# Parallel Selection algorithm

**Dr.N.Sairam & Dr.R.Seethalakshmi**
**School of Computing,**
**SASTRA Univeristy,**
**Thanjavur-613401.**

## Contents



# 1. Parallel Selection algorithm

Parallel Selection algorithm is applicable for selecting the $k^{th}$ element in the very large array. The parallel selection uses n processors to select the desired data. Generally it is assumed that there is randomized distribution of data to n processors. The algorithm does not depend upon the number of data distributed to the processors, nor the values distributed. The input to the processor is from the array A and output is the element selected from array A with the rank i.

The selection algorithm for rank *k* assumes that input data A of size *n* are distributed uniformly across the *p* processors, such that each processor holds *n/ p* elements. The output, namely the element from *X* with rank *k*, is returned on each processor. The randomized selection algorithm locates the element of rank *k* by pruning the set of candidate elements using the iterative procedure as explained below:

Two splitter elements *(s*1,*s*2*)* are chosen that partition the input into three groups, A0,*A*1, and A2, such that each element in A0 is less than s1, each element in A1 lies between s1 and s2 and each in A2 is greater than s2. The desire is to have the middle group A1 much smaller than the outer two groups with the condition that the selection index lies within this middle group. The process is repeated iteratively on the group holding the selection index until the size of the group is small enough, whereby the remaining elements are gathered onto a single processor and the problem is solved sequentially. The key to this approach is choosing splitters s1 and s2 that minimize the size of the middle group while maximizing the probability of the condition that the selection index lies within this group. Splitters are chosen from a random sample of the input, by finding a pair of elements of certain rank in the sample. The algorithm is written below.

**Fast Randomized algorithm for processor $P_i$**

**Input**
n - Total number of elements
p - Total number of processors, labeled from 0 to p − 1
$A_i$- List of elements on processor $P_i$, where $|A_i| = n/p$
C-A constant
ξ- $\log_n$ of the sample size
τ- Selection coefficient
s- selection coefficient multiplier
ŋ- Min/Max constant
rank-desired rank among the elements

begin

```
 Set nᵢ = n/p
       While (n > C) and (|n − rank| > ŋ)
              Collect a sample Sᵢ fromAᵢ by picking nᵢ elements at random on Pᵢ .
              S = Gather(Sᵢ, p).
             Set z = TRUE and select τ
             While (z ≡ TRUE)
                    On P0
                     Select s1, s2 from S with the appropriate ranks
                    Broadcast s1 and s2.
                    Partition Aᵢ into < s1 and [s1, s2], and > s2, to give counts less,
       middle,
                    (and high). Only save the elements that lie in the middle partition.
                    c_less = Combine(less,+);
                    c_mid = Combine(middle,+);
                        If (rank € (c_less, c_less + c_mid ])
                            n = c_mid ;
                            nᵢ = middle;
                            rank = rank − c_less ;
                            z = FALSE
                       Else
                            On P0:  τ =s . τ
                    Endif
            Endwhile
       Endwhile
If (|n − rank|<= ŋ ) then
If rank<= ŋ  then the "minimum" approach is used, otherwise,  the "maximum"
             approach in parentheses, as follows is used.

        Sequentially sort our nᵢ elements in nondecreasing (nonincreasing) order using a
       modified insertion sort with output size |Aᵢ| = min(rank, nᵢ )
       (|Aᵢ| = min(n−rank+1, nᵢ )). An element that is greater (less) than the Li minimum
       (maximum) elements is discarded.

        Gather the p sorted subsequences onto P0.

        Using a p-way tournament tree of losers  constructed from the p sorted
       subsequences, (rank) (n − rank + 1) elements are extracted, to find the element q
       with selection index rank.
Else
       A = Gather(Aᵢ ).
On P0
       Perform sequential selection to find element q of rank in L;
Endif
       result = Broadcast(q).
end
```

The parallel selection algorithm finds its application in image processing for computer vision and remote sensing, computational aerodynamics and data mining of large databases. The algorithm can be implemented in shared memory system and cluster systems.