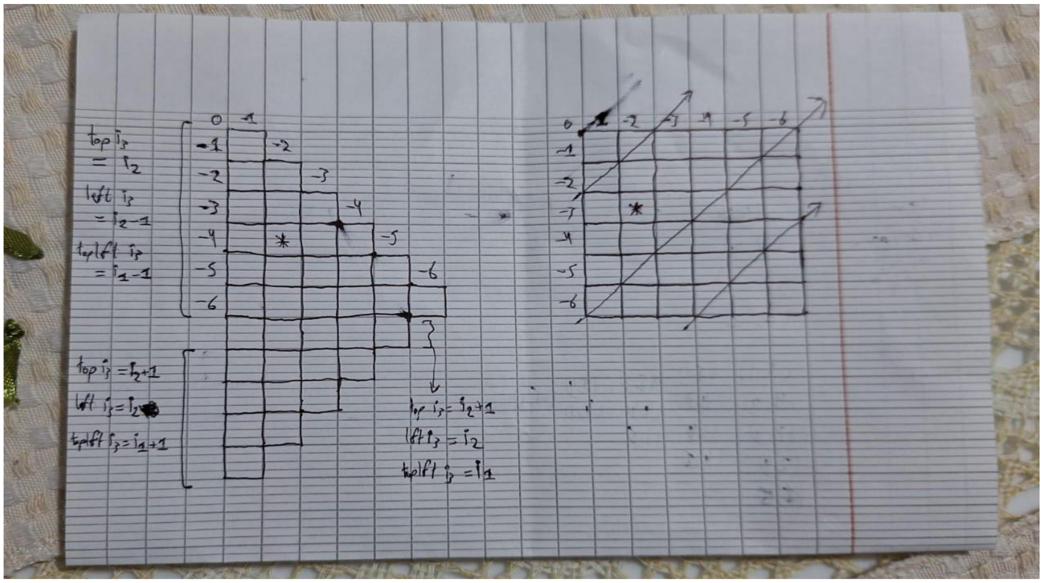
1st Optimization

Our Intuition:

- We are allocating a very huge amount of memory of the array of matrices.
- A lot of none coalesced accesses to global memory and stores, for top, left, and topleft of each output element.
- Also, we are accessing the global memory to compare the corresponding elements of the sequence pair to compute each of the output elements of each matrix.
- So first, we need to store the sequence pair in shared memory.
- We noticed that we only need for each iteration, the two preceding iterations, thus for each anti diagonal to be computed, we need the preceding two anti-diagonals to get the left, top, and topleft of each output element in the current anti-diagonal.
- So, no need to allocate this huge array of matrices, we only need three buffers instead of each matrix, and those can fit in shared memory.
- So, we've gotten rid of none coalesced memory accesses, by having all our accesses from shared memory.
- Now the indexing is changed as illustrated in the below picture that we developed, and hence this will be done on three phases (1st triangle half of the matrix reaching the main diagonal, the main diagonal itself, the 2nd triangle half of the matrix), each phase has its proper indexing.



top i_3 = i_2
left i_3 = $i_2 - 1$
topleft i_3 = $i_1 - 1$
0
-1
-2
-3
-4
-5
-6
*
top $i_3 = i_2 + 1$
left $i_3 = i_2$
topleft $i_3 = i_1 + 1$
top $i_3 = i_2 + 1$
left $i_3 = i_2$
topleft $i_3 = i_1$
0
-1
-2
-3
-4
-5
-6
-1
-2
-7
-4
-5
-6
*