Algorithms and Applications

Areas done in textbook:

- Sorting Algorithms
- Numerical Algorithms
- Image Processing
- Searching and Optimization

Chapter 10

Sorting Algorithms

- rearranging a list of numbers into increasing (strictly nondecreasing) order.

Potential Speedup

$O(n \log n)$ optimal for any sequential sorting algorithm without using special properties of the numbers.

Best we can expect based upon a sequential sorting algorithm using $n$ processors is

$$\text{Optimal parallel time complexity} = \frac{O(n \log n)}{n} = O(\log n)$$

Has been obtained but the constant hidden in the order notation extremely large.
Compare-and-Exchange Sorting Algorithms
Compare and Exchange

Form the basis of several, if not most, classical sequential sorting algorithms.

Two numbers, say A and B, are compared. If A > B, A and B are exchanged, i.e.:

\[
\begin{align*}
\text{if (A > B)} & \{
\text{temp = A;}
\text{A = B;}
\text{B = temp;}
\}\end{align*}
\]

Message-Passing Compare and Exchange

Version 1

\(P_1\) sends A to \(P_2\), which compares A and B and sends back B to \(P_1\) if A is larger than B (otherwise it sends back A to \(P_1\)):

Alternative Message Passing Method

Version 2

For \(P_1\) to send A to \(P_2\) and \(P_2\) to send B to \(P_1\). Then both processes perform compare operations. \(P_1\) keeps the larger of A and B and \(P_2\) keeps the smaller of A and B:

Note on Precision of Duplicated Computations

Previous code assumes that the if condition, A > B, will return the same Boolean answer in both processors.

Different processors operating at different precision could conceivably produce different answers if real numbers are being compared.

This situation applies to anywhere computations are duplicated in different processors to reduce message passing, or to make the code SPMD.
Data Partitioning
(Version 1)

$p$ processors and $n$ numbers. $n/p$ numbers assigned to each processor:

Bubble Sort

First, largest number moved to the end of list by a series of compares and exchanges, starting at the opposite end.

Actions repeated with subsequent numbers, stopping just before the previously positioned number.

In this way, the larger numbers move (“bubble”) toward one end.
Time Complexity

Number of compare and exchange operations

\[ \sum_{i=1}^{n-1} \frac{n(n-1)}{2} \]

Indicates a time complexity of O\(n^2\) given that a single compare-and-exchange operation has a constant complexity, O(1).

Odd-Even (Transposition) Sort

Variation of bubble sort.

Operates in two alternating phases, even phase and odd phase.

**Even phase**

Even-numbered processes exchange numbers with their right neighbor.

**Odd phase**

Odd-numbered processes exchange numbers with their right neighbor.

Odd-Even Transposition Sort

Sorting eight numbers

Parallel Bubble Sort

Iteration could start before previous iteration finished if does not overtake previous bubbling action:

Odd-Even Transposition Sort

Sorting eight numbers
Mergesort

A classical sequential sorting algorithm using divide-and-conquer approach. Unsorted list first divided into half. Each half is again divided into two. Continued until individual numbers are obtained.

Then pairs of numbers combined (merged) into sorted list of two numbers. Pairs of these lists of four numbers are merged into sorted lists of eight numbers. This is continued until the one fully sorted list is obtained.

Parallelizing Mergesort

Using tree allocation of processes

Analysis

Sequential

Sequential time complexity is $O(n \log n)$.

Parallel

2 log $n$ steps in the parallel version but each step may need to perform more than one basic operation, depending upon the number of numbers being processed - see text.

Quicksort

Very popular sequential sorting algorithm that performs well with average sequential time complexity of $O(n \log n)$.

First list divided into two sublists. All numbers in one sublist arranged to be smaller than all numbers in other sublist.

Achieved by first selecting one number, called a pivot, against which every other number is compared. If the number is less than the pivot, it is placed in one sublist. Otherwise, it is placed in the other sublist.

Pivot could be any number in the list, but often first number in list chosen. Pivot itself could be placed in one sublist, or the pivot could be separated and placed in its final position.
Parallelizing Quicksort

Using tree allocation of processes

With the pivot being withheld in processes:

Analysis

Fundamental problem with all tree constructions – initial division done by a single processor, which will seriously limit speed.

Tree in quicksort will not, in general, be perfectly balanced Pivot selection very important to make quicksort operate fast.

Work Pool Implementation of Quicksort

First, work pool holds initial unsorted list. Given to first processor which divides list into two parts. One part returned to work pool to be given to another processor, while the other part operated upon again.
Neither Mergesort nor Quicksort parallelize very well as the processor efficiency is low (see book for analysis).

Quicksort also can be very unbalanced. Can use load balancing techniques.

Batcher’s Parallel Sorting Algorithms

- Odd-even Mergesort
- Bitonic Mergesort

Originally derived in terms of switching networks.

Both are well balanced and have parallel time complexity of \( O(\log^2 n) \) with \( n \) processors.

Odd-Even Mergesort

Odd-Even Merge Algorithm

Start with odd-even merge algorithm which will merge two sorted lists into one sorted list. Given two sorted lists \( a_1, a_2, a_3, \ldots, a_n \) and \( b_1, b_2, b_3, \ldots, b_n \) (where \( n \) is a power of 2).
Odd-Even Mergesort
Apply odd-even merging recursively

Odd-Even Mergesort

Compare and exchange

Bitonic Mergesort
Bitonic Sequence

A monotonic increasing sequence is a sequence of increasing numbers.

A bitonic sequence has two sequences, one increasing and one decreasing. e.g.

\[ a_0 < a_1 < a_2, a_3, \ldots, a_{i-1} < a_i > a_{i+1}, \ldots, a_{n-2} > a_{n-1} \]

for some value of \( i \) (0 <= i < n).

A sequence is also bitonic if the preceding can be achieved by shifting the numbers cyclically (left or right).

Bitonic Sequences

“Special” Characteristic of Bitonic Sequences

If we perform a compare-and-exchange operation on \( a_i \) with \( a_{i+n/2} \) for all \( i \), where there are \( n \) numbers in the sequence, get TWO bitonic sequences, where the numbers in one sequence are all less than the numbers in the other sequence.
Creating two bitonic sequences from one bitonic sequence

Starting with the bitonic sequence

3, 5, 8, 9, 7, 4, 2, 1

we get:

\[ \text{Bitonic sequence} \]

3, 5, 8, 9, 7, 4, 2, 1

\[ \text{Compare and exchange} \]

3, 4, 2, 1, 7, 5, 8, 9

Bitonic sequence

Bitonic sequence

Sorting a bitonic sequence

Compare-and-exchange moves smaller numbers of each pair to left and larger numbers of pair to right. Given a bitonic sequence, recursively performing operations will sort the list.

Bitonic sequence

3, 5, 8, 9, 7, 4, 2, 1

Compare and exchange

3, 4, 2, 1, 7, 5, 8, 9

Sorted list

3, 4, 2, 1, 7, 5, 8, 9

3, 4, 2, 1, 7, 5, 8, 9

Sorting

To sort an unordered sequence, sequences are merged into larger bitonic sequences, starting with pairs of adjacent numbers.

By a compare-and-exchange operation, pairs of adjacent numbers formed into increasing sequences and decreasing sequences. Pairs form a bitonic sequence of twice size of each original sequences.

By repeating this process, bitonic sequences of larger and larger lengths obtained.

In the final step, a single bitonic sequence sorted into a single increasing sequence.

Bitonic Mergesort

Unsorted numbers

Bitonic sorting operation

Direction of increasing numbers

Sorted list
Phases

The six steps (for eight numbers) are divided into three phases:

Phase 1 (Step 1) Convert pairs of numbers into increasing/decreasing sequences and into 4-bit bitonic sequences.

Phase 2 (Steps 2/3) Split each 4-bit bitonic sequence into two 2-bit bitonic sequences, higher sequences at center. Sort each 4-bit bitonic sequence increasing/decreasing sequences and merge into 8-bit bitonic sequence.

Phase 3 (Steps 4/5/6) Sort 8-bit bitonic sequence.

Number of Steps

In general, with \( n = 2^k \), there are \( k \) phases, each of 1, 2, 3, ..., \( k \) steps. Hence the total number of steps is given by

\[
\text{Steps} = \sum_{i=1}^{k} i \cdot \frac{k(k+1)}{2} = \frac{\log n (\log n + 1)}{2} = O(\log^2 n)
\]

Sorting Conclusions so far

Computational time complexity using \( n \) processors

- Odd-even transposition sort - \( O(n) \)
- Parallel mergesort - \( O(n) \) but unbalanced processor load and Communication
- Parallel quicksort - \( O(n) \) but unbalanced processor load, and communication can generate to \( O(n^2) \)
- Odd-even Mergesort and Bitonic Mergesort \( O(\log^2 n) \)

Bitonic mergesort has been a popular choice for a parallel sorting.