Parallel Algorithms - sorting

Fernando Silva

DCC-FCUP

(Some slides are based on those from the book “Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers, 2nd ed.” de B. Wilkinson)
Sorting in Parallel

Why?
- it is a frequent operation in many applications

Goal?
- sorting a sequence of values in increasing order using $n$ processors

Potential speedup?
- best sequential algorithm has complexity $O(n \log n)$
- the best we can aim with a parallel algorithm, using $n$ processors is:
  - optimal complexity of a parallel sorting algorithm: $O(n \log n)/n = O(\log n)$
Compare-and-swap with message exchange (1/2)

Sequential sorting requires the comparison of values and swapping in the positions they occupy in the sequence. And, if it is in parallel? And, if the memory is distributed?

**version 1:**

- $P_1$ send $A$ to $P_2$, this compares $B$ with $A$ and sends to $P_1$ the $min(A, B)$.
version 2:

- $P_1$ sends $A$ to $P_2$; $P_2$ sends $B$ to $P_1$; $P_1$ does $A = \min(A, B)$ and $P_2$ does $B = \max(A, B)$. 

![Diagram of the compare-and-swap algorithm with message exchange version 2]
Data partition

Version 1:

- $n$ numbers and $p$ processors
- $n/p$ numbers assigned to each processor.
Merging two sub-lists

version 2:
Bubble Sort

- compares two consecutive values at a time and swaps them if they are out of order.
- greater values are being moved towards the end of the list.
- number of comparisons and swaps: \[ \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \]

which corresponds to a time complexity \( O(n^2) \).
Bubble-sort example

Original sequence: 4 2 7 8 5 1 3 6

Phase 1
Place largest number

Phase 2
Place next largest number

Time
Parallel Bubble-sort

The idea is to run multiple iterations in parallel.
Odd-Even with transposition (1/2)

- It is a variant of the bubble-sort
- Operates in two alternate phases:

  **Phase-even:**
  - Even processes exchange values with right neighbors.

  **Phase-odd:**
  - Odd processes exchange values with right neighbors.
## Odd-Even with transposition (2/2)

<table>
<thead>
<tr>
<th>Step</th>
<th>P₀</th>
<th>P₁</th>
<th>P₂</th>
<th>P₃</th>
<th>P₄</th>
<th>P₅</th>
<th>P₆</th>
<th>P₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Parallel algorithm - Odd-Even with transposition

```c
void ODD-EVEN-PAR(n) {
    id = process label
    for (i = 1; i <= n; i++) {
        if (i is odd)
            compare-and-exchange-min(id+1);
        else
            compare-and-exchange-max(id-1);
        if (i is even)
            compare-and-exchange-min(id+1);
        else
            compare-and-exchange-max(id-1);
    }
}
```
Mergesort (1/2)

- Example of a *divide-and-conquer* algorithm
- Sorting method to sort a vector; first subdivides it in two parts, applies again the same method to each part and when they are both sorted (2 sorted vectors/lists) with $m$ and $n$ elements, they are merged to produce a sorted vector that contains $m + n$ elements of the initial vector.
- The average complexity is $O(n \log n)$. 

```
6  24 28  3 13 10  7 30 22 16  8 25 12  5

separa em 2 listas e ordenar cada uma

3  6  7 10 13 24 28
5  8 12 16 22 25 30

junta as listas ja ordenadas.

3  5  6  7  8 10 12 13 16 22 24 25 28 30
```
Parallel Mergesort (2/2)

Using a strategy to assign work to processors organized in a tree.
Parallel Quicksort

Using a strategy for work-assignment in a tree-fashion.
Difficulties with the allocation of processes organized in a tree

- the initial division starts with just one process, which is limiting.
- the search tree of quicksort is not, in general, balanced
- selecting the pivot is very important for efficiency
Odd-Even mergesort

- complexity: $O(\log^2 n)$
- merging the two lists $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, where $n$ is a power of 2.
Odd-Even mergesort

Apply recursively odd-even merge:
Bitonic Sort (1/7)

- complexity: $O(\log^2 n)$
- a sequence is bitonic if it contains two sequences, one increasing and one decreasing, i.e.
  \[ a_1 < a_2 < \ldots < a_{i-1} < a_i > a_{i+1} > a_{i+2} > \ldots > a_n \]
  for some $i$ such that $(0 \leq i \leq n)$
- a sequence is bitonic if the property described is attained by a circular rotation to the right of its elements.
- Examples:
Bitonic Sort (2/7)

Special characteristic of bitonic sequences:

- if we do a compare-and-exchange operation with elements $a_i$ and $a_{i+n/2}$, for all $i$, in a sequence of size $n$,
- we obtain two bitonic sequences in which all the values in one sequence are smaller than the values of the other.
- Example: start with sequence 3, 5, 8, 9, 7, 4, 2, 1 and we obtain:
Bitonic Sort (3/7)

- the compare-and-exchange operation moves smaller values to the left and greater values to the right.
- given a bitonic sequence, if we apply recursively these operations we get a sorted sequence.
Bitonic Sort example (4/7)
Bitonic Sort (5/7)

To sort an unsorted sequence

- merge sequences in larger bitonic sequences, starting with adjacent pairs, alternating monotonicity.
- in the end, the bitonic sequence becomes sorted in a unique increasing sequence.
Bitonic Sort (6/7)

<table>
<thead>
<tr>
<th>Step No.</th>
<th>Processor No.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>L</td>
</tr>
<tr>
<td>6</td>
<td>L</td>
</tr>
</tbody>
</table>
Unsorted sequence $\Rightarrow$ bitonic sequence $\Rightarrow$ sorted sequence.