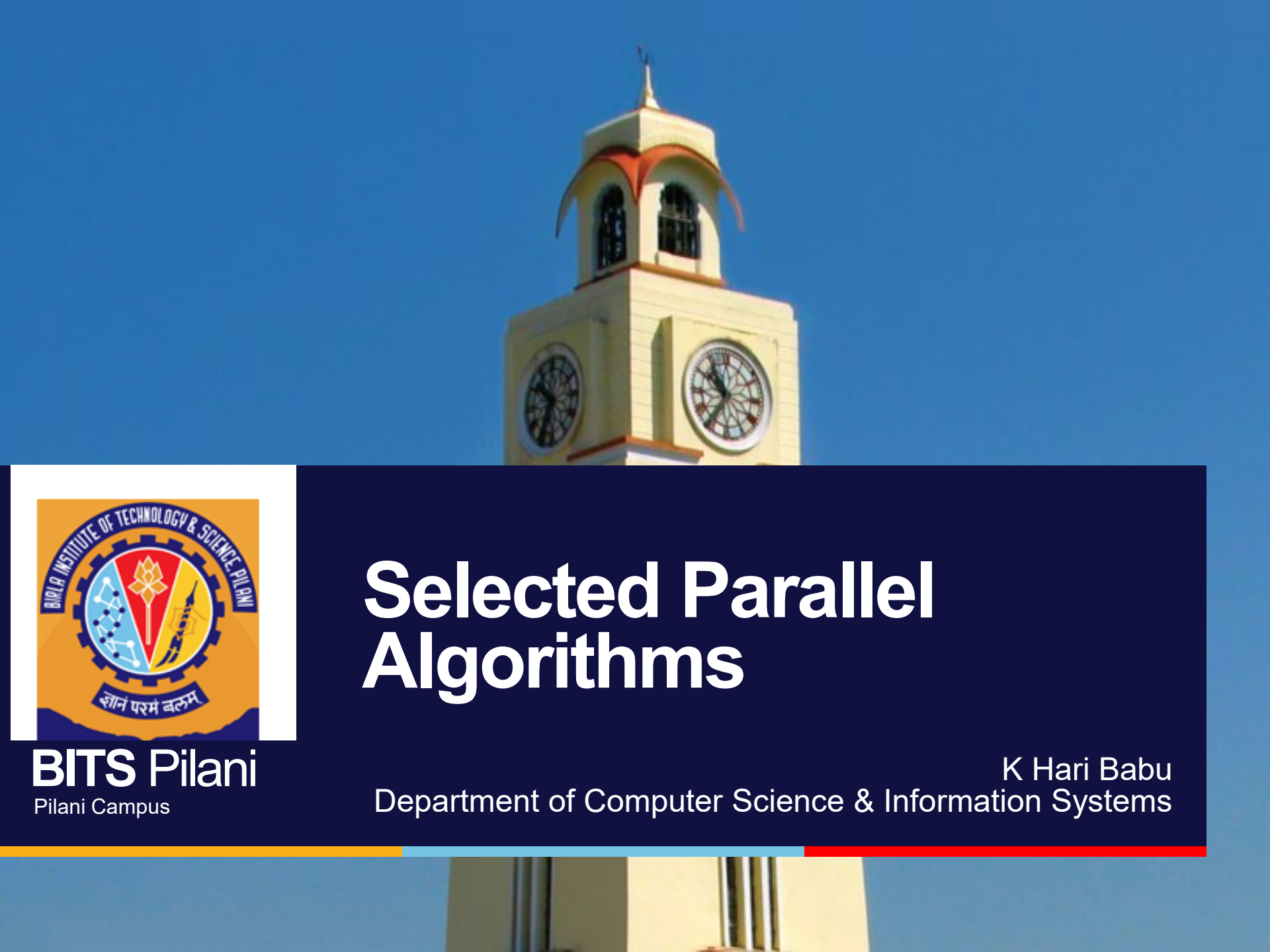


Selected Parallel Algorithms

BITS Pilani
Pilani Campus

K Hari Babu
Department of Computer Science & Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

Quick Sort

Quicksort

- Quicksort is one of the most common sorting algorithms for sequential computers because of its simplicity, low overhead, and optimal average complexity.
- Quicksort selects one of the entries in the sequence to be the pivot and divides the sequence into two - one with all elements less than the pivot and other greater.
- The process is recursively applied to each of the sublists.

Quicksort

- Quicksort is a divide-and-conquer algorithm that sorts a sequence by recursively dividing it into smaller subsequences.
  - Assume that the n-element sequence to be sorted is stored in the array A[1...n].
- Quicksort consists of two steps: divide and conquer
  - During the divide step
    - a sequence A[q...r] is partitioned (rearranged) into two nonempty subsequences A[q...s] and A[s + 1...r] such that each element of the first subsequence is smaller than or equal to each element of the second subsequence.
    - During the conquer step, the subsequences are sorted by recursively applying quicksort. Since the subsequences A[q...s] and A[s + 1...r] are sorted and the first subsequence has smaller elements than the second, the entire sequence is sorted.

Quicksort

- How is the sequence A[q...r] partitioned into two parts - one with all elements smaller than the other?
  - This is usually accomplished by selecting one element x from A[q...r] and using this element to partition the sequence A[q...r] into two parts - one with elements less than or equal to x and the other with elements greater than x. Element x is called the pivot.
    - The complexity of partitioning a sequence of size k is Q(k).

Quicksort

```
1.   procedure QUICKSORT (A, q, r)
2.   begin
3.       if q < r then
4.       begin
5.           x := A[q];
6.           s := q;
7.           for i := q + 1 to r do
8.               if A[i] ≤ x then
9.               begin
10.                  s := s + 1;
11.                  swap(A[s], A[i]);
12.              end if
13.          swap(A[q], A[s]);
14.          QUICKSORT (A, q, s);
15.          QUICKSORT (A, s + 1, r);
16.      end if
17.  end QUICKSORT
```

The sequential quicksort algorithm.

Pivot Selection

- Quicksort's performance is greatly affected by the way it partitions a sequence.
  - Consider the case in which a sequence of size k is split poorly, into two subsequences of sizes 1 and k - 1. The run time in this case is given by the recurrence relation $T(n) = T(n - 1) + Q(n)$, whose solution is $T(n) = Q(n^2)$.
  - Alternatively, consider the case in which the sequence is split well, into two roughly equal-size subsequences of $k/2$ and $k/2$. In this case, the run time is given by the recurrence relation $T(n) = 2T(n/2) + Q(n)$, whose solution is $T(n) = Q(n \log n)$. The second split yields an optimal algorithm
- The average number of compare-exchange operations needed by quicksort for sorting a randomly-ordered input sequence is $1.4n \log n$, which is asymptotically optimal
- There are several ways to select pivots.
  - For example, the pivot can be the median of a small number of elements of the sequence, or it can be an element selected at random. Some pivot selection strategies have advantages over others for certain input sequences.

Parallelizing Quicksort

- Quicksort can be parallelized in a variety of ways.
- Naïve parallel formulation
  - Initially start with a single process.
  - For every recursive call, create a new process to execute it
  - Upon termination, each process holds an element of the array, and the sorted order can be recovered by traversing the processes
  - This parallel formulation of quicksort uses n processes to sort n elements.
  - Its major drawback is that partitioning the array A[q ...r] into two smaller arrays, A[q ...s] and A[s + 1 ...r], is done by a single process.
    - Since one process must partition the original array A[1 ...n], the run time of this formulation is bounded below by W(n). This formulation is not cost-optimal, because its process-time product is $W(n^2)$.

Parallelizing Quicksort

- Performing partitioning in parallel is essential in obtaining an efficient parallel quicksort
- Why?
  - Consider the recurrence equation $T(n) = 2T(n/2) + Q(n)$
  - The term $Q(n)$ is due to the partitioning of the array. Compare this complexity with the overall complexity of the algorithm, $Q(n \log n)$.
  - From these two complexities, we can think of the quicksort algorithm as consisting of $Q(\log n)$ steps, each requiring time $Q(n)$ - that of splitting the array. Therefore, if the partitioning step is performed in time $Q(1)$, using $Q(n)$ processes, it is possible to obtain an overall parallel run time of $Q(\log n)$, which leads to a cost-optimal formulation
  - However, without parallelizing the partitioning step, the best we can do (while maintaining cost-optimality) is to use only $Q(\log n)$ processes to sort n elements in time $Q(n)$. Hence, parallelizing the partitioning step has the potential to yield a significantly faster parallel formulation.

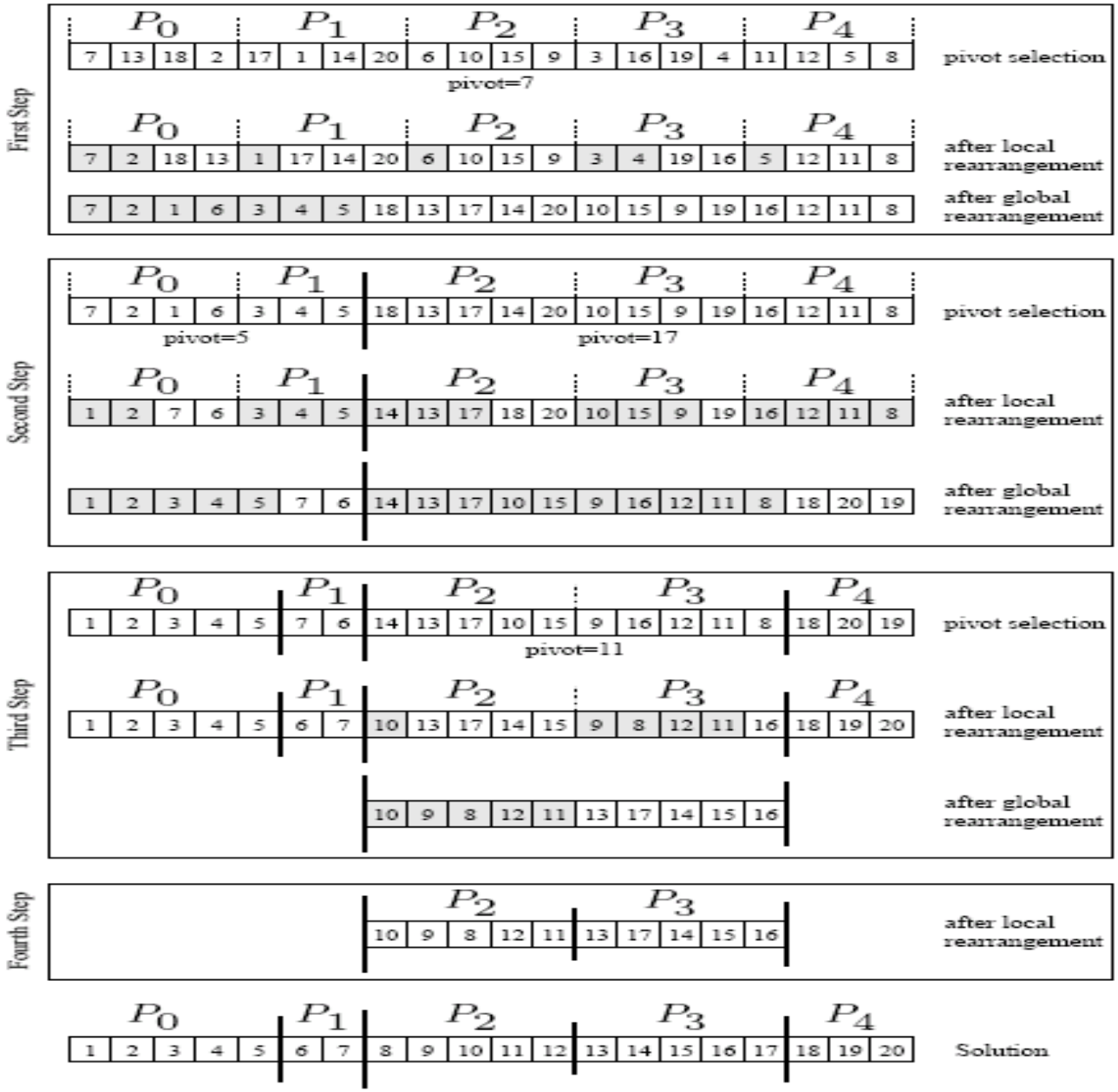
Parallelizing Quicksort

- Can we parallelize the partitioning step - in particular, if we can use n processors to partition a list of length n around a pivot in $O(1)$ time, we have a winner.
- This is difficult to do on real machines, though.

Parallelizing Quicksort: Shared Address Space Formulation

- Consider a list of size n equally divided across p processors.
- A pivot is selected by one of the processors and made known to all processors.
- Each processor partitions its list into two, say L_i and U_i, based on the selected pivot.
- All of the L_i lists are merged and all of the U_i lists are merged separately.
- The set of processors is partitioned into two (in proportion of the size of lists L and U). The process is recursively applied to each of the lists.

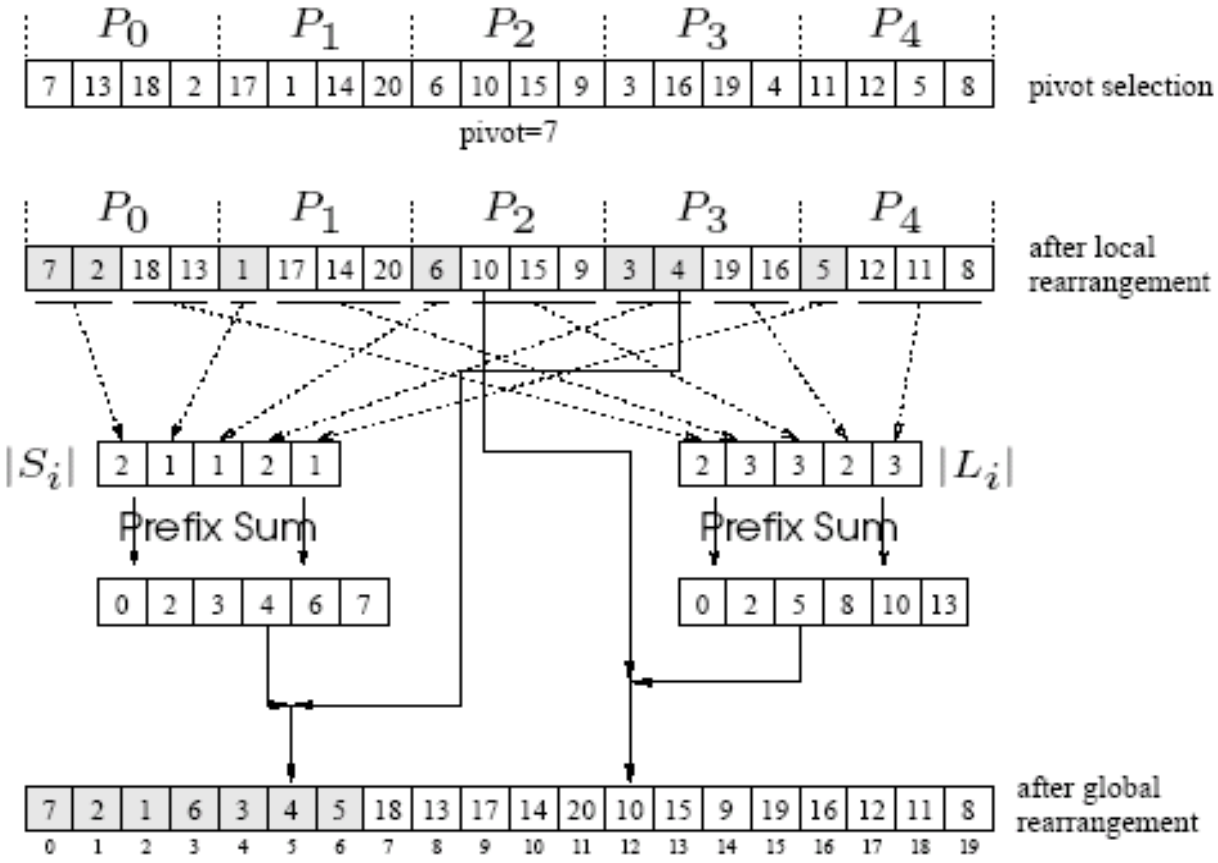
Shared Address Space Formulation



Parallelizing Quicksort: Shared Address Space Formulation

- The only thing we have not described is the global reorganization (merging) of local lists to form L and U.
- The problem is one of determining the right location for each element in the merged list.
- Each processor computes the number of elements locally less than and greater than pivot.
- It computes two sum-scans to determine the starting location for its elements in the merged L and U lists.
- Once it knows the starting locations, it can write its elements safely.

Parallelizing Quicksort: Shared Address Space Formulation



Efficient global rearrangement of the array.

Parallelizing Quicksort: Shared Address Space Formulation

- The parallel time depends on the split and merge time, and the quality of the pivot.
- The latter is an issue independent of parallelism, so we focus on the first aspect, assuming ideal pivot selection.
- The algorithm executes in four steps: (i) determine and broadcast the pivot; (ii) locally rearrange the array assigned to each process; (iii) determine the locations in the globally rearranged array that the local elements will go to; and (iv) perform the global rearrangement.
- The first step takes time $\Theta(\log p)$, the second, $\Theta(n/p)$, the third, $\Theta(\log p)$, and the fourth, $\Theta(n/p)$.
- The overall complexity of splitting an n-element array is $\Theta(n/p) + \Theta(\log p)$.

Parallelizing Quicksort: Shared Address Space Formulation

- The process recurses until there are p lists, at which point, the lists are sorted locally.
- Therefore, the total parallel time is:

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right) + \Theta(\log^2 p)}^{\text{array splits}}. \qquad (4)$$

- The corresponding isoefficiency is $\Theta(p\log^2 p)$ due to broadcast and scan operations.

Parallelizing Quicksort: Message Passing Formulation

- A simple message passing formulation is based on the recursive halving of the machine.
- Assume that each processor in the lower half of a p processor ensemble is paired with a corresponding processor in the upper half.
- A designated processor selects and broadcasts the pivot.
- Each processor splits its local list into two lists, one less (L_i), and other greater (U_i) than the pivot.
- A processor in the low half of the machine sends its list U_i to the paired processor in the other half. The paired processor sends its list L_i.
- It is easy to see that after this step, all elements less than the pivot are in the low half of the machine and all elements greater than the pivot are in the high half.

Parallelizing Quicksort: Message Passing Formulation

- The above process is recursed until each processor has its own local list, which is sorted locally.
- The time for a single reorganization is $\Theta(\log p)$ for broadcasting the pivot element, $\Theta(n/p)$ for splitting the locally assigned portion of the array, $\Theta(n/p)$ for exchange and local reorganization.
- We note that this time is identical to that of the corresponding shared address space formulation.
- It is important to remember that the reorganization of elements is a bandwidth sensitive operation.

References

- Chapter 8.1, 8.2, 9.4, 10.1-4 of text book.

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You