Sorting Networks

Indranil Banerjee

George Mason University

ibanerje@gmu.edu

March 3, 2016

Hardware Level Parallelism

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

Oblivious Sorting

1. Since the comparisons are not data dependent, we can precoumpute the comparisons and directly implemented them inside a hardware
2. An oblivious sorting algorithm proceeds in stages
3. Each stage consists of a number of comparisons which occur concurrently
4. We will look at one such algorithm: **Batcher's Odd-Even Merge Sort**

Odd-Even Merge Sort

The Algorithm: $\textsc{OddEvenMergeSort}(X)$

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 $n > 1$:
   $\textsc{OddEvenMergeSort}(X_L)$
   $\textsc{OddEvenMergeSort}(X_R)$
   $\textsc{OddEvenMerge}(X_L, X_R) \leftarrow$ Recursive

Odd-Even Merge

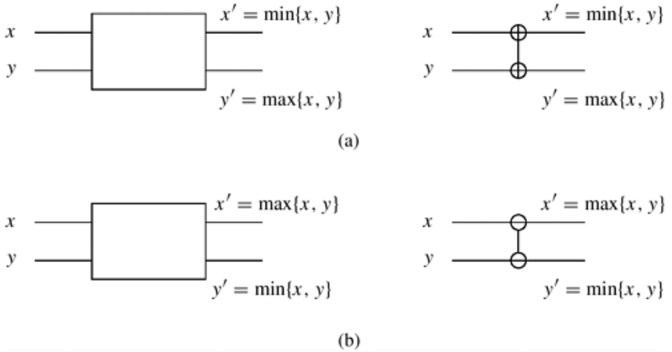
The Algorithm: OddEvenMerge(X)

Input: An arry X whose two halvs 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 OddEvenMerge(X_{Even})
   - ii OddEvenMerge(X_{Odd})
   - iii Pardo: Compare(x_{2i-1}, x_{2i})
     While $(1 \leq i \leq (n-2)/2)$
2. Compare(x_0, x_1)

Sorting Network

A Comparator:



Source: `http://parallelcomp.uw.hu/ch09lev1sec2.html`

Sorting Network

Series Parallel Comparisons:

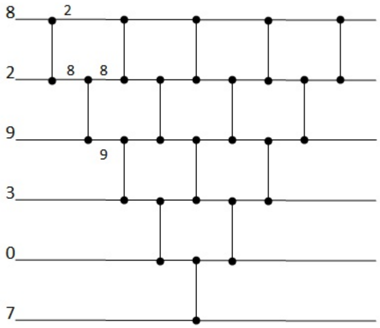


Figure: What is this sorting algorithm?

Source:

http://www.cs.cmu.edu/~tcortina/15110m14/ps9/

Sorting Network

Batcher's Odd-Even Merge Sort Network:

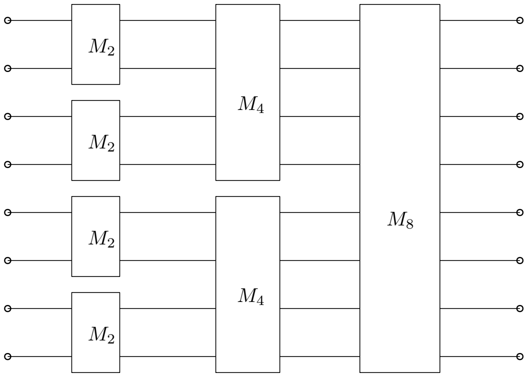


Figure: The comparator blocks are individual merging networks

Sorting Network

Batcher's Odd-Even Merging Network:

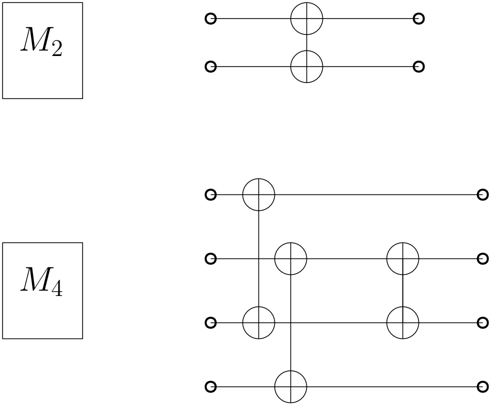
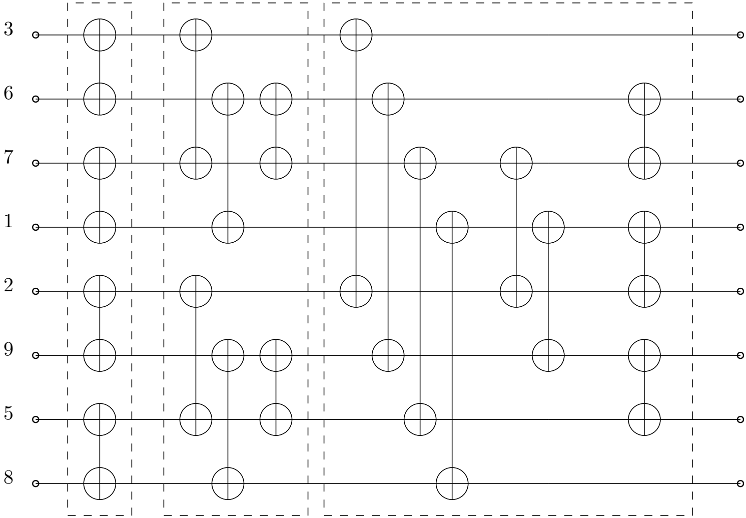


Figure: Merging networks for $n = 2, 4$

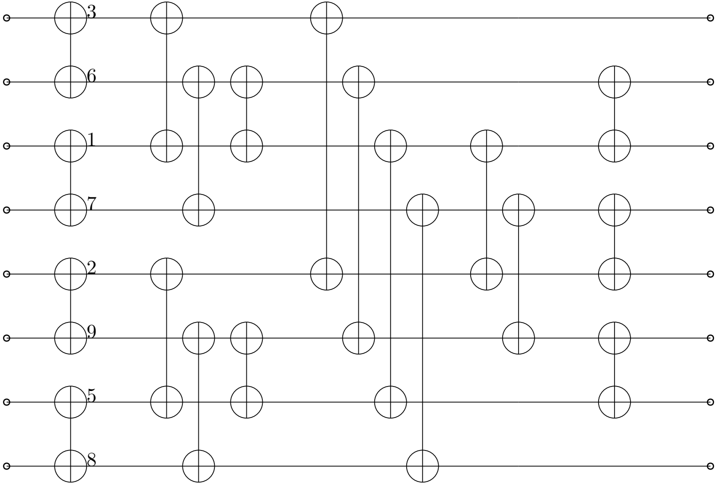
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



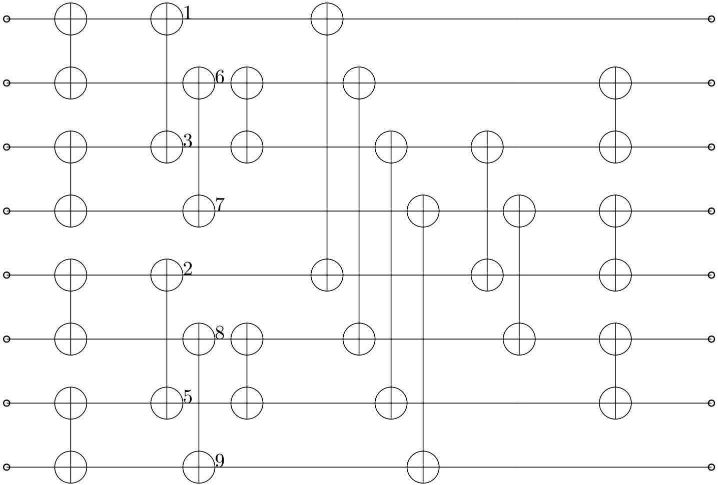
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



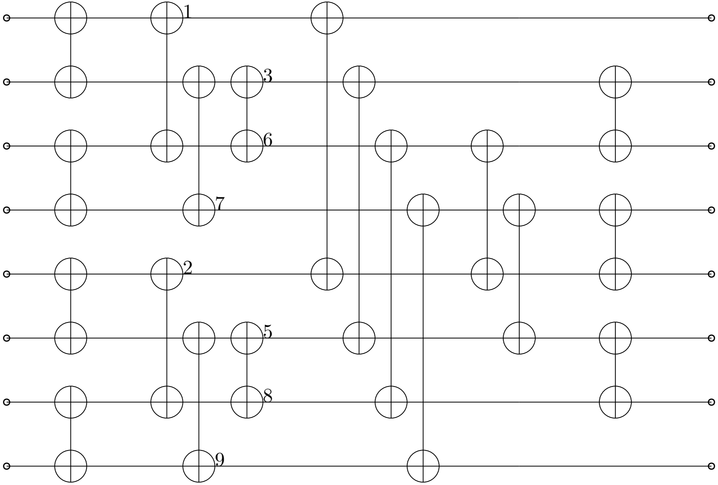
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



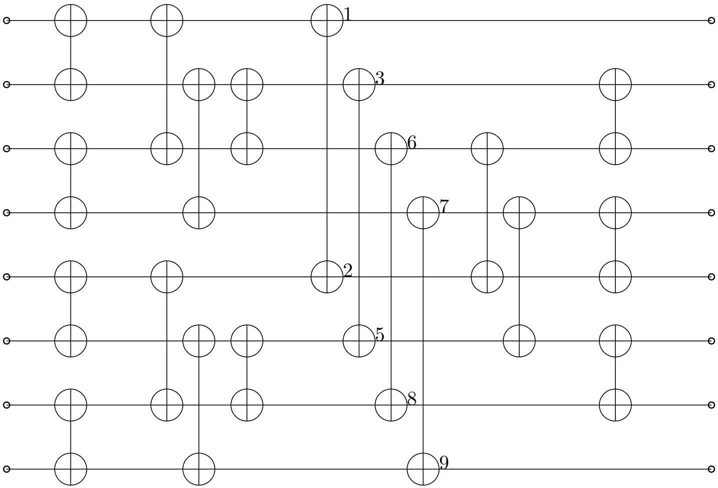
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



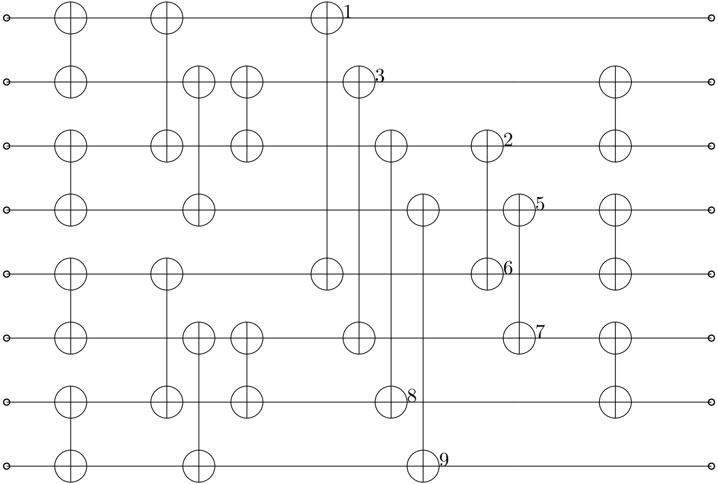
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



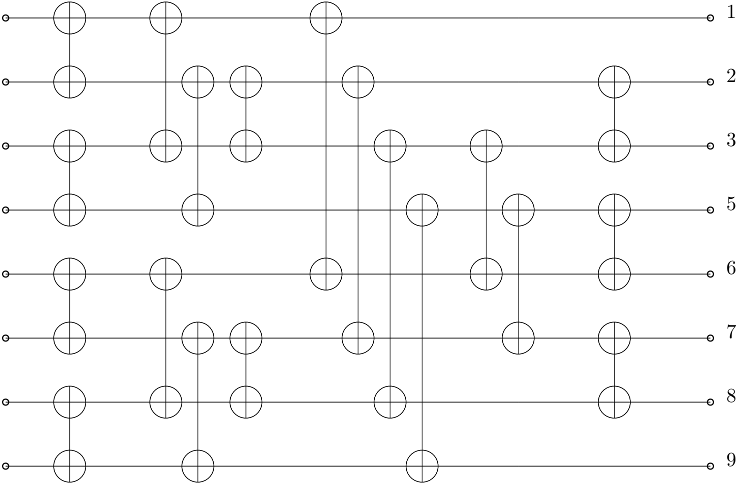
Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



Sorting Network

Batcher's Odd-Even Merge Sort Network (Expanded):



Proving Correctness: The 0-1 Principle

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?

Proving Correctness: The 0-1 Principle

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)

Q & A

1. Can we have sorting networks with $O(\log n)$ depth and $O(n \log n)$ size?
2. How do we implement such networks?