Parallel Quicksort

Ned Nedialkov

McMaster University
Canada

CS/SE 4F03
March 2016

Outline

Quick sort

Algorithm

$Quicksort(A, p, r)$
if $p \geq r$
    $x = A[r], \; i = p - 1$
    for $j = p$ to $r - 1$
        if $A[j] \leq x$
            $i = i + 1$
            $swap(A[i], A[j])$
    $swap(A[i + 1], A[r])$
    $Quicksort(A, p, i)$
    $Quicksort(A, i + 2, r)$

After first partitioning

10 2_i	2_j 10_j	1 1	3 3	7 7	6 6	4 4	5 5
2_i 2	10 1_i	1_j 10_j	3 3	7 7	6 6	4 4	5 5
2 2	1_i 1	10 3_i	3_j 10_j	7 7	6 6	4 4	5 5
2 2	1 1	3_i 3	10 4_i	7 7	6 6	4_j 10_j	5 5
2	1	3	4	5	6	10	7

- Initial array A

10	2	1	3	7	6	4	5

- Quicksort(A,1,8)
  - pivot 5, after partition, $i = 4$

2	1	3	4	5	6	10	7

  - Quicksort(A,1,4)
  - Quicksort(A,6,8)
- Quicksort(A,1,4)
  - pivot 4, after partition, $i = 3$

2	1	3	4	5	6	10	7

  - Quicksort(A,1,3)
  - Quicksort(A, 5, 4) : return

- Qicksort(A,1,3)
  - pivot 3, after partition, $i = 2$

2	1	3	4	5	6	10	7

  - Quicksort(A,1,2)
  - Quicksort(A,4,3) : return
- Quicksort(A,1,2)
  - pivot 1, after partition, $i = 0$

1	2	3	4	5	6	10	7

  - Quicksort(A,1,0) : return
  - Quicksort(A,2,2) : return
- Quicksort(A,6,8)
  - pivot 7, after partition, $i = 6$

1	2	3	4	5	6	7	10

  - Quicksort(A,6,6) : return
  - Quicksort(A,8,8) : return

Performance

- Worst case $O(n^2)$
- Average case $O(n \log n)$
- Crucial for the performance is how to select pivots
- One can select at random

Parallel formulation

- Assume n numbers and p processes
- Each process gets n/p consecutive elements
- Select a pivot
- Broadcast it to all processes
- Process i computes block of elements S_i and L_i such that

$$S_i \leq \text{pivot} < L_i$$

- Rearrange the elements of the original array such that it is partitioned as

$$S = \cup_i S_i \leq \text{pivot} < L = \cup_i L_i$$

- Assign p_1 processes to work on S and p_2 processes to work on L, $p_1 + p_2 = p$

- Apply the same scheme on S with p_1 processes and L with p_2 processes
- $p_1 = \lfloor |S| p/n + 0.5 \rfloor$, $p_2 = p - p_1$
- Partition until a block is assigned to a single process and then sort serially

Example

<table>
<tr><th colspan="5">P_0</th><th colspan="5">P_1</th><th colspan="5">P_2</th><th colspan="5">P_3</th></tr>
<tr><td>86</td><td>79</td><td>52</td><td>18</td><td>40</td><td>14</td><td>4</td><td>94</td><td>31</td><td>30</td><td>34</td><td>47</td><td>65</td><td>3</td><td>85</td><td>56</td><td>86</td><td>35</td><td>6</td><td>45</td></tr>
</table>

After first partition with pivot 45

<table>
<tr><th colspan="5">P_0</th><th colspan="5">P_1</th><th colspan="5">P_2</th><th colspan="5">P_3</th></tr>
<tr><td>18</td><td>40</td><td>52</td><td>86</td><td>79</td><td>14</td><td>4</td><td>31</td><td>30</td><td>94</td><td>34</td><td>3</td><td>85</td><td>47</td><td>65</td><td>35</td><td>6</td><td>45</td><td>86</td><td>56</td></tr>
</table>

After combining the lower and upper parts

<table>
<tr><th colspan="5">P_0</th><th colspan="5">P_1</th><th></th><th colspan="4">P_2</th><th colspan="5">P_3</th></tr>
<tr><td>18</td><td>40</td><td>14</td><td>4</td><td>31</td><td>30</td><td>34</td><td>3</td><td>35</td><td>6</td><td>45</td><td>52</td><td>86</td><td>79</td><td>94</td><td>85</td><td>47</td><td>65</td><td>86</td><td>56</td></tr>
</table>

After second partition P_0, P_1 with pivot 6, P_2, P_3 with pivot 56

<table>
<tr><th colspan="5">P_0</th><th colspan="5">P_1</th><th></th><th colspan="4">P_2</th><th colspan="5">P_3</th></tr>
<tr><td>4</td><td>31</td><td>14</td><td>18</td><td>40</td><td>3</td><td>6</td><td>30</td><td>35</td><td>34</td><td>45</td><td>52</td><td>94</td><td>79</td><td>86</td><td>47</td><td>56</td><td>65</td><td>86</td><td>85</td></tr>
</table>

After combining the lower and upper parts

<table>
<tr><th colspan="2">P_0</th><th></th><th colspan="7">P_1</th><th></th><th colspan="2">P_2</th><th></th><th colspan="6">P_3</th></tr>
<tr><td>4</td><td>3</td><td>6</td><td>31</td><td>14</td><td>18</td><td>40</td><td>30</td><td>35</td><td>34</td><td>45</td><td>52</td><td>47</td><td>56</td><td>94</td><td>79</td><td>86</td><td>65</td><td>86</td><td>85</td></tr>
</table>

P_0			P_1								P_2			P_3					
4	3	6	31	14	18	40	30	35	34	45	52	47	56	94	79	86	65	86	85

Now each process sorts sequentially

P_0			P_1								P_2			P_3					
3	4	6	14	18	30	31	34	35	40	45	47	53	56	65	79	85	86	86	94

Pivot selection

- Selecting a pivot at random works well in the sequential quick sort
- A process from a process group can select a pivot at random
- If a “bad” partition occurs, we may have load imbalance
- Assume uniform distribution of the elements
- If we assume uniform distribution of elements, n/p can be considered as a representative sample
- A process can pick the median of these n/p elements and round to the closest element
- The partitions are roughly in half

Combining blocks

- How to arrange the S_i and L_i?
- We can concatenate them in process order
- Need to find where each block starts
- S_0 is at the beginning of the array
- S_1 starts at location $|S_0|$
- S_2 starts at location $|S_0| + |S_1|$
- jth element of S_i is at location $\sum_{k=0}^{i-1} |S_k| + j$
- jth element of L_i is at location $\sum_{k=i}^{p-1} |L_k| + j$
- We can maintain arrays

$$Q_i = \sum_{k=0}^{i-1} S_k, \quad R_i = \sum_{k=0}^{i-1} L_k$$

- Q_i is the start of S_i
- R_i is the start of L_i, where numbering is from the last element of the last S_i

MPI version

The algorithm so far is suitable for a shared memory implementation. How to do a distributed implementation?

- Distribute the array to be sorted
- Broadcast a pivot among p processes
- Each partitions n/p elements in $O(n/p)$
- The re-arrangement of lower and upper parts involves communication
  - Need to know where a process should send its S_i and L_i parts
  - Once determined, data gets exchanged