架构说明与分工明细

马晋

2025 年 6 月 21 日

目录

1 架构说明

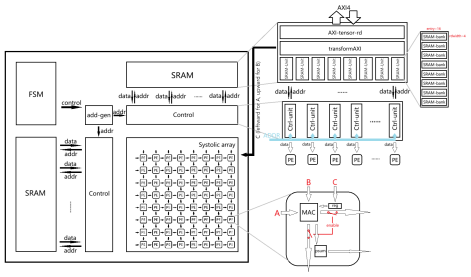

图 1: A/B/C 数据流向示意

1.1 整体架构

- transformAXI: 将 AXI 传递的信号进行转换: 针对 A,B,C 不同的类型进行不同的转换
- A 传入了 SRAMA,B 传入了 SRAMB,C 传入了 systolic(但是为了不造成布线密集,C 一次只能向一个 systolic 传入 32bit)
- addrgen.sv: 生成取数的地址, 在 SRAMA，SRAMB 中取数
- PE.sv 单个的处理单元,systolic.sv:systolic array

1.2 创新点

- 每个 PE 可以支持 1 个 FP16/FP32 的乘累加，以及 4 个 INT8 或 8 个 INT4 的乘加 (完全复用了 32bit 的带宽)
- 浮点乘法采用 pipeline 形式,INT4,INT8 复用了加法器,FP16/FP32 复用了 MAC
- 采用了分块乘法的方式, 在最后阶段进行加法, 提高并行性，也就是实现了 group 乘

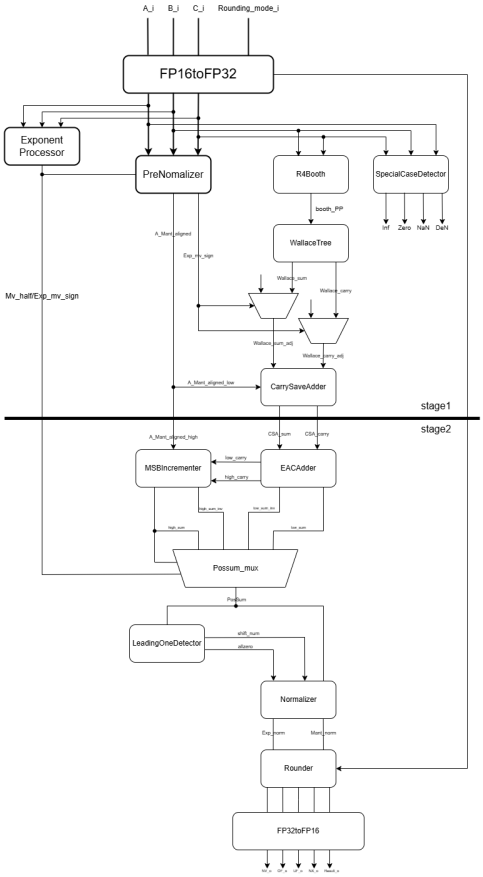

图 2: 3 种形状的运算

- 新增 addrgen 与 control 模块, 将复杂的逻辑与 PE 解耦合, 使得 PE 仅需提供简单的功能
- 将所有的 enable 使能信号也完全变成 systolic 形式, 从而不必关注 PE 内部各种判断与 stall,PE 完全接受外部使能信号

1.3 计算方法

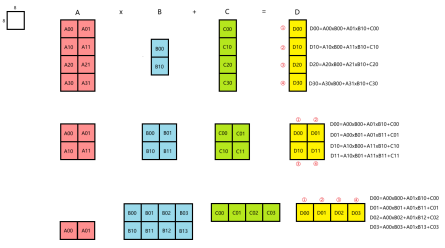

图 3: 3 种形状的运算

如图所示, 这里的 A,B,C 中的每一块都是 8*8 的矩阵, 从而对于 D 而言, 一共四块, 最后都存放在 PE 中的 regfile 上, 第一块的 (0,0) 存在 regfile 的 31 0bit, 第二块的 (0,0)

存在 PE 的 63 32bit, 依次类推

对于 FP32,FP16 而言, 需要考虑的是两级流水, 所以两次计算之间不能产生依赖, 这些已经由 addrgen 完成取数的无关性对于 INT8, 由于能一次算 8 个块乘法, 因此需要在最终加入一个累加阶段, 将 regfile 中的数进行累加, 这里巧妙地运用了 INT4 中最多每个结果只需要 16bit, 从而在 regfile 只有 128bit 的情况下, 复用了资源同时存 8 个 INT4 乘累加的结果

1.4 信号说明

para_pkg.sv:

data_type: 0:FP32,1:FP16,2:INT4,3:INT8

compute_shape: 计算的是哪种类型的乘法:0:M32K16N8,1:M16K16N16,2:M8K16N32

AXI_in_t:AXI 需要传入 tensorecore 的信号, 作为一个接口，我要构建 tensorcore 只关心这些信号

AXI_out_t:tensorcore 需要传出到 AXI 的信号, 作为一个接口，我要构建 tensorcore 只关心这些信号

其它重要信号在代码中有注释

transformAXI.sv:

传入了 256bit(rdbus 的带宽) 的信号, 现在需要将这些信号传递到 SRAM(A,B) 和 systolic(C) 中

addrgen.sv:

在从 systolic array 中接收到 en 后, 开始进行计算, 维护一个计数器 counter, 从而能够实现有规律的取数逻辑

control.sv:

在从内存中取出数字后, 按照类型进行一定的重排/复制

sram.sv:

这里实现了两个 SRAM(但我比较担心后面可能会被综合成寄存器)

SRAM_A: 由 8 个小 SRAM 组成, 每个小 SRAM 由 8 个 bank 组成 (每个 bank4bit，刚好最低支持 INT4), 每个小 SRAM 作为对接 PE 的端口, 其中写入逻辑用 FIFO 实现 (不对外提供写地址)，对于一个 bank: 要么写入 32bit, 要么不写

SRAM_B: 由 8 个小 SRAM 组成, 每个小 SRAM 由 8 个 bank 组成 (每个 bank4bit，刚好最低支持 INT4)，每个小 SRAM 作为对接 PE 的端口，写入用 FIFO, 对于一个 bank: 要么写入 4bit, 要么不写

PE.sv:

作为一个 PE，需要维护 register_pointer, 指向需要写入的寄存器

ENABLE.sv:

实现了信号的 systolic 传输

1.5 项目不足

- 目前我认为瓶颈主要在于取数带宽上, 对于我的 systolic 与调度能提供高性能计算, 在有多个 SRAM 取数从而隐藏传输延迟的情况下效果更佳
- 由于我限制了 systolic 中的每个 PE 每次传入 C 只能写入 32bit, 又由于 AXI 取数方式的约束, 导致此时大大浪费了 AXI 的带宽 (但是确实在一些情况下无法做到)
- 目前部分信号宽度太大, 但是感觉综合工具应该能自动优化 (纯粹是一开始没确定位宽)
- SRAM 是存储资源最丰富的地方, 目前来看可能很难综合成 SRAM, 而是会变成寄存器堆

2 如何验证架构

目前马晋已经完成了 tensorcore 本身的部分, 想要验证, 只需 cd tensorcore/rtl/verilator, make run 即可, 马晋配置了 verilator 环境, 在 tensorcore/.vscode/settings.json 中助教老师遇到任何关于 tensorcore 验证的问题都可以直接联系 imagine(ma15807252614) 验证所写的文件在 tensorcore/rtl/verilator/main.cpp 中, 助教老师可以用 main.cpp 的方式验证对于不同 datatype 和 matrix shape 的测试, 只需要移除部分注释即可, 通过 make run > out.txt, 在 out.txt 中能看见输入矩阵, 以及按照 C++ 计算得到的输出结果以及按照 tensorcore 给出的输出结果。

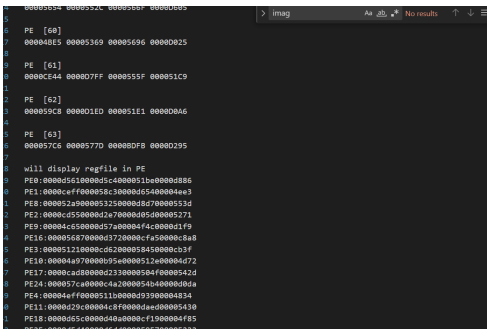

图 4: 输出结果

如图所示,will display 前面的是用软件计算的结果, will display 的是 tensorcore 输出的结果。注意其中的 mixed 表示是否用混合精度训练, 混合精度下的 compute_type 与正常 FP16 相同, 就是 mixed 信号置为 1, 注意同时更改 fmacase 中的类型, 混合精度为 <half,half,float,float>, 上面这四个分别对应于 A,B,C,D 的类型, 如果是 FP16, 则为 <half,half,half,half>, 如果是 INT4, 则为 <int4_t,int4_t,int4_t,int>

3 综合结果

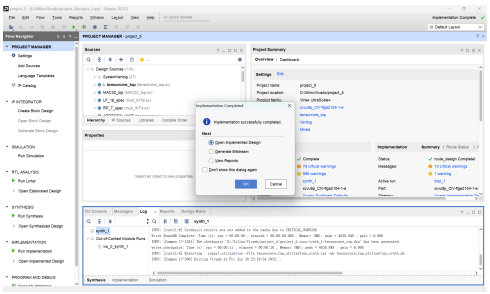

图 5: 布线

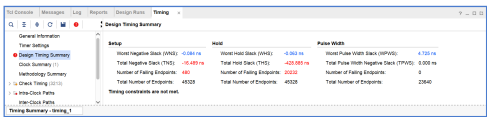

图 6: 综合 100M

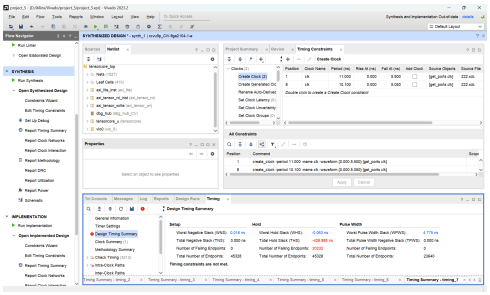

图 7: 综合 99M

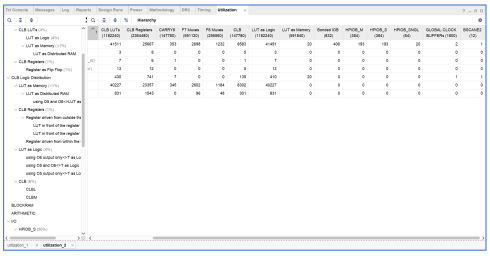

图 8: 资源

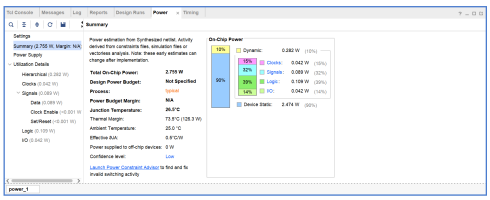

图 9: 功耗

如图所示, 经过测试, 瓶颈在于 PE 只能达到 100M(关键路径上), 但是有一件感觉在综合中感觉逆天的事情是: 我尝试综合了 PE 中的 MAC_top.sv(为了结果准确, 在现有的 MAC_top.sv 输出加了一级寄存器), 发现能达到远不止 100M 的频率, 但是 PE.sv 不知为何无法达到相应的频率,PE.sv 中有很长的 if-else, 但我觉得这个路径相对于 MAC_top.sv 中的乘加部分造成的延时应该微不足道。总而言之, 逆天 PE 触发的逆天机制是约束频率的罪魁祸首。

4 分工

分工主要是为了方便助教能了解每位同学负责什么, 因此在查阅相应的部分没有完成或完善有问题的时候, 可以去联系相应的同学

4.1 马晋

- 完成了基本的 MAC 模块, 并进行了二级流水 (参考了多种版本 FP 的比较: 排除了 FPnew, 因为 FPnew 冗余过多及实现简单,pipeline 方式低效)
- MAC 模块 FP32 的 pipeline+MAC 模块中的加法部分 (同时支持 4bit 和 8bit)+FP16toFP32.sv
- transformtoAXI.sv+addrgen.sv+systolic.sv+control.sv+ENABLE.sv+tensorcore.sv, 从数据传入到数据写回
- 对整体的 tensorcore 进行了测试以及 debug, 包括找到部分 MAC 的错误
- 完成了 tensorcore 中 burst_num 和 burst_size 以及 systolic_time 和 waitwrite_time 的计算并加入
- 全局统筹项目 (前期讨论, 后期解答王俊涵综合的问题)

4.2 李亚飞

- 完成了 tensorcore 中 burst_num 和 burst_size 以及 systolic_time 和 waitwrite_time 的计算并加入
- 完成了各种 axi 的逻辑
- 完成了 FP16toFP32 模块 +FP32toFP16.sv 从而组合出 MAC_FP
- 组装了 MAC_FP 和 MAC_adder 从而完成顶层 MAC, 同时支持多精度 FP,INT
- 对整个 MAC 模块进行了测试以及 debug

4.3 王俊涵

- 完成了 mult_INT4(自写 testbench),mult_INT8(自写 testbench),CLA(自写 testbench),MAC(testbench 由马晋提供) 的验证
- 完成了 MAC 的综合 + 最终 top 的综合

4.4 补充

- 马晋和李亚飞参与了初步的架构构建, 在这个过程中马晋提出了采用分块乘法并在最后阶段做加法的想法, 两人确定了其中部分参数
- 马晋自己完成了最终的架构构建: 考虑了数据排布的影响, 否决了前面提出的矩阵乘法的顺序
- 在最终的架构构建阶段, 考虑将 systolic 与 PE 的复杂性解耦合, 用 addrgen 专门生成数据取地址, 大大提高了效率
- 马晋可能会考虑继续对于架构进行优化, 目前在 INT4 与 INT8 的情况下, 算力 » 带宽, 但是很容易将架构进行扩展, 利用多个 SRAM 使得算力得到充分运用

4.5 贡献度: 仅供参考, 并不准确, 详情可看 github 仓库

- 马晋: 72%
- 李亚飞: 23%
- 王俊涵: 5%