AI??????2D??

1. 卷积天然的数据复用度是 Dot的 9倍，对于芯片的压力更小
2. 算力缩放是一个非常重要的问题，涉及架构各代之间的稳定性，保护客户的价值
3. L1/L2/L0 怎么支持 reshape或者 swizzel
4. 不同的 layout （ NHWC等） 以及 BPI BPK FF
5. 不同数据精度的支持和混合计算
6. L0 L1 Fusion的支持
7. 存储的 mapping 利于运算和 fusion
8. feature*weight vs weightT*featueT

?2D Dot?????

1. 指令表达为小尺寸，对不同的计算需求（ 1D、2D）的 fusion比较友好
    1. 但是小尺寸，需要每个单元的 L0进行 broadcast来复用给其他单元，节省 L1 的带宽需求，增加了 2D的复杂度
    2. 小尺寸对 load/store的要求也比较高
    3. 需要 **2D**的计算表达有一个好的抽象，同时配合非常精密的同步，解决上述问题
    4. 为了在各代的代码不发生变化，需要硬件 /编译器有自动合并 /优化的能力
2. cuda因为需要兼容 simt的编程，复用 warp内 thread 的寄存器，同时为了拼接更多的硬件做一个更大的 2D（为了 PPA好）
    1. 整体上单个 thread表达的数据计算尺寸比较小
    2. 硬件自动进行合并成大的 2D指令
    3. 硬件自动处理同步
    4. 硬件自动处理数据搬运
    5. 因为需要协调多个 sm subcore，甚至是 block group，灵活性一样是个大问题
3. XXU4.0利用多个 thread之间的手动给的 footprint order信息来组织多个 thread的数据共享
    1. 左数每次都从 L1load重复 load，右数缓存在 L0，并且各个 subcore复用
4. 2D-1D-2D的 Fusion
    1. 涉及到多次计算后结果的数据精度（位宽）不匹配的问题，需要合并和拆封来保持操作数的宽度一致
        1. 8bit int8 需要 64个组成 512bit，而 32bit只要 16个
    2. 不同的操作的输入输出，可能需要有装置的操作
    3. layout的区别: NCHW NHWC
    4. 卷积的 stride不等 1
5. 预测: 2D算力会无限的膨胀下去， 1D和 2D的比例会比较固定
6. 精细化控做到 L1，则所有的抗 latency的容量就会开在 L1，消耗: L1_base x latency，如果是在 L0，则 L0_base比 L1_base小很多。节省很多面积
7. **swizzle**和 **renaming**都可以解决 **L0**的 **bank conflict**， **renaming**会消耗大量的面积

????

1. 算力和存储器（ SRAM）都不能做到单体很大
1. 很大的单体有利于软件使用方便
2. 多个小的存储器实例（ bank）组成一个大的抽象存储器
2. 物理单元之间很难方便做到 cycle by cycle的同步，必须使用额外的同步逻辑
3. 硬件电路虽然可以并行，理论算力很高，但是 latency是必然存在的

NV Blackwell

从 PTX、SASS可以看到 AI面对的问题和很多解决方法都是共同的。

https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#tensorcore-5th-generation-

family-instructions https://docs.nvidia.com/cuda/cuda-binary-utilities/index.html#blackwell-instruction-set

1. L0的 memory变化
1. 显式的手动管理 tensor memory ，是不是 2D/1D更像是独立的一个 NPU/DSA单元？
2. Tensor计算直接使用专门的 Tensor memory，做 memory to memory的Tensor计算
，且支持软件动态分配
3. 1D 可以直接 load/store这块 memory
4. 这样让 L0可以动态，目的估计是 破除 2D/1D 每写一种 L0 fusion就要写一份 code 示例， Fusion算子可以隔离开发
2. 更大的矩阵形状 128/256，降低 2D带宽需求
3. 支持 TPC 内 2个SM之间的 2D 协同计算，看起来是 2个core支持的读数据复用
4. 支持 OCP-MX的 micro-scaling
5. 标量 thread issue Tensor计算指令，不再是 SIMT style的多个 thread
6. 直接支持卷积指令，支持 weight-stationary GEMM，带上 mask bit 表达哪些是 padding 的0

Revision #8
Created 2025-01-11 09:46:28 UTC by Colin
Updated 2026-04-29 07:33:45 UTC by Colin