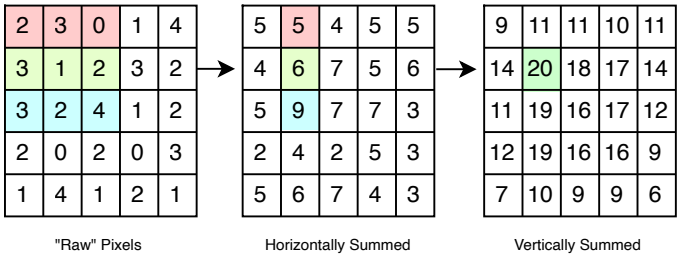
Algorithm Restructuring

Instead of building a global SAT, which requires communication between horizontal and vertical blocks of calculation, take advantage of the fact that we know the fixed kernel size ahead-of-time. Additionally, we can take advantage of trading off larger FPGA area to do many simultaneous operations instead of trying to minimize operations on a standard CPU.

The sum of pixels over a kernel area can be calculated by calculating the horizontal sums for each row of the kernel, then summing these to get the total:



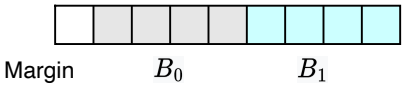
Thus, by calculating the horizontal sum for the known kernel width at each pixel, we can trivially calculate the area sum given only the single central column of the kernel's position.

Calculations are localised and there doesn't need to be a global synchronization between horizontal blocks to calculate a full SAT.

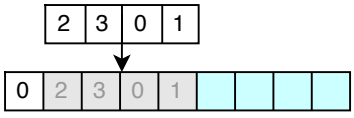
Implementation

Using Kernel halfwidth $K_w = 1$ and processing block size $B = 4$ here for simplicity of demonstration - this means the total kernel is $2K_w + 1 = 3$. In real scenarios, $K_w = 3$ and $B = 16$ or $B = 32$.

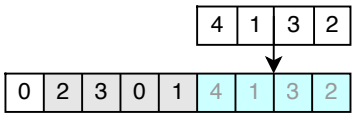
Create a read buffer of size $K_w + 2B$ pixels:



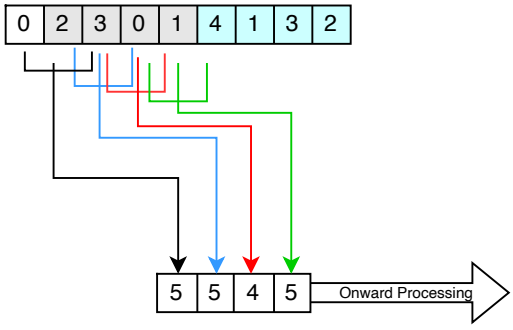
As the initial (pre-loop) step, read the first block of pixels into B_0:



At the start of the loop, read the next block into B_1:

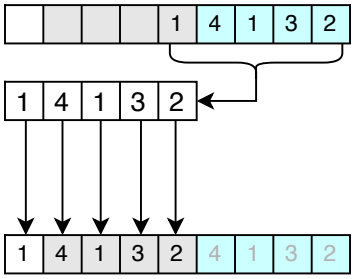


We now have enough data to calculate, for every pixel in B_0, the sum of every other pixel within the *horizontal* kernel centered on that pixel, with $(2K_w + 1)B$ addition operations:

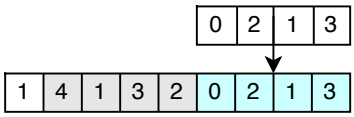


This calculated block can now be sent off for further processing. For now we stay with the read buffer.

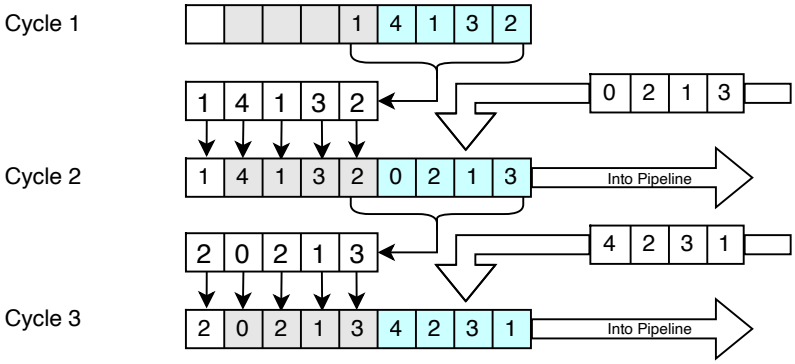
We now shift the entire buffer left by $B + K_w$ pixels, discarding the pixels shifted off the buffer:



And finally, continue the next iteration of the loop by reading the next block into B_1 again:



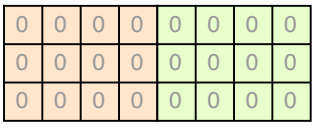
This can be pipelined! Since we never write back to the buffer when calculating sums, we can do the shift and block read in a single cycle, siphoning off the completed state into a pipeline process:



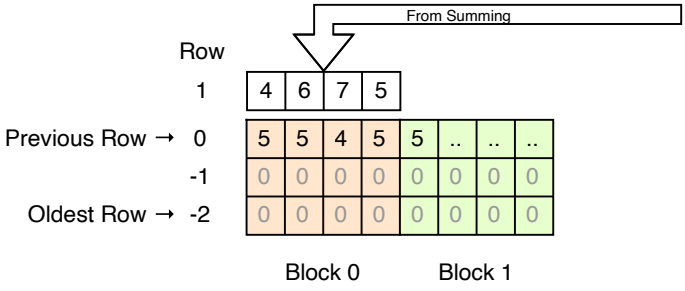
Using the Summed Blocks

As output from above, we get a per-cycle stream of "Summed" blocks, each of which already has it's horizontal kernel sum calculated. We need to sum these along the vertical dimension to calculate the sum over the full kernel centered on each pixel.

Declare a storage for enough simultaneous rows to cover the full $H_k = 2K_h + 1$ for a full row of blocks - so BN_B pixels, to cover an entire module ($N_B = 2$ shown)

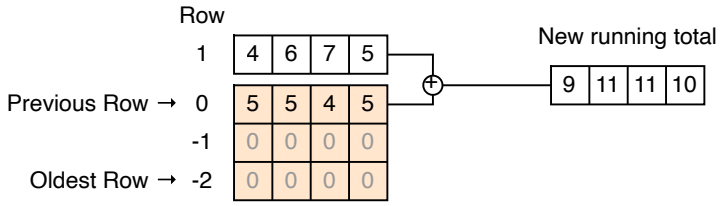


Now, every time we get a new block, we know which block, on which row this is. (We've also jumped into a state where we already did this process for the first row, and we're now receiving second row blocks):

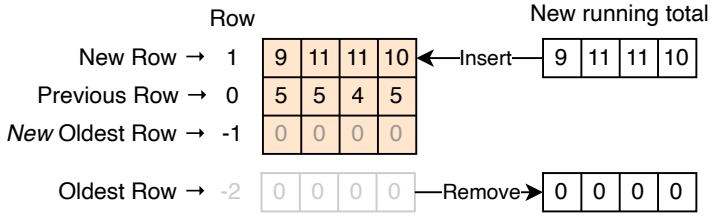


From this point, the process is independent from which whichever block we are processing. We've already computed the horizontally-dependent parts of the eventual total.

We want to maintain a running sum for the current kernel range, so add the total from the previous row



We now shift the entire array down, removing the oldest row (and keeping it) and putting the new total on the top:



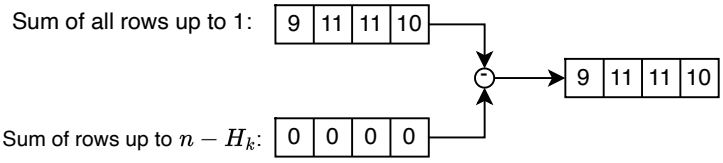
In practice, instead of shifting the entire array, we use a fixed buffer and calculate

$$i_{\text{oldest}} = i_{\text{next}} = y \mod H_k$$

for the oldest row and insertion position, and, for the running sum of the previous row:

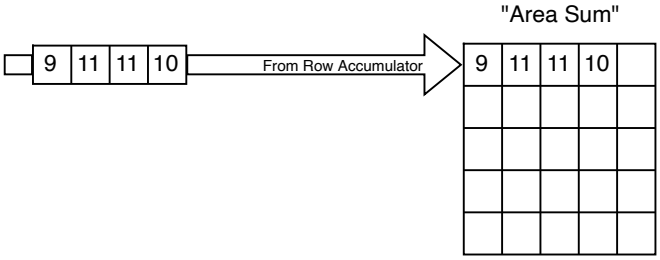
$$(y-1) \mod H_k$$

Now, we subtract the oldest row - the sum of all rows "below" it - off of the current running total



We've now calculated the total over just the rows stored in the buffer array - and since these values already include the horizontal contribution, this value is now the summed total over the kernel area centered at $y - K_h$.

Ignoring output values for the first $K_h -1$ rows (as they represent the area covered by negative rows), we can now write this into the "Final Sum" memory for this centered row:



This gives a value identical to that of forming a full SAT and calculating the summed area over kernel width for each centre pixel.