PARALLEL QUICKSORT

Student: Chia Chen Chen
Advisor: Prof. Russ Miller
Sorting Algorithm

- A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements.
- The comparison operator is used to decide the new order of element in the respective data structure.
Sorting

- A brute way is to start from the array beginning and go through every elements every time.
- Store the first element in an empty array.
- Iterating the array, search for a smaller and hasn’t been stored yet element.
Then store the next element and continue the iteration until all the elements have been added
Then store the next element and continue the iteration until all the elements have been added.
Sorting

If the array has n elements

=> Running time: $O(n^2)$
Different Sorting Strategy

- Selection Sort
- Bubble Sort
- Recursive Bubble Sort
- Insertion Sort
- Etc.

- Recursive Insertion Sort
- Merge Sort
- Bitonic Sort
- Quick Sort
QuickSort

- QuickSort is a Divide and Conquer algorithm.
- It picks an element as pivot and partitions the given array around the picked pivot. There are different ways of quicksort that pick pivot in different ways.

Always pick the first element as pivot.
Always pick the last element as pivot.
Pick a random element as pivot.
Pick median as pivot.
Quicksort Algorithm Design

• **Divide:** Partition the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$.
  - $A[q] \leq$ each element in $A[q+1..r-1]$
  - Index $q$ is computed as part of the partitioning procedure
Partitioning in Quicksort

Partition(A, p, r):
  x, i := A[r], p – 1;
  for j := p to r – 1 do
    if A[j] < x then
      i := i + 1;
  A[i + 1] ↔ A[r];
  return i + 1
QuickSort Algorithm Design

• **Divide:** Partition the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$.
  ➢ $A[q] \leq$ each element in $A[q+1..r-1]$
  ➢ Index $q$ is computed as part of the partitioning procedure

• **Conquer:** Sort the two subarrays by recursive calls to quicksort
Recursive call Quicksort

QuickSort A[left...right]:

1. if left < right:
   1. Partition A[left...right] such that: //pivot = A[q]
      all A[left...q-1] elements are less than A[q],
      all A[q+1...right] elements are >= A[q]
   2. Quicksort A[left...q-1]
   3. Quicksort A[q+1...right]

2. Terminate
QuickSort Algorithm Design

• **Divide:** Partition the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$.
  - $A[q] \leq$ each element in $A[q+1..r-1]$
  - Index $q$ is computed as part of the partitioning procedure

• **Conquer:** Sort the two subarrays by recursive calls to quicksort

• **Combine:** The subarrays are sorted in place, no work is needed to combine them
Quicksort

• QuickSort is a Divide and Conquer algorithm.
• It picks an element as pivot and partitions the given array around the picked pivot. There are different ways of quicksort that pick pivot in different ways.

Always pick the first element as pivot.
Always pick the last element as pivot.
Pick a random element as pivot.
Pick median as pivot.
Quicksort Pseudocode

Quicksort(A, p, r):
   if p < r then
      q := Partition(A, p, r);
      Quicksort(A, p, q – 1);
      Quicksort(A, q + 1, r)

Partition(A, p, r):
   x, i := A[r], p – 1;
   for j := p to r – 1 do
      if A[j] < x then
         i := i + 1;
   A[i + 1] ↔ A[r];
   return i + 1

x: pivot
r: length of array
Example

Initially:

\[
\begin{array}{cccccccccc}
& & & & & & & & & & \\
& 2 & 5 & 8 & 3 & 9 & 4 & 1 & 7 & 10 & 6 \\
\end{array}
\]

pivot \((x) = 6\)

Iteration:

\[
\begin{array}{cccccccccc}
& & & & & & & & & & \\
& 2 & 5 & 8 & 3 & 9 & 4 & 1 & 7 & 10 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
& & & & & & & & & & \\
& 2 & 5 & 3 & 8 & 9 & 4 & 1 & 7 & 10 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
& & & & & & & & & & \\
& 2 & 5 & 3 & 8 & 9 & 4 & 1 & 7 & 10 & 6 \\
\end{array}
\]

Partition(A, \(p, r\)):

\[
x, i := A[r], p – 1;
\]

\[
\text{for } j := p \text{ to } r – 1 \text{ do}
\]

\[
\text{if } A[j] < x \text{ then}
\]

\[
i := i + 1;
\]

\[
A[i] \leftrightarrow A[j]
\]

\[
A[i + 1] \leftrightarrow A[r];
\]

\[
\text{return } i + 1
\]
Example

Partition(A, p, r):

\[
x, i := A[r], p - 1;
\]

for \( j := p \) to \( r - 1 \) do

if \( A[j] < x \) then

\[
i := i + 1;
\]

\[
A[i] \leftrightarrow A[j];
\]

\[
A[i + 1] \leftrightarrow A[r];
\]

return \( i + 1 \)

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 5 3 8 9 4 1 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 5 3 8 9 4 1 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 5 3 4 9 8 1 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 5 3 4 9 8 1 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 5 3 4 1 8 9 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 5 3 4 1 8 9 7 10 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>After final swap</td>
<td>2 5 3 4 1 6 9 7 10 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Next oteratopm:</td>
<td>2 5 3 4 1 6 9 7 10 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final:</td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quicksort Running time

- Quicksort is usually $O(n \log n)$
- Worst case: if the array is sorted to begin with, the running time will be $O(n^2)$
  - Array is already sorted in the same order.
  - Array is already sorted in reverse order.
  - All elements are the same
Partitioning in Quicksort

A[p..r]

P

Numbers less than p

Pivot

Numbers greater than or equal to p
Parallel quicksort

numbers less than p  pivot  numbers greater than or equal to p

Send the partition to another node

Number of layers depend on how many nodes

Sorted array
Results

Different size data

(ms)

(core)

Single node
Results

![Graph showing results](image)
Results

(ms)  8 nodes
130 120 110 100 90 80 70
1 2 4 8 16

(ms)  16 nodes
96 94 92 90 88 86 84
1 2 4 8 16

(core) (ms)  8 nodes
(core)
# Results

The following table and graph compare the performance of 8 and 16 nodes.

<table>
<thead>
<tr>
<th>(ms)</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>120.3148</td>
<td>92.78802</td>
<td>95.00372</td>
<td>95.9435</td>
<td>93.82286</td>
</tr>
<tr>
<td>16</td>
<td>85.76758</td>
<td>91.6233</td>
<td>92.7243</td>
<td>93.58568</td>
<td>93.80679</td>
</tr>
</tbody>
</table>

The graph shows the performance over different core counts for 16 and 8 nodes, indicating a slight improvement in performance with 16 nodes compared to 8 nodes.
Thank you
Reference

- Algorithms Sequential & Parallel: A Unified Approach (Dr. Russ Miller, Dr. Laurence Boxer)
- Python Program for QuickSort – GeeksforGeeks
- Sorting Algorithms – GeeksforGeeks
- MPI for Python — MPI for Python 3.1.3 documentation (mpi4py.readthedocs.io)