm: Batcher’s Odd-Even Merge Sort.
The Algorithm: OddEvenMergeSort(X)

**Input:** Array \( X = \{x_0, x_1, \ldots, x_{n-1}\} \) (Assume \( n \) is power of 2)

**Output:** Sorted sequence \( X \)

1. \( X_L = \{x_0, \ldots, x_{n/2-1}\} \) and \( X_R = \{x_{n/2}, \ldots, x_{n-1}\} \)
2. If \( n > 1 \):
   - OddEvenMergeSort(\( X_L \))
   - OddEvenMergeSort(\( X_R \))
   - OddEvenMerge(\( X_L, X_R \)) ← Recursive
Odd-Even Merge

The Algorithm: $\text{OddEvenMerge}(X)$

**Input:** An array $X$ whose two halves $X_L$ and $X_R$ are sorted (Assume $n = |X_L| = |X_R|$ is power of 2)

**Output:** Sorted sequence $X$

1. **If** $n > 2$ **Then**:
   - Let $X_{Even} = \{x_0, x_2, ..., x_n\}$ and $X_{Odd} = \{x_1, x_3, ... x_{n-1}\}$
   - i. $\text{OddEvenMerge}(X_{Even})$
   - ii. $\text{OddEvenMerge}(X_{Odd})$
   - iii. Pardo: $\text{Compare}(x_{2i-1}, x_{2i})$
     
     **While** $(1 \leq i \leq (n - 2)/2)$

2. $\text{Compare}(x_0, x_1)$
A Comparator:

Source: http://parallelcomp.uw.hu/ch09lev1sec2.html
Series Parallel Comparisons:

![Sorting Network Diagram](http://www.cs.cmu.edu/~tcortina/15110m14/ps9/)

**Figure:** What is this sorting algorithm?

**Source:**
http://www.cs.cmu.edu/~tcortina/15110m14/ps9/
Batcher’s Odd-Even Merge Sort Network:

Figure: The comparator blocks are individual merging networks
Batcher’s Odd-Even Merging Network:

\[ M_2 \]

\[ M_4 \]

**Figure:** Merging networks for \( n = 2, 4 \)
Batcher’s Odd-Even Merge Sort Network (Expanded):
Batcher’s Odd-Even Merge Sort Network (Expanded):
Batcher’s Odd-Even Merge Sort Network (Expanded):

```
1 3 6 7 2 5 8 9
```
Batcher’s Odd-Even Merge Sort Network (Expanded):
Batcher’s Odd-Even Merge Sort Network (Expanded):

![Batcher's Odd-Even Merge Sort Network Diagram](image-url)
Batcher’s Odd-Even Merge Sort Network (Expanded):

```
1
3
8
9
2
6
5
7
```

Sorting Networks
Indranil Banerjee
GMU March 3, 2016 15 / 19
Batcher’s Odd-Even Merge Sort Network (Expanded):
The correctness of the any oblivious sorting algorithm can be proven using the 0-1-principle.

**0-1-principle:** If a sorting network sorts every sequence of 0’s and 1’s, then it sorts every arbitrary sequence of values.

Complexity?
1. The correctness of the any oblivious sorting algorithm can be proven using the 0-1-principle.

2. **0-1-principle**: If a sorting network sorts every sequence of 0’s and 1’s, then it sorts every arbitrary sequence of values.

**Complexity?** Can be answered directly by looking at the network.

1. **Size**: $O(n \log^2 n)$ (This is the serial runtime)

2. **Depth**: $O(\log^2 n)$ (This is the parallel runtime)
1. Can we have sorting networks with $O(\log n)$ depth and $O(n \log n)$ size?

2. How do we implement such networks?