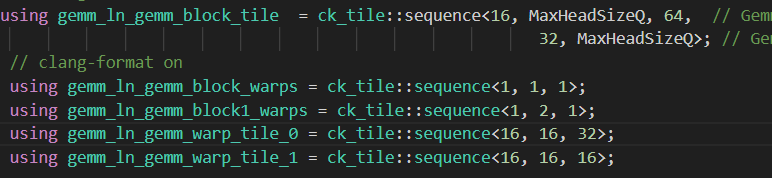
2024/12/25:

1. Gemmlngemm:

对于batch小于16的情况，我们想按照multihead的思路将kM固定为16,并且当KN1>=32时,将warp排布在KN1上。

对于上述思路我们做了如下修改:

1. block块和warp的配置



1. 对于shape配置的修改，增加了Gemm1的warp配置

A screen shot of a computer code

Description automatically generated

1. 对policy中的RKBlockGemm传入的warp配置改为warpgemm1

A screen shot of a computer program

Description automatically generated

最终结果当seqlen\_k>16时会错误，现象如下图：

A computer screen with white text

Description automatically generated

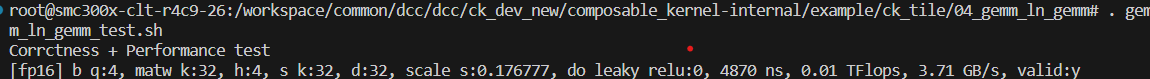
但当将上述（4）中的 RKBlockGemm中的 warpgemm1配置(1,2,1)改回warpgemm0配置(1,1,1)，结果会正确。如下图：

修改：

A screenshot of a computer program

Description automatically generated

结果：



1. Multihead:

1)

A screen shot of a computer program

Description automatically generated

如上图当KN1>=32时 为什么gemm1\_warps的配置是固定为<1,2,1>， 为什么不是KN1/16？

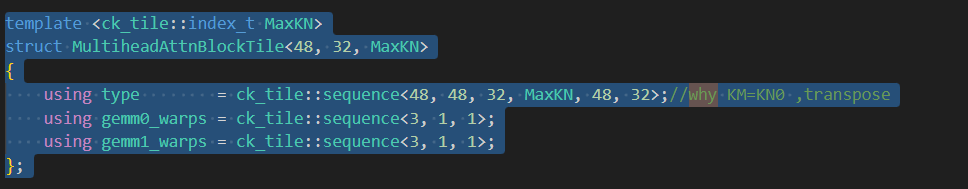
2)

A screen shot of a computer program

Description automatically generated

当前方法读取A 时是将数据加载到vgr，如果先读到lds 再存到vgr，哪种方式的效率更高？

3）

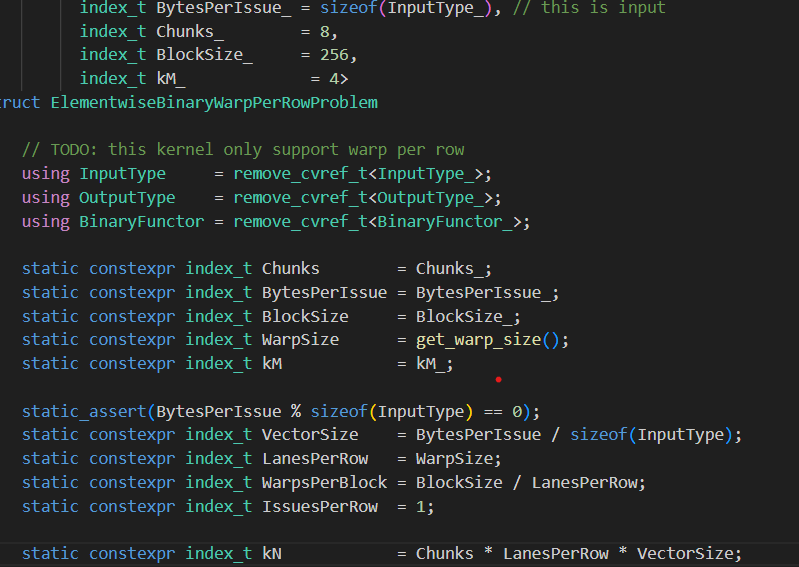


如上图为什么所有配置中的KM都和KN0相等?

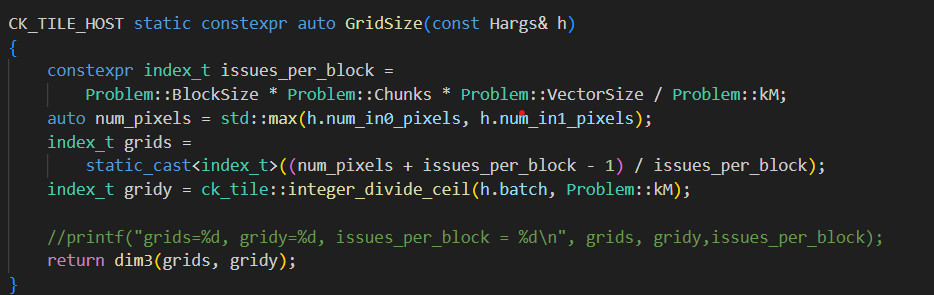
1. Elementwise :

我们想通过像multihead一样的方式配置elementwise的blcok块，改变warp的排布方向，将warp排布在M方向，但仍保持Alignment根据dim的实际对齐8/4/2情况，并且仍然考虑chunk和vectorload，但性能对比flatten情况较差。我们做了如下修改：

1. 增加KM维度实际为block的warp数，KN维度为一个warp在一行上处理的长度（一个warp处理一行）。如下图



1. 根据（1）中的 Block快大小改变BlockDim.x和BlockDim.y的分配，BlockDim.x=batch/KM , BlockDim.y=dim/(一个warp处理长度)。如下图：

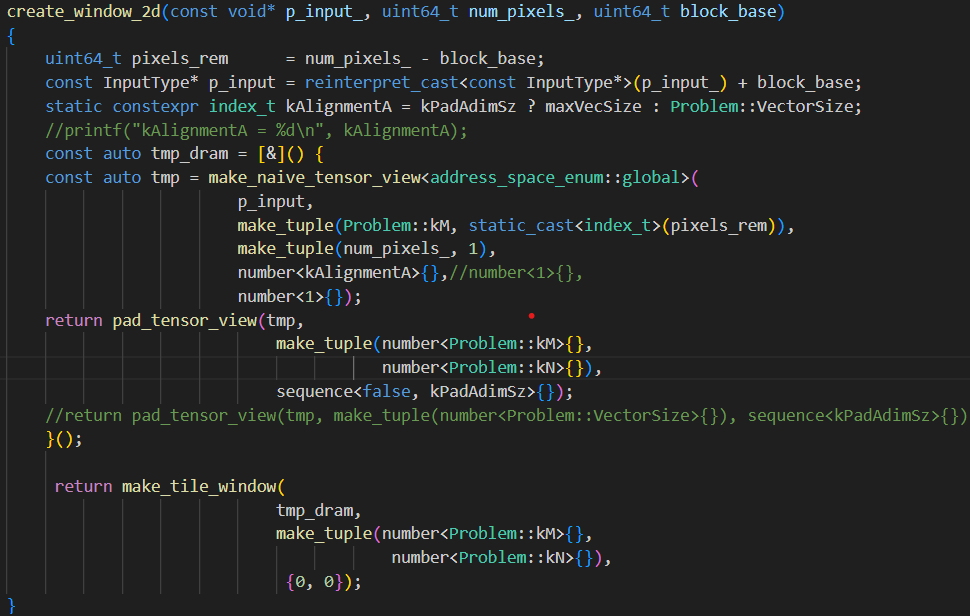


1. 我们将policy中的input和output的distribution改为二维，如下图

A computer screen shot of a program

Description automatically generated

1. 二维的input和output的tile\_window配置如下图：



1. 最后我们将pipline中的 计算方式改为通过sweep\_tile\_span将二维的input和一维的bais相加。

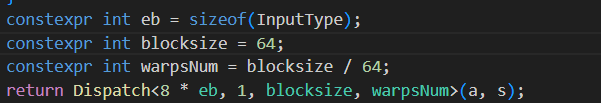
A screen shot of a computer code

Description automatically generated

1. 结果：对于同一种shape :batch=2000,dim=512

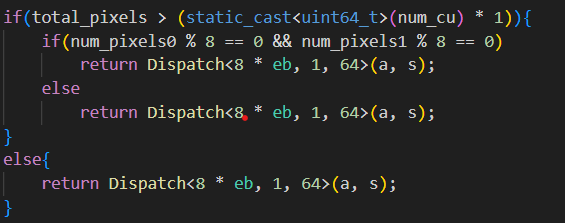
2维block:时间为49us，dispatch配置如下





1维blcok:时间为7.8us，dispatch配置如下





7）疑问：对于（6）中的dispatch配置 均为Blocsize=64,Alignment=8,chunk=1,warp=1。所以对于（6）中相同的输入形状，一个warp恰好处理一个batch,二维的配置也等同于一维并且chunk和Alignment没有浪费，但最终两种配置的性能差的很远。