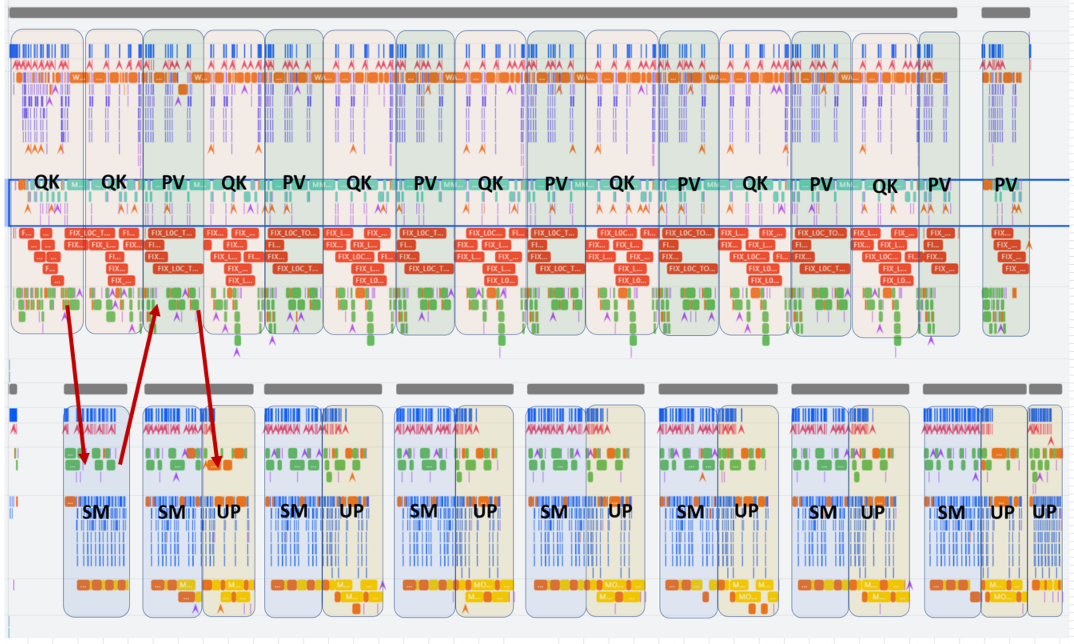
Int8 Optimizations: (Implemented from Zhentao)

(I)



<u>Original Pipeline:</u>

(1) chunk=1

QK, QK, PV, QK, PV, QK, PV, PV, __

__, SM, SM, UP, SM, UP, SM, UP, UP

<u>New Pipeline:</u>

(2) chunk=2

QK, QK, PV, PV, QK, QK, PV, PV, __ SideNote: No need to change workspace.

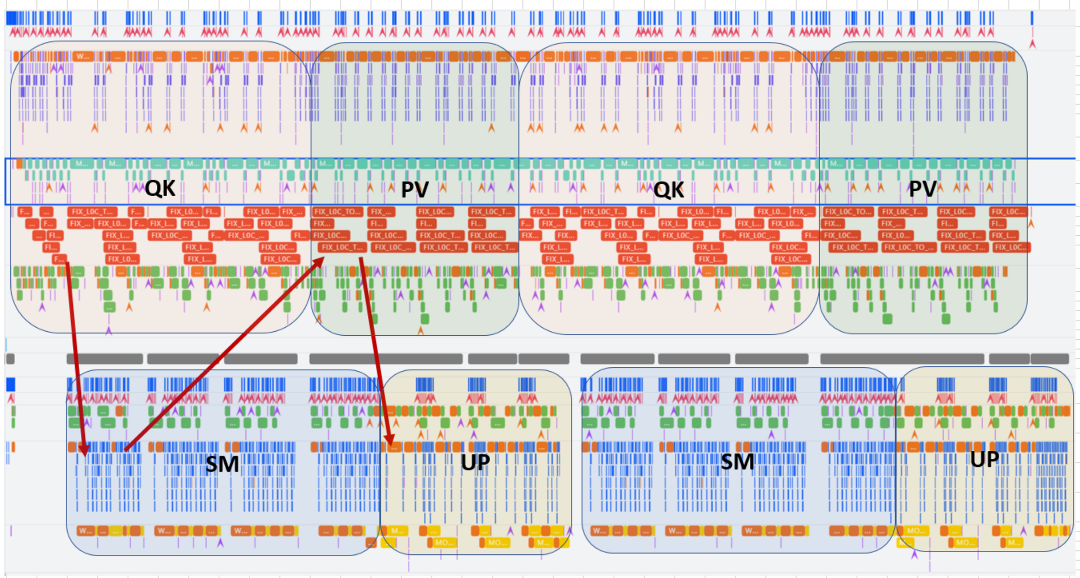
__, SM, SM, UP, UP, SM, SM, UP, UP

(3) chunk=4

QK, QK, QK, QK, PV, PV, PV, PV ... SideNote: The gm workspace of S, P, Otmp would be doubled. But since in Int8,

__, SM, SM, SM, SM, UP, UP, UP, UP... it still has the same workspace size compared to fp16.

Observation:
1. The granularity of each stage is small, which requires perfect pipeline runtime to make the bubble disappear, which has less robustness.
2. In the actual run(not the simulation), the SM duration is long, so there is a big bubble in the end for PV waiting for the SM.



Purpose of the New Pipeline with large chunksize: Enlarge the granularity of the pipeline between CV.

Side Advantaged:
1. .Utilized more L1 Buffer with pingpong
2. leave the room for full cube utilization
3. leave the room for UP memory transfer reduction.

Side Disadvantaged:
1. Doubled the HBM usages on S, P, Otmp.

Result:

Bs1_K4096_C24
Original Pipeline: us
New Pipeline: us
Improvement: **4.5%**

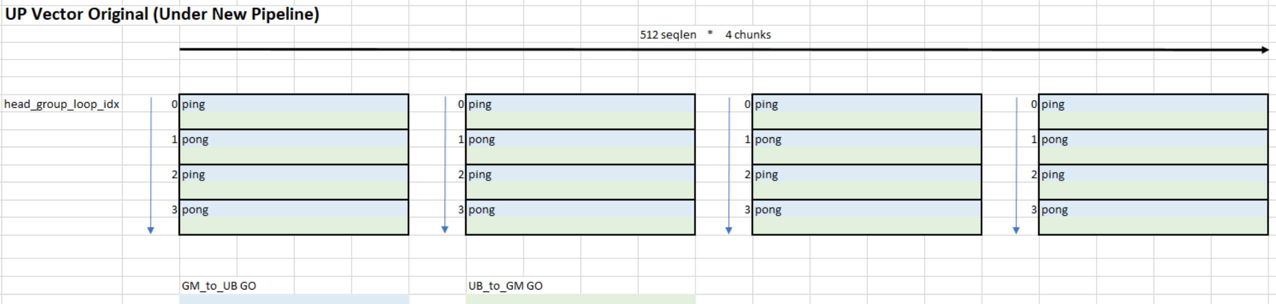
Bs1_K2048_C24
Original Pipeline: us
New Pipeline: us
Improvement: **9.2%**

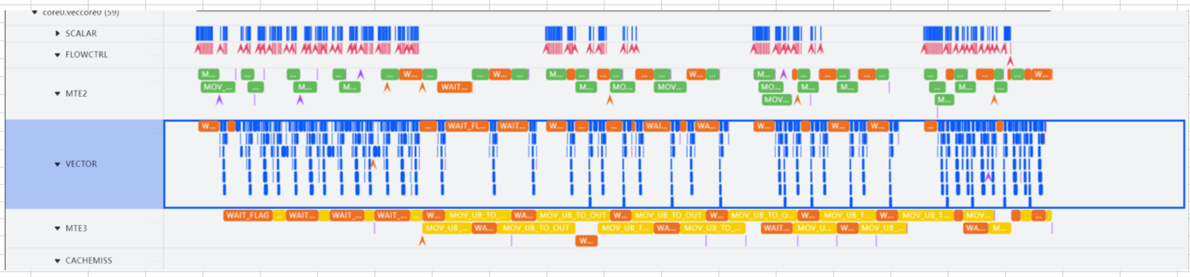
(II)

Background: UP Algorithm (last stage in FlashMLA MTP)

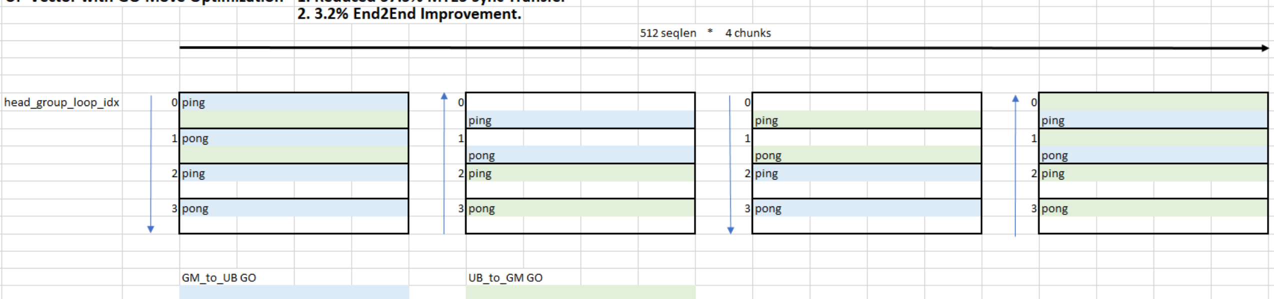
Load lo (Int32), dequant to lo (Fp32)
Load go (Fp32)
dm = exp(dm)
go = go * dm_block
go = lo + go
for the last 512 seqlen, go = go / gl_block
go = castfp32to16(go)
Move go to gm

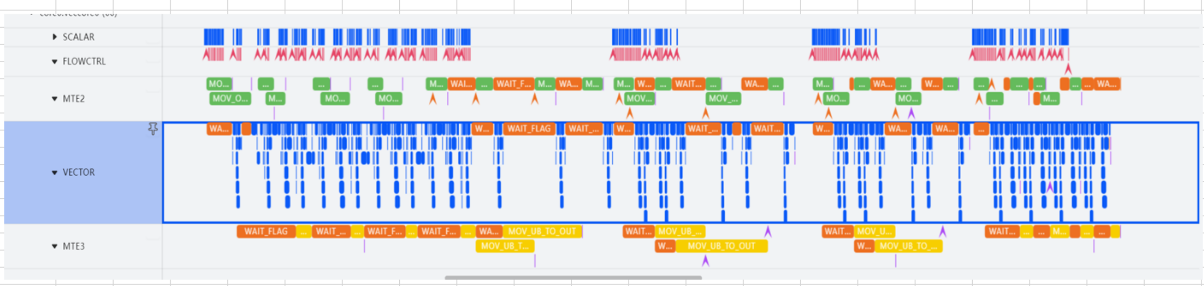
UP Vector Original (Under New Pipeline)





UP Vector with GO Move Optimization 1. Reduced 37.5% MTE3 Sync Transfer
2. 3.2% End2End Improvement.

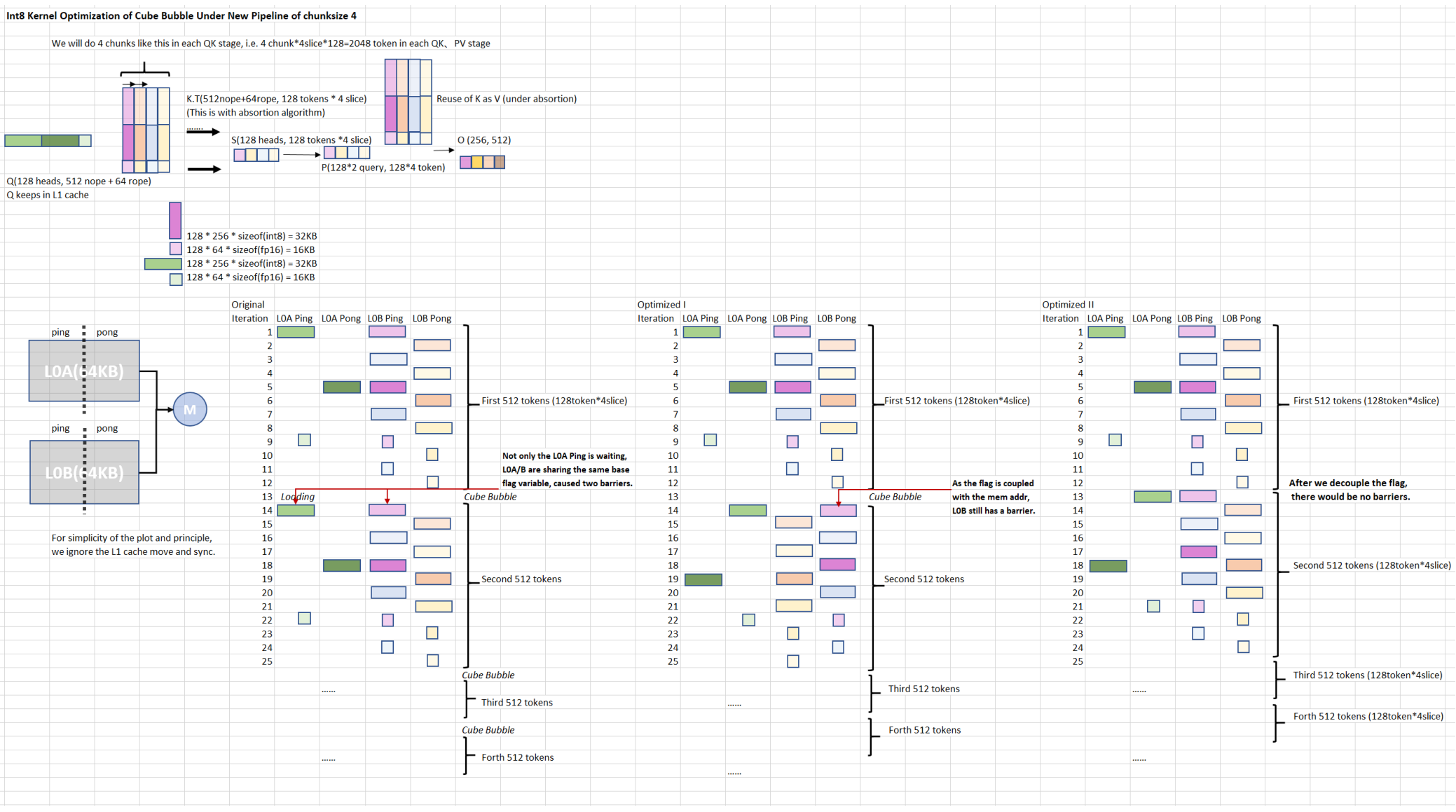




	without UP Optimiza	with UP Optimization	
	OpComLibV7	OpComLibV9	V9/V7
b1_k512_c24			1.0702
b1_k2048_c24			1.02816
b1_k4096_c24			1.02482
b2_k512_c24			0.94719
b2_k2048_c24			1.03273
b2_k4096_c24			1.03943
b4_k512_c24			0.98463
b4_k2048_c24			1.04529
b4_k4096_c24			1.06415
b8_k512_c24			0.98329
b8_k2048_c24			1.03882
b8_k4096_c24			1.02439
b16_k512_c24			1.03539
b16_k2048_c24			0.99793
b16_k4096_c24			1.02525
b24_k512_c24			1.05869
b24_k2048_c24			1.06629
b24_k4096_c24			1.06168
b48_k512_c24			1.02313
b48_k2048_c24			1.04312
b48_k4096_c24			1.07853
		AVG	1.03205

(III)

Reduction Strategy:



Int8 Overall Improvement (includes other optimzation from teammates): 19%

	OpComLibV0 (us)	OpComLibV11 (us)	V11/V0
b1_k512_c24			1.07568
b1_k2048_c24			1.39753
b1_k4096_c24			1.22115
b2_k512_c24			1.10849
b2_k2048_c24			1.20156
b2_k4096_c24			1.17252
b4_k512_c24			1.125
b4_k2048_c24			1.24277
b4_k4096_c24			1.19364
b8_k512_c24			1.13337
b8_k2048_c24			1.25688
b8_k4096_c24			1.22447
b16_k512_c24			1.04
b16_k2048_c24			1.23246
b16_k4096_c24			1.2045
b24_k512_c24			1.10615
b24_k2048_c24			1.23452
b24_k4096_c24			1.22962
b48_k512_c24			1.13204
b48_k2048_c24			1.2534
b48_k4096_c24			1.25591
		AVG	1.19246

Side Explanation:

min{max{ }} → min{max{ }} → min{max{ }} *Nah! You can't let the second optimzation takes all the contribution, optimizations are sythetic.*

The real case is more complicated since each component can not be seperated independently.

Other Optimizations:

Some solid optimization without End2End Improvement (but definetely no harm):

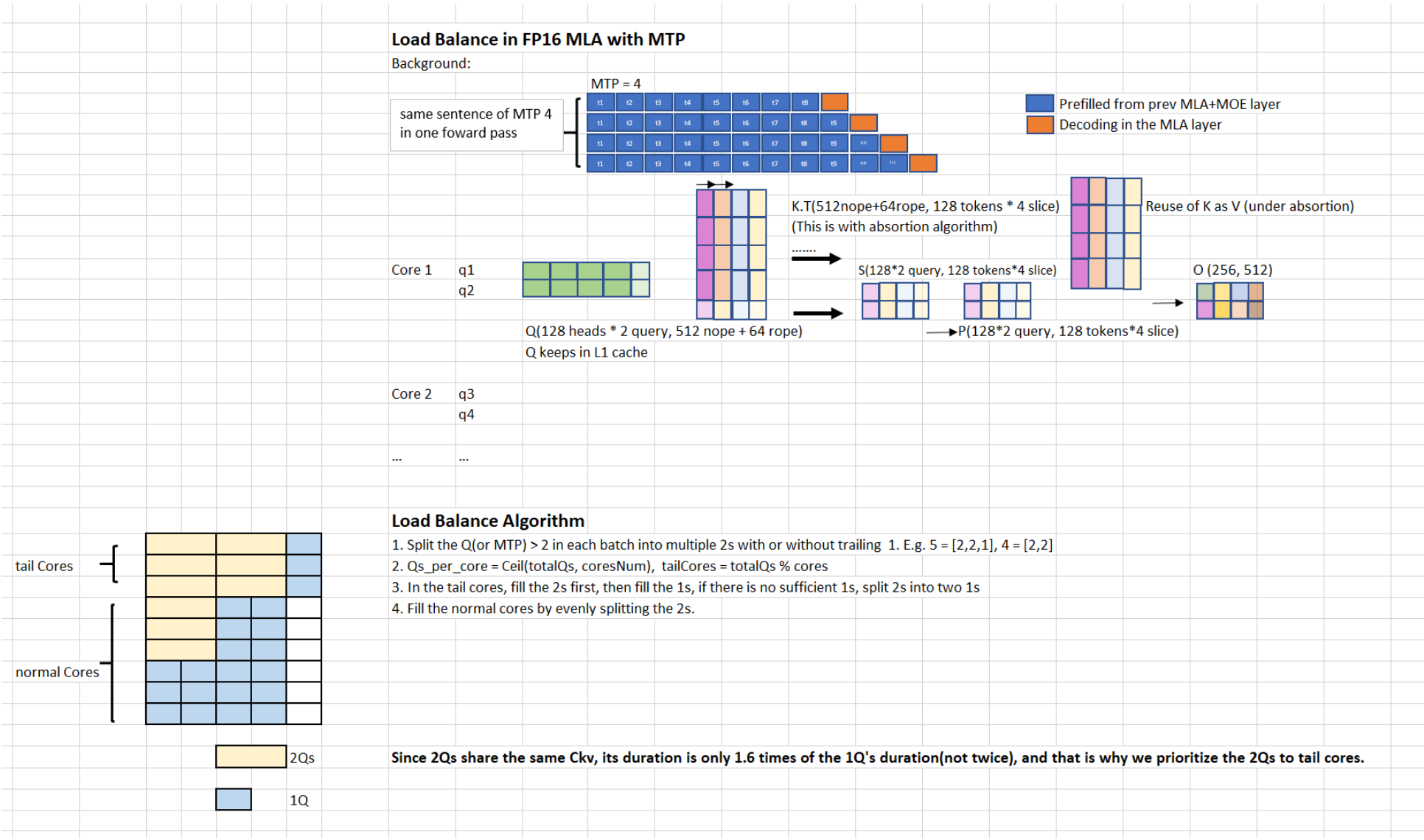
1. S/P Layout optimization in Int8 case.
2. Setting two stages of cross-core synchronization for QK stage, one for nope(S_nope int32), one for rope(S_rope fp32). So once the S_nope is calculated, it can move to vector core and do dequantization.

Other concerns: multi-platform; code vulnerability.

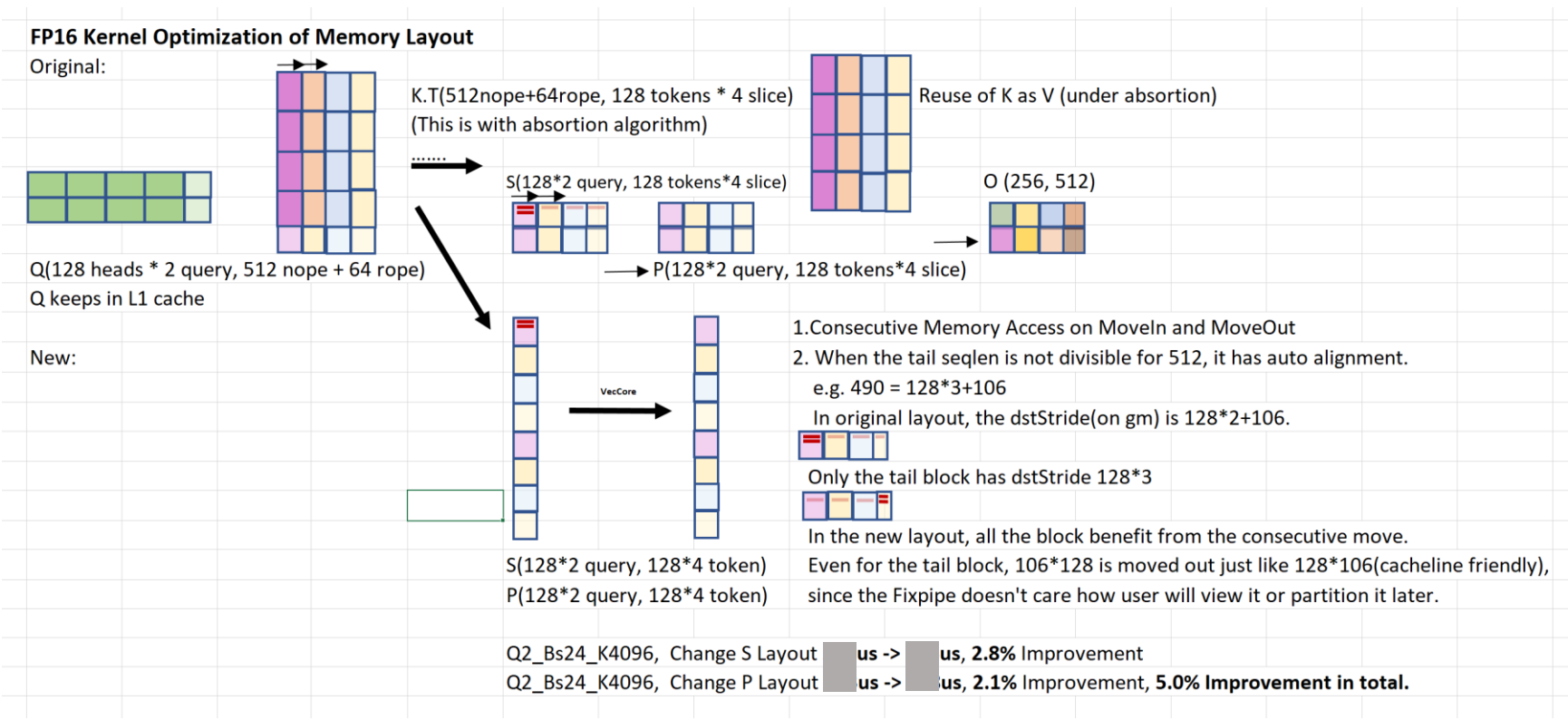
Teammates' optimizations: tail rowsum/max; cube/L0C unitflag; nope/rope seperation; UP stage pingpong (me participated in this as well); enlarge the headsgroup size in SM stage; synchronization reduction, etc.

FP16 Optimizations: (Implemented from Zhentao)

(I) Yicai He did the first version but Zhentao Fan rewrote it with a different algorithm.



(II)



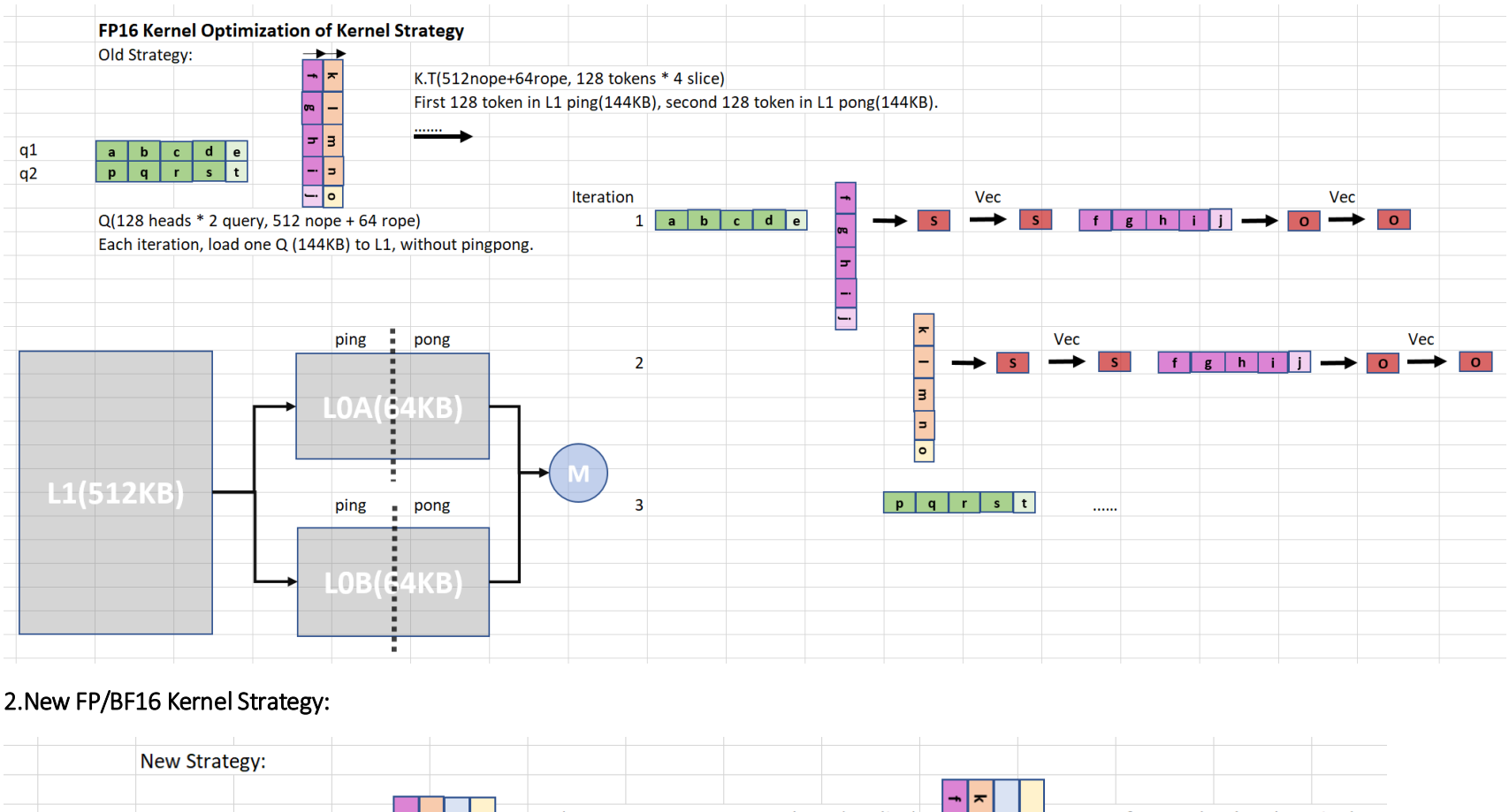
FP16 Overall Improvement (includes other optimzation from teammates):

On XX cluster, with optimized operator, the **full model**(not the single operator) inference has **5.XX%** speed up on MTP1_Bs32/64_Prefill2048_Decode2048 case, and **1.XX%** speed up on MTP1_Bs16_Prefill2048_Decode2048. (Results from the model team.) The single **operator** improvement on MTP1 case is also **around 20%.**

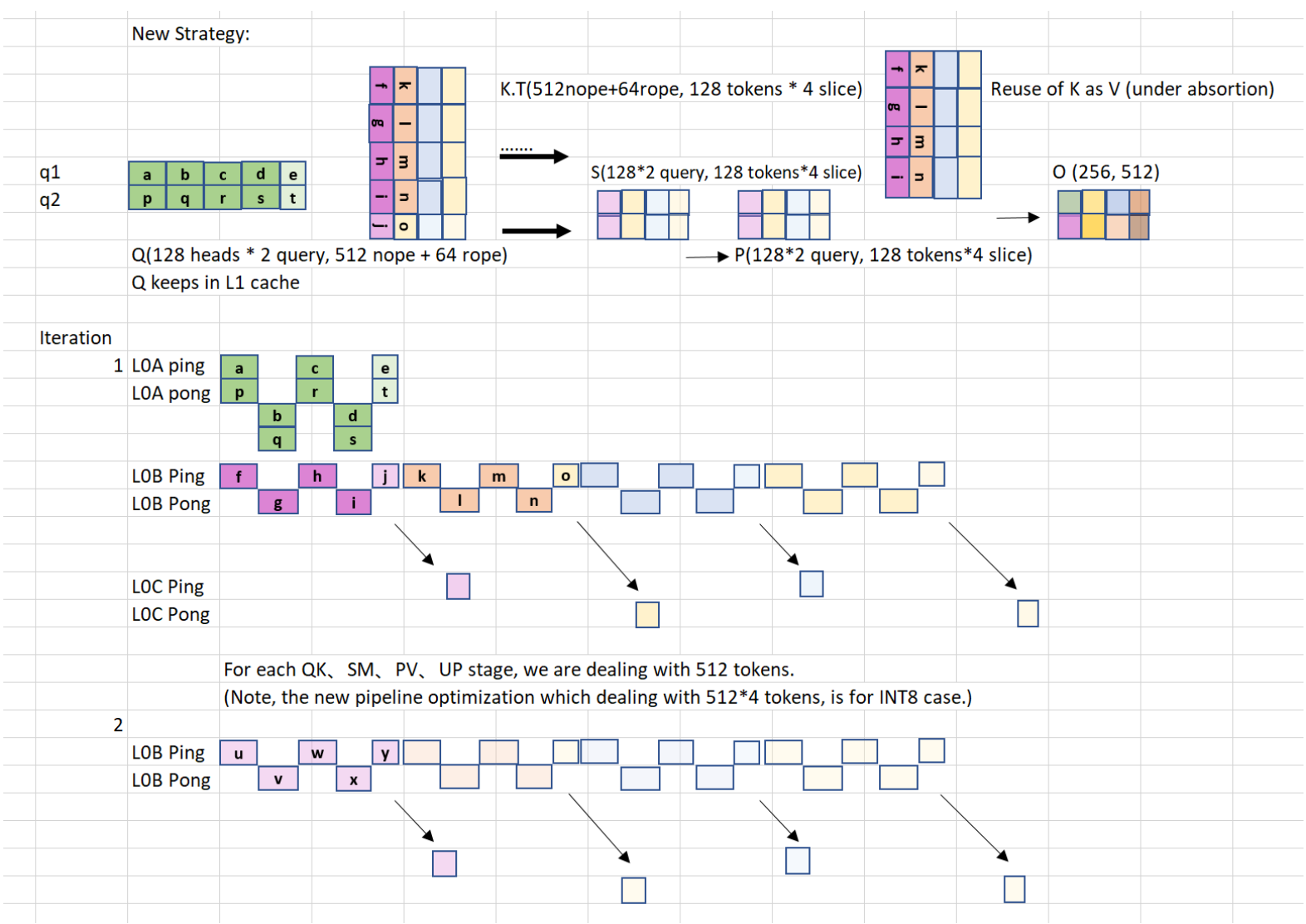
For FP16 kernel, the main speed up comes from the load balancing(shown above) and a new Q2 kernel strategy(shown below). Most code of this new kernel strategy (below) is developed by my teammates, and I was mainly for the bugs they stuck with, solved quite lot critical bottlenecks during that development.

Appendix:

1.Old FP/BF16 kernel strategy:



2.New FP/BF16 Kernel Strategy:



3. MLA Inference Operators on Ascend (Execute Sequentially):

1. MLAPO(i.e. Preprocess Operator)[rope, Wk absortion, data preparation for paged attention]
2. MLAPA(i.e. Paged Attention) [flashMLA] ←This is what we are optimizing at
3. Einsum [Wv absortion)]