Parallel Recursion: Batcher’s Bitonic Sort

Greg Plaxton
Theory in Programming Practice, Spring 2005
Department of Computer Science
University of Texas at Austin
Overview

- Compare-interchange sorting algorithms
  - Adaptive versus oblivious
  - Zero-one principle
  - Comparator networks

- Batcher’s bitonic sort
  - High-level structure
  - Bitonic merge
  - Analysis
Compare-Interchange Operation

• Given an array of \( n \) items drawn from a totally ordered set (e.g., the integers) a *compare-interchange operation* is specified by an ordered pair \((i, j)\) of distinct array indices
  
  – The effect of this operation is to compare the two items in array locations \( i \) and \( j \) and interchange if necessary so that, after the operation, the item in location \( i \) is at most the item in location \( j \)
**Compare-Interchange Algorithm**

- Given an array of \( n \) items drawn from a totally ordered set (e.g., the integers) a *compare-interchange algorithm* performs a sequence of compare-interchange operations on the array.
  - No other kinds of operations are performed on the array.

- A compare-interchange algorithm is *oblivious* if, for any given \( n \), it specifies a fixed sequence of compare-interchange operations.

- A compare-interchange algorithm that is not oblivious is *adaptive*.
  - An adaptive algorithm might take into account the outcomes of previous compare-interchange operations (i.e., whether or not an interchange took place) to decide which compare-interchange operation to perform next.
Compare-Interchange Sorting Algorithm

- A compare-interchange algorithm is a sorting algorithm if it permutes the items of any given input array into ascending order.

- Example: For $n = 3$, the sequence of compare-interchange operations $(1, 2), (1, 3), (2, 3)$ corresponds to an oblivious compare-interchange sorting algorithm.
Zero-One Principle

• Theorem: If an oblivious compare-interchange algorithm sorts all zero-one inputs (i.e., any array in which each array item is either 0 or 1), then it is a sorting algorithm

• It is sufficient to prove that the theorem holds for any fixed $n$, that is, if a compare-interchange algorithm sorts all $2^n$ zero-one inputs of length $n$, then it sorts any input of length $n$

• So let us fix $n$ in the proof of the zero-one principle that follows

• Remark: The zero-one principle also holds for adaptive compare-interchange algorithms if we assume that ties are broken in a consistent manner
  – For example, we could break a tie between two items with equal keys according to the array indices of their initial locations
  – In this course, our use of the zero-one principle is confined to the oblivious case, so we will focus on that case in what follows
Proof of the Zero-One Principle: Overview

- Definition of a $k$-partitioner
- Proof of a lemma related to $k$-partitioners
- Proof of the zero-one principle using the $k$-partitioner lemma
Definition of a $k$-Partitioner

- Let $k$ be an integer such that $0 \leq k \leq n$

- A compare-interchange algorithm is a $k$-partitioner if it permutes the items of any given array of length $n$ so that, when the algorithm terminates, for every item $x$ in the first $k$ array locations, and every item $y$ in the last $n - k$ locations, $x \leq y$
$k$-Partitioner Lemma

- If an oblivious compare-interchange algorithm sorts every input consisting of $k$ 0's and $n - k$ 1's, then it is a $k$-partitioner
Proof of the Zero-One Principle

• By the $k$-partitioner lemma, it is sufficient to prove the following: If an oblivious compare-interchange algorithm is a $k$-partitioner for $0 \leq k \leq n$, then it is a sorting algorithm.
Comparator Networks

• An oblivious compare-interchange algorithm is also called a comparator network
  – In this context, a compare-interchange algorithm is called a comparator

• An oblivious compare-interchange sorting algorithm is also called a sorting network

• A useful pictorial representation

• Size and depth of a comparator network
A Lower Bound on the Size of any Sorting Network

- A sorting network has to be able to apply $n!$ different permutations to the input
- Therefore it needs to contain at least $\log_2(n!)$ comparators
- It is not hard to argue that $\log_2(n!) = \Theta(n \log n)$
A Lower Bound on the Depth of any Sorting Network

- Each level of a sorting network can contain at most $n/2$ comparators
- Since the size of a sorting network is $\Omega(n \log n)$, the depth is $\Omega(\log n)$
Batcher’s Bitonic Sort

- An elegant construction that achieves depth \( O(\log^2 n) \) and size \( O(n \log^2 n) \)

- Much more complicated constructions have been given that achieve depth \( O(\log n) \) and size \( O(n \log n) \)
  - As we have seen, these bounds are optimal
Batcher’s Bitonic Sort: High Level

• We will assume that $n$ is a power of 2
• If $n = 1$, do nothing
• Otherwise, proceed as follows:
  – Partition the input into two subarrays of size $n/2$
  – Recursively sort these two subarrays in parallel
  – Merge the two sorted subarrays
Bitonic Merge: Overview

- Definition of a bitonic zero-one sequence
- Recursive construction of a comparator network that sorts any bitonic sequence
- Observe that the preceding comparator network can be used for merging two sorted zero-one sequences
Bitonic Zero-One Sequence

- A zero-one sequence is said to be bitonic if it is either of the form $0^a1^b0^c$ or it is of the form $1^a0^b1^c$, where $a$, $b$, and $c$ are integers.
A Comparator Network that Sorts any Bitonic Zero-One Sequence

• Assume that the length of the sequence is a power of 2
• If the sequence is of length 1, do nothing
• Otherwise, proceed as follows:
  – Split the bitonic zero-one sequence of length \( n \) into the first half and the second half
  – Perform \( n/2 \) compare interchange operations in parallel of the form \((i, i + n/2)\), \( 0 \leq i < n/2 \) (i.e., between corresponding items of the two halves)
  – Claim: Either the first half is all 0’s and the second half is bitonic, or the first half is bitonic and the second half is all 1’s
  – Therefore, it is sufficient to apply the same construction recursively on the two halves
Analysis of Bitonic Merge

- Let $M(n)$ denote the depth of the bitonic merging network
- $M(1) = 0$ and $M(n) = M(n/2) + 1$ for $n > 1$
- Thus $M(n) = \log_2 n$
We will assume that $n$ is a power of 2

If $n = 1$, do nothing

Otherwise, proceed as follows:

- Partition the input into two subarrays of size $n/2$
- Recursively sort these two subarrays in parallel, one in ascending order and the other in descending order
- Observe that any 0-1 input leads to a bitonic sequence at this stage, so we can complete the sort with a bitonic merge
Analysis of Bitonic Sort

• Let $T(n)$ denote the depth of the bitonic sorting network

• $T(1) = 0$ and $T(n) = T(n/2) + \log_2 n$ for $n > 1$

• This recurrence implies $T(n) = O(\log^2 n)$

• It follows that the size of the bitonic sorting network is $O(n \log^2 n)$