# Step1. Workload Analysis

- Number of attention blocks: 1000
- Embedding dimension (d): 2048
- Query/Key/Value dimensions (Dk and Dv): 768
- Sequence length (L): 512 (fixed)
- Batch size (B): 1 – 128 (varied)

**P1-1**: The total number of MAC operations to complete one inference

For each attention block (since there are a total of 1000 blocks for each inference), the only steps that involve BMM are step 1, step2 and step 6, since scaling, masking, and softmax are ignored. Because the batch size is 1, 3 BMM in step 1 is of shape [L, d]*[d, Dk/Dv], BMM in step 2 is of shape [L, Dk]*[Dk, L], while BMM in step 6 in shape [L, L]*[L, Dv]. Therefore, for each attention block, the number of MAC is:

$$Dk/Dv \times L \times d \times 3 = 2415919104(step1)$$

$$Dk \times L \times L = 201326592(step2)$$

$$L \times L \times Dv = 201326592(step6)$$

$$2415919104 + 201326592 + 201326592 = 2818572288(MACs/block \bullet inference)$$

For the entire inference run, the number of MAC is thus:

$$2818572288 \times 1000 = 2818572288000(MACs/inference)$$

**P1-2**: The minimum number of PEs to complete the LLM inference within one second, 1.5GHz frequency

To complete in 1s under 1.5GHz frequency, there are total of $1.5 \times 10^9$ cycles. The assumption, 100% utilization of all PEs, means all PEs are in execution during these cycles. Assuming each MAC takes one cycle to complete, the minimum number of PEs is:

$$Total\ MACs/Total\ Cycles = ceil(\frac{2818572288000}{1.5 \times 10^9}) = ceil(1879.048) = 1880$$

# Step2. Hardware Design Parameter Exploration

2048=2^11 PEs

**P2-1**: PE array organization

(512,768)*(768,512); (512,512)*(512,768)

The above dimensions are the ones involved in the BMM computation. It can be seen that 512 and 768 share common factors 1, 2, 4, 8, 16, 32, 64, 128, and 256. to maximize PE utilization, we need to 1. make the organization dimension divisible by the data dimension (the reasoning behind this is that when it is not fully divisible, the last block of PEs will inevitably have idle PE(s) that does not have available data to do computations which leads to lower utilization) and 2. we need to fully leverage data reuse, e.g. a 4 * 512 can achieve 100% PE utilization under given assumptions, but the data reuse is poor on the dimension of 4.

Rule 1. makes sure that the final organization needs to have their common factors as their dimensionalities, which maximizes PE utilization. Rule 2. asks us to try to maximize overall data reuse on both dimensions. Thus, we can choose from 32*64 and 16*128, which two dimensions are closer to each other to both accommodate reuse in two dimensions. It is more preferred to use 32*64 under the same logic.

**P2-2**: Minimum memory size that minimizes the DRAM traffic (i.e., DRAM is only accessed for fetching input Q, K, V tensors and sending the final output back to the

DRAM)

The entire computation is first to do a BMM of Q and K_t and then another BMM of previous BMM output and V. Given the fact that V does not need to be loaded into the memory when it needs to compute Q and K_t, the memory required for the first BMM is just the size of Q and K. After the first BMM is computed, the result will be stored back to the memory. At this point, it needs V and the output from the first BMM. As a summary:
The first BMM needs at least L*Dk*N + Dk*L*N=2N*L*Dk=786432 numbers.
The second BMM needs at least L*L*N+L*Dv*N=655360 numbers.
Considering the memory size should be fixed, we should take the maximum of the two to be the minimum size of the memory. In this case, assuming 8-bits data representation, the memory size should be 786432 * 2 Bytes=1536 KB = 1.5 MB.

# Step3. Dataflow Exploration

32*64 Systolic Array

**P3-1**: Which dimensions to parallelize over rows and columns of PE array? What is the processing order?

Since we only consider the projections, the dimensionalities involved are N(1), d(2048), L(512), and Dk(768).
My choice of parallelization is as follows: the row (32) is mapped to Dk (768), and the column is mapped to d (2048). In this case, it will be:
TemporalMap(32, 32) Dk
TemporalMap(64 , 64) d
and it requires a total number of 24*32=768 temporal tiles.
The reason is that by making spatial mapping of the largest dimensions, it maximizes reuse for more data. It on the other hand reduces the number of iterations of temporal mappings, which directly affects overall latency.
The processing order from slow to fast is B, L, Dk, and d. The fastest should be the one with the largest dimension, which is expected to be fully parallelized. The slowest one if batch since it does not change.

**P3-2**: If batch is not 1, how does it change P3-1?

If my previous statement is correct, then it will not change the mapping. B is in range [1, 128] and it is still one of the smallest dimensions.

# Step4. Systolic array and Dataflow Description

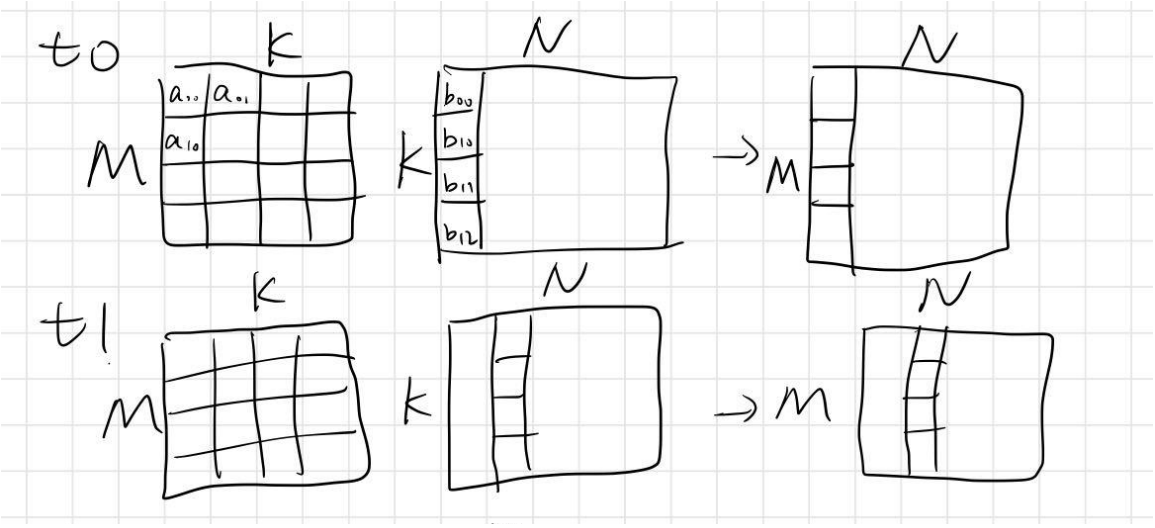
[M: 16, N: 4, K: 16]
Assume the following for your estimation:

- MAC computation on each PE requires one clock cycle
- data forwarding between adjacent PEs requires one clock cycle
- All the necessary data are pre-loaded to sufficiently large on-chip global buffers for matrices A and B
- You don't have to count the cycles to send output from the global shared buffer (L2) to DRAM.

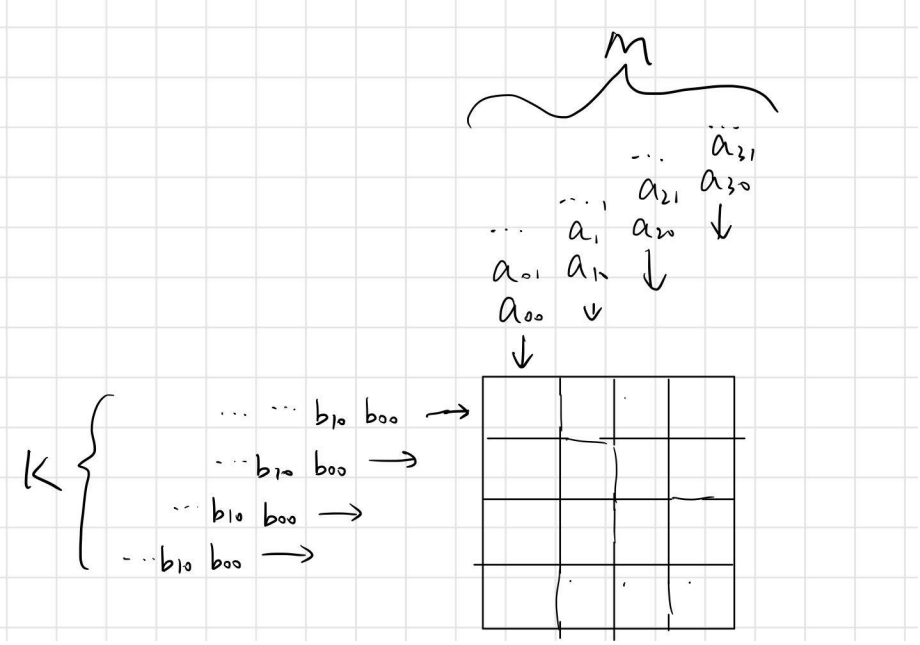
(Hint: The assumptions above create the same environment as the systolic array examples in lecture slides)

**P4-1**: Assuming a systolic array of 16x16 PEs (row: parallelize the K dimension, column: parallelize the M dimension), estimate the latency of the systolic array for computing the matrix multiplication. Answer in cycle counts.

And since M=K=16=PE corresponding dimensions, no need to update M and K. The problem becomes the latency to update N, as the scratch below indicates:




When considering the total latency, the flow of data is illustrated as follows:

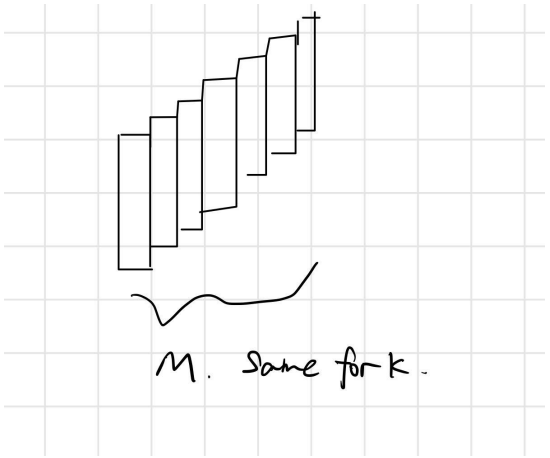



The data forwarding between PE rows should be temporally separated by 2 cycles, since each PE requires one cycle for the computation. **The assumption I make here is that the PE cannot do computation and data forwarding at the same cycle.** In this case, (take a00 and a01 from the first column of the PE array as example,) a00 and a01 from the first column of the PE array should behave like this:

| | |
|---|---|
| t0 | a00 (and b00) is loaded to the PE array [0,0] |
| t1 | a partial sum is computed |

| | |
|---|---|
| t2 | a01 and b10 are loaded to the PE array [0,0] |

Under this assumption, it can be observed that each data (number) takes 2 cycles for movement and computation. Given the temporal shifts in both M and K, the total latency can be illustrated by the shift diagram below:




Each rectangle represents 32 cycles, which corresponds to 16 data on that dimension. The total latency on both M and K are the same, given they are both 16 and the PE array is also 16*16. Thus, the total latency=(16-1)+32=47 cycles.

**P4-2**: Please describe the dataflow given in Problem 4-1.

You can use a loop nest or the data-centric directives (50 pts)

* Hint: Please review Lecture 12-13 slides

The reason I got the answer below is from the scratch in P4-1. As it can be observed that M and K are parallelized with a size of 1 and an offset of 1. N is not parallelized and the update of each time is 1. It can also be viewed from data indices from each PE.

TemporalMap1, 1)N
SpatialMap(1,1) K
TemporalMap(16, 16)M
cluster(16)
SpatialMap(1, 1)M