

Intro to sorting

9/21/07

Serial sorting algorithms

- Compare-exchange sorts
 - $O(n \log n)$:: Quicksort, Merge sort
- Non-comparison sorts
 - $O(n)$:: Counting sort, Bucket sort

Parallelization

- If we were able to achieve 100% efficiency on n processors, i.e., $S(p) / p = 1$ would require:
 - Comparison-based sort: $T(n,p) = O(\log n)$
 - Non-comparison-based: $T(n,p) = O(1)$
- Optimally efficient comparison-based sorts have been developed, but they have a very large constant associated.
 - Not useful in practice

Assumptions

- Before sorting
 - Data is distributed on the nodes
 - Data consists of positive integers
 - Data is to be sorted by its numerical value
- After sort
 - Sorted sub-list on each node (distributed)
 - Sub-lists are globally ordered in some specific way

Counting sort

- Suppose we have an array of values, A , that are all between $0 \dots k$, where k is some constant
- How can we sort these in parallel?

Problem #4

- Let A be an array of n elements. We are given an array L of the same length such that $L[i]$ is the label of the element $A[i]$. Assume that each label is an integer in the range $1, 2, \dots, k$, where k is a constant. We want to assign a unique rank in the range $1, 2, \dots, n$ to each element of $A[i]$ according to the following rules:
 - (1) If $A[i]$ and $A[j]$ have different labels, the element with the smaller label should have the lower rank.
 - (2) If $A[i]$ and $A[j]$ have the same label, the element at the lower index should have the lower rank.

Overview

- Although we have two arrays, A and L , we are only concerned with L
- Idea:
 - Note all elements with a smaller label go before me. So my rank is at least this total
 - Further, my rank is offset by the number of elements with my label preceding me

Solution

- If all data are distributed:
 - Each processor declares an array “local_count” of size k , and then counts the number of local occurrences of each label $1 \leq i \leq k$.
 - Use parallel prefix sum to determine the number of occurrences of each label on all processors with a lower rank, and store this in array “Psum”.
 - Concurrently, find the total number of occurrences of each label on all processors and store this information in an array “TSum”
 - Let r be the label of element $A[j]$. If so Rank of $A[j] = (\text{Sum of all TSum}[j], \text{ where } j < r) + \text{Psum}[r] + \# \text{ of elements before } A[i] \text{ on the the same processor with label } r$.

Two methods

- Data division
 - Divide the data among the processors
 - Each processor counts the data it owns
 - Parallel prefix sum reduction across all processors to get the final count
- Bucket division
 - divide the buckets across the processors
 - broadcast data chunks
 - each processor updates its buckets
 - condense at the end to get the final collection

Parallel sorting approaches

- There are almost as many parallel sorts as there are serial sorting algorithms.
- We will focus on four in class:
 - Parallel bubble sort
 - Odd-even transposition sort
 - Bitonic sort
 - Sample sort

Compare exchange

- If ($A > B$)
 - $Temp = A$
 - $A = B$
 - $B = Temp$;

Bubble sort

```
for (i=0; i<n-1; i++) {  
  for (j=0; j<n-1-i; j++)  
    if (a[j+1] < a[j]) { /* compare the two neighbors */  
      tmp = a[j]; /* swap */  
      a[j] = a[j+1];  
      a[j+1] = tmp;  
    }  
}
```

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.

Example

4 ⇔ 3 2 ⇔ 1
3 4 ⇔ 1 2
3 ⇔ 1 4 ⇔ 2
1 3 ⇔ 2 4
1 ⇔ 2 3 ⇔ 4