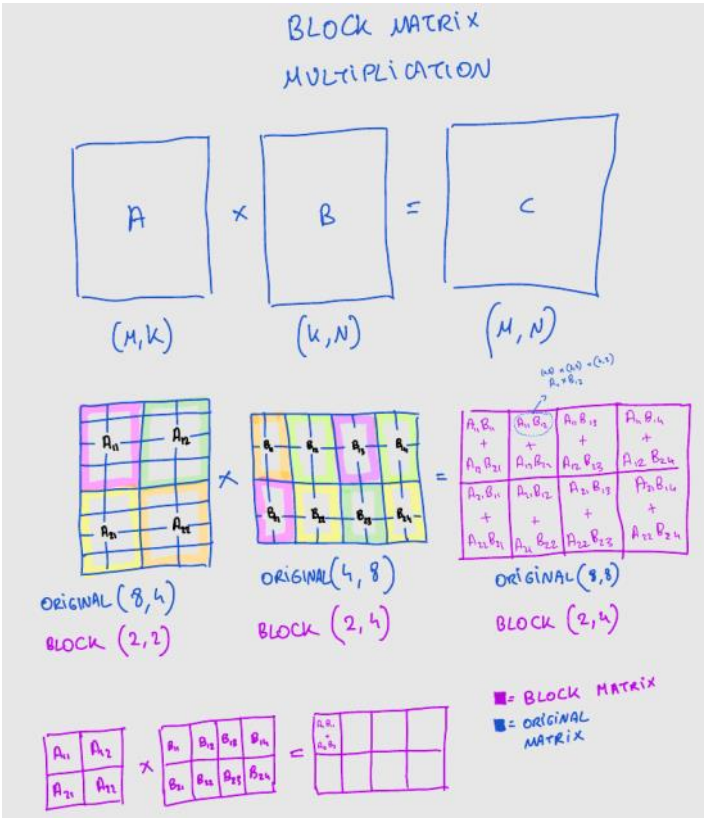
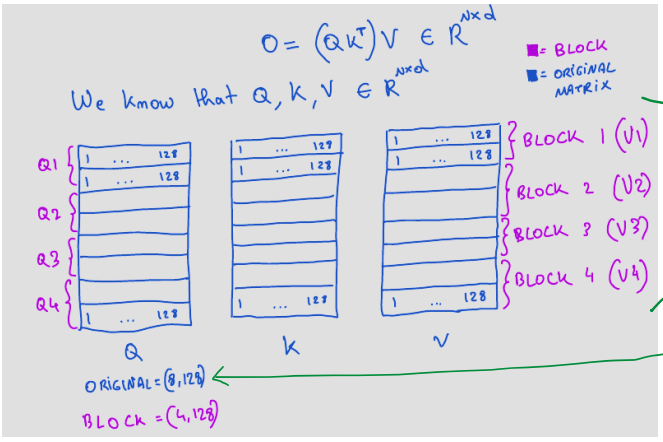
Block Matrix Multiplication (how it works on a GPU)

Friday, October 24, 2025 10:35 AM





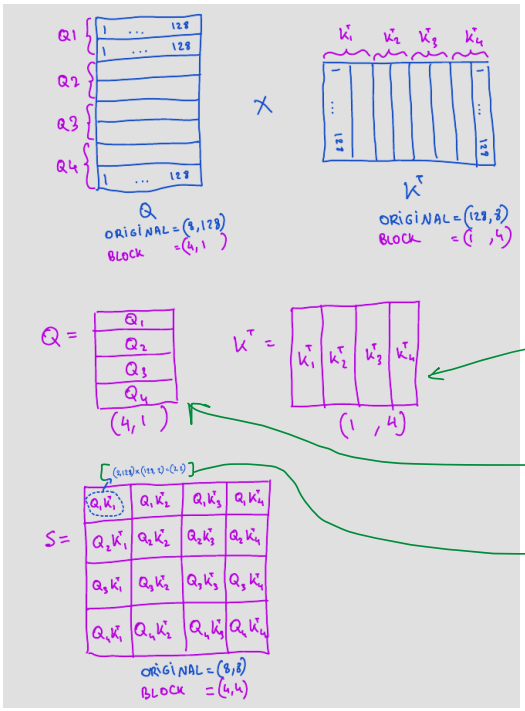
We have chosen 1 Block to process 2 Tokens here (we can choose this however we want, but keeping in mind that the Blocks will be accessed in GPU memory, so all the "(number of Blocks [grid_size], number of Threads [block_size])" selection for your CUDA kernel will HAVE to be done accordingly)

Triton optimizes this!!

Number of Tokens

Consider (sequence_length, embedding_vector_dimension) : (8, 128)

Dimension of each Token (aka model_dimension)



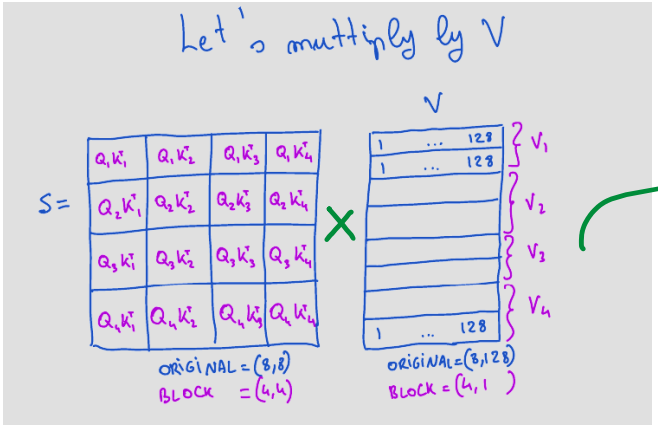
Internally, each K_i.T is (128, 2)
Reason: we chose our Block to be (2, 128), so each K_i is (2, 128)

Internally, each Q_i is (2, 128)
Reason: that's what we've chosen our Block to be

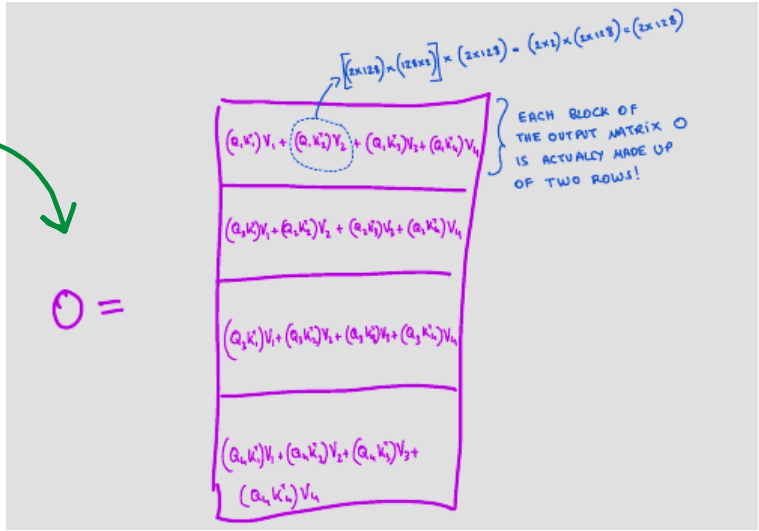
$(Q_i \cdot K_i^T)$

each dot_product(Q_i, transpose(K_i)) here:

- if we forget that the Blocks even exist (for a moment) and think of it in terms of the original dimension, the matrix multiplication looks like: (2, 128) @ (128, 2) = (2, 2)
- Notice that this is EXACTLY what we get doing the regular "full Q matrix @ full K matrix = full S matrix" matrix multiplication
- Very cool! This means Block Matrix Multiplication actually works!
- Effectively, this means we can allocate 1 Thread per 1 Block on the GPU (remember our "1 Block -> 2 Tokens" mapping choice) , effectively parallelizing our ENTIRE matrix multiplication computation.
- Extremely cool! Why? Faster matrix multiplication compute due to parallelizing!
- So what then? Faster Attention computation!



what we get



PSEUDO CODE

```
FOR EACH BLOCK Qi
    Oi = ZEROES(2, 128)   // OUTPUT IS INITIALLY ZEROES
    FOR EACH BLOCK KJ
        Oi ← Oi + (Qi KJ^T) VJ
    END FOR
END FOR
```

Pseudocode of how we'll do Block Matrix Multiplication (in actual Triton code) to get the full O matrix

Softmax will be applied to this, which is then matmulled with V_J