Parallel Quicksort

Ned Nedialkov

McMaster University Canada

> CS/SE 4F03 March 2016

Outline

- Quick sort
- Performance
- Parallel formulation
- Example
- **Pivot selection**
- **Combining blocks**
- **MPI** version

Quick sort

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

After first partitioning

10	2 _j	1	3	7	6	4	5
2,	1Ój	1	3	7	6	4	5
2 _i 2	10	1 j	3	7	6	4	5
	1 <i>i</i>	10 _j	3	7	6	4	5
2 2	1 _i	10	3 _j	7	6	4	5
	1	3 <i>i</i>	10 _j	7	6	4	5
2 2	1	3 <i>i</i>	10	7	6	4 _j	5
2	1	3	4 <i>i</i>	7	6	10 _j	5
2	1	3	4	5	6	10	7

Initial array A

Performance

- 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(n²)
- 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

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

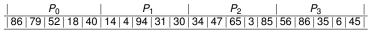
Assign p₁ processes to work on S and p₂ processes to work on L, p₁ + p₂ = p

Apply the same scheme on S with p₁ processes and L with p₂ 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



After first partition with pivot 45

 P0
 P1
 P2
 P3

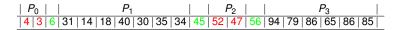
 18
 40
 52
 86
 79
 14
 4
 31
 30
 94
 34
 3
 85
 47
 65
 35
 6
 45
 86
 56

After combining the lower and upper parts

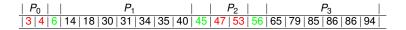
After second partition P_0 , P_1 with pivot 6, P_2 , P_3 with pivot 56

After combining the lower and upper parts

Quick sort	Performance	Parallel formulation	Example	Pivot selection	Combining blocks	MPI version
------------	-------------	----------------------	---------	-----------------	------------------	-------------



Now each process sorts sequentially



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
- S₀ is at the beginning of the array
- S_1 starts at location $|S_0|$
- S_2 starts at location $|S_0| + |S_1|$
- *j*th element of S_i is at location $\sum_{k=0}^{i-1} |S_k| + j$
- *j*th 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