I. Introduction

- In computer science, the problem of maximum sum subsequence or maximum subarray is the task of finding the contiguous subarray within a one-dimensional array i.e. X[1…n] of numbers which has the maximum possible sum.
- The array often comprises of both positive as well as negative numbers.
- Some other properties of Maximum Sum Subsequence are:
    i. If all the numbers in the array are positive, then the solution is sum total of all the numbers in the array.
    ii. On the contrary, if all the numbers in the array are negative, the solution is an empty set.
- Hence the problem is only interesting when both positive and negative numbers are involved and that is the case I have considered to implement the solution.
- For Example,
  If a = [-2, 1, -3, 4, -1, 2, 1, -5, 4], the maximum sum subsequence array would be {4, -1, 2, 1} with sum 6.

II. Approach

a. Sequential Algorithm

- A sequential algorithm with a lower bound of $\Omega(n)$ exists that examines every element of the array sequentially to determine the contiguous array as follows:

- *Global_Max* ← *x0*
- *u* ← *0* {Start index of global max subsequence}
- *v* ← *0* {End index of global max subsequence}
- *Current_Max* ← *x0*
- *q* ← *0*
- For i = 1 to n -1, do
  - If *Current_Max* ≥ 0 Then

    Current_Max ← *Current_Max + x1*
  - Else

    Current_Max ← *x(i)*

    q ← *i*
  - End Else
  - If *Current_Max > Global_Max, Then*

    Global_Max ← *Current_Max*

    u ← *q*

    v ← *i*
  - End If
- End For

- Since the loop is performed Θ(n) times, the running time of the algorithm is Θ(n), which is optimal and all *n* entries of the array must be examined.

b. Parallel Algorithm

- Parallel algorithm uses a parallel prefix approach to compute max sum subsequence.
- Our algorithm would target a Θ(log n) time algorithm on a machine with Θ(n/log n) processors.
- Such an algorithm would be both time and cost-optimal.