Parallel Sorting

Murat Manguoğlu

Middle East Technical University

Computer Engineering

CENG478 – Spring 2019

Acknowledgement: Introduction to Parallel Computing, by Grama, Gupta, Kumar, and Karypis, Addison Wesley.
Sorting basics

- Comparison based sorting
  (mergesort, quicksort, shellsort ...)

- Noncomparison based sorting
  (bucket sort, counting sort, radix sort ...)

Seq. Lower bound:

$O(n \log n)$

$O(n)$
Parallel vs Sequential Sorting Differences

• Need to know where input and output are stored
• Need to know underlying architecture
• Communication
• Best sequential sorting algorithms are not always the best in parallel
Comparison based sorting

Fundamental operation is compare and exchange

Cost of communication: $t_s + mt_m$
Comparison based sorting

Fundamental operation is (more general) compare and split

Cost of communication: $t_s + (n/p)mt_m$
Sorting networks

*A hardware for sorting that is composed of comparators**

** Comparator:
Bitonic Merge networks

*A hardware that sorts a bitonic sequence

** A bitonic sequence is a sequence of numbers that are strictly increasing and decreasing (in a cyclic way)

i.e. \( x_0 \leq \cdots \leq x_k \geq \cdots \geq x_{n-1} \) for some \( k \) or a circular shift of such a sequence.

Bitonic sequence examples:

\{0,2,5,6,10,9,7,1\}

\{1,0,2,5,6,10,9,7\}

\{7,1,0,2,5,6,10,9\}
Bitonic Merge network

Log₂p stages - Each stage has n/2 comparators
Bitonic Sort network

Any input seq. $\rightarrow$ Bitonic seq. $\rightarrow$ Sorted seq.

n-numbers to sort

Total number of stages:

$$(1 + 2 + 3 + \ldots + \log_2 n) \text{ stages}$$

$$= \log_2 n (\log_2 n + 1) / 2 \text{ stages}$$

i.e. $O((\log_2 n)^2)$ Time

$$(n/2) \cdot \log_2 n (\log_2 n + 1) / 2 \text{ comparators}$$

i.e. $O(n(\log_2 n)^2)$ comparators
Bitonic Sort Algorithm on Hypercube

Each stage corresponds to communication along a dimension of an hypercube network

Case 1: one element per node/processor/processing element

\[ T_{\text{comm}} = \log^2 n * (t_s + m * t_m) \]

\[ T_{\text{comp}} = \log^2 n * t_c \]
Bitonic Sort Algorithm on Hypercube

Each stage corresponds to communication along a dimension of an hypercube network

Case 2) more than one \((n/p)\) element per node/processor/processing element

\[
T_{\text{comm.}} = \log^2 p \left( t_s + \frac{n}{p}m t_m \right)
\]

\[
T_{\text{comp.}} = \left[ \left( \frac{n}{p} \log \left( \frac{n}{p} \right) + \frac{n}{p} \log^2 p \right) + \frac{n}{p} \log^2 p \right] t_c
\]

- Local sort
- Comparisons per each merge
Sequential Bubble sort

procedure BUBBLE_SORT(n)
begin
    for i := n - 1 downto 1 do
        for j := 1 to i do
            compare-exchange(a_j, a_{j+1});
    end BUBBLE_SORT

Cost: \( O(n^2) \), algorithm is inherently sequential but there is a variant that can be parallelized.
Sequential Odd-Even Transposition

procedure ODD-EVEN(n)
begin
  for i := 1 to n do
    begin
      if i is odd then
        for j := 0 to n/2 − 1 do
          compare-exchange(a_{2j+1}, a_{2j+2});
      if i is even then
        for j := 1 to n/2 − 1 do
          compare-exchange(a_{2j}, a_{2j+1});
    end for
  end ODD-EVEN
Odd-Even Transposition

Sorting \( n = 8 \) elements, using the odd-even transposition sort algorithm. During each phase, \( n = 8 \) elements are compared.

Unsorted

\[
\begin{array}{cccccccc}
3 & 2 & 3 & 8 & 5 & 6 & 4 & 1 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 1 (odd)

\[
\begin{array}{cccccccc}
2 & 3 & 3 & 8 & 5 & 6 & 1 & 4 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 2 (even)

\[
\begin{array}{cccccccc}
2 & 3 & 3 & 5 & 8 & 1 & 6 & 4 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 3 (odd)

\[
\begin{array}{cccccccc}
2 & 3 & 3 & 5 & 1 & 8 & 4 & 6 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 4 (even)

\[
\begin{array}{cccccccc}
2 & 3 & 3 & 1 & 5 & 4 & 8 & 6 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 5 (odd)

\[
\begin{array}{cccccccc}
2 & 3 & 1 & 3 & 4 & 5 & 6 & 8 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 6 (even)

\[
\begin{array}{cccccccc}
2 & 1 & 3 & 3 & 4 & 5 & 6 & 8 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 7 (odd)

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 3 & 4 & 5 & 6 & 8 \\
\| & | & | & | & | & | & |
\end{array}
\]

Phase 8 (even)

Sorted

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 3 & 4 & 5 & 6 & 8 \\
\| & | & | & | & | & | & |
\end{array}
\]
Parallel Odd-Even Transposition

Case 1) one element per node/processor/processing element

\[ T_{\text{comm.}} = n^*(t_s + m*t_m) \]

\[ T_{\text{comp.}} = n^*t_c \]
Parallel Odd-Even Transposition

Case 2) \((n/p)\) elements per node/processor/processing element

\[ T_{\text{comm.}} = p \cdot (t_s + (n/p) \cdot m \cdot t_m) \]

\[ T_{\text{comp.}} = [(n/p) \log(n/p) + (n/p) \cdot p] \cdot t_c \]

Local sort \hspace{2cm} \text{Comparisons}
An example of the first phase of parallel shellsort on an eight-process array, second phase is odd-even transpositions as long as needed (assume L times)
Shell sort

(n/p) elements per node/processor/processing element

\[ T_{\text{comm.}} = (\log_2 p + L)(t_s + (n/p)m t_m) \]

\[ T_{\text{comp.}} = [(n/p)\log(n/p) + (n/p)(\log_2 p + L)]^1 t_c \]
Bucket and Sample sort

• In Bucket sort, the range \([a,b]\) of input numbers is divided into \(m\) equal sized intervals, called buckets.

• Each element is placed in its appropriate bucket.

• If the numbers are uniformly divided in the range, the buckets can be expected to have roughly identical number of elements.

• Elements in the buckets are locally sorted.

• The run time of this algorithm is \(\Theta(n\log(n/m))\)
Parallel Bucket Sort

- Parallelizing bucket sort is relatively simple. We can select $m = p$
- In this case, each processor has a range of values it is responsible for
- Each processor runs through its local list and assigns each of its elements to the appropriate processor
- The elements are sent to the destination processors using a single all-to-all personalized communication
- Each processor sorts all the elements it receives
Parallel Bucket and Sample sort

• The critical aspect of the above algorithm is one of assigning ranges to processors. This is done by suitable splitter selection.

• The splitter selection method divides the $n$ elements into $m$ blocks of size $n/m$ each, and sorts each block by using quicksort.

• From each sorted block it chooses $m - 1$ evenly spaced elements.

• The $m(m - 1)$ elements selected from all the blocks represent the sample used to determine the buckets.

• This scheme guarantees that the number of elements ending up in each bucket is less than $2n/m$. 
Parallel Bucket and Sample sort

An example of the execution of sample sort on an array with 24 elements on three processes.
Parallel Bucket and Sample sort

- The splitter selection scheme can itself be parallelized
- Each processor generates the $p - 1$ local splitters in parallel
- All processors share their splitters using a single all-to-all broadcast operation
- Each processor sorts the $p(p - 1)$ elements it receives and selects $p - 1$ uniformly spaced splitters from them
Parallel Bucket and Sample sort

- The internal sort of $n/p$ elements requires time $\Theta((n/p)\log(n/p))$, and the selection of $p - 1$ sample elements requires time $\Theta(p)$.

- The time for an all-to-all broadcast is $\Theta(p^2)$, the time to internally sort the $p(p - 1)$ sample elements is $\Theta(p^2\log p)$, and selecting $p - 1$ evenly spaced splitters takes time $\Theta(p)$.

- Each process can insert these $p - 1$ splitters in its local sorted block of size $n/p$ by performing $p - 1$ binary searches in time $\Theta(p\log(n/p))$.

- The time for reorganization of the elements is $O(n/p)$.
Parallel Bucket and Sample sort

The total time is given by:

\[ T_P = \Theta \left( \frac{n}{p} \log \frac{n}{p} \right) + \Theta \left( \frac{p^2 \log p}{n} \right) + \Theta \left( \frac{p \log \frac{n}{p}}{n} \right) + \Theta \left( \frac{n}{p} \right). \]