中国科学院大学
《计算机组成原理(研讨课)》实验报告

姓名 朱夏楠 学号 2022K8009929031 专业 计算机科学与技术

实验项目编号 5.3 实验名称 深度学习算法与硬件加速器

注 1:	撰写此 Word 格式实验报告后以 PDF 格式保存 SERVE CloudIDE 的 `/home/serve-ide/cod-lab/reports` 目录下 (注意: reports 全部小写)。文件命名规则: `prjN.pdf`, 其中 `prj` 和后缀名 `pdf` 为小写, `N` 为 1 至 4 的阿拉伯数字。例如: `prj1.pdf`。PDF 文件大小应控制在 5MB 以内。此外, 实验项目 5 包含多个选做内容, 每个选做实验应提交各自的实验报告文件, 文件命名规则: `prj5-projectname.pdf`, 其中 "-" 为英文标点符号的短横线。文件命名举例: `prj5-dma.pdf`。具体要求详见实验项目 5 讲义。
注 2:	使用 `git add` 及 `git commit` 命令将实验报告 `PDF` 文件添加到本地仓库 master 分支, 并通过 `git push` 推送到实验课 SERVE GitLab 远程仓库 master 分支(具体命令详见实验报告)。
注 3:	实验报告模板下列条目仅供参考, 可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明(比如 Verilog HDL 关键代码段 {包含注释} 及其对应的逻辑电路结构图 {自行画图, 推荐用 PPT 画逻辑结构框图后保存为 PDF, 再插入到 LaTeX. 中}、相应信号的仿真波形和信号变化的说明等)

- **乘法指令通路实现。**

本实验的乘法指令是在已经实现的 RISCV-32 流水线处理器的基础上进行拓展得到的, 在实现时没有用 verilog 语言搭建乘法器, 而是选择使用了 verilog 语法提供的乘法, 观察 RV32M 标准拓展中 MUL 指令的操作码, 发现 opcode 段为 0110011, 可以将其归入 R-Type 类型的指令, 再考虑到实现 RISCV32 处理器时, ALU 部件实现的 ALU 运算有 7 种, 对应的 ALUop 有三位, 因此选择直接在 ALU 中添加一个乘法操作, 可以利用 ALUop 剩余的一个位置, 并且数据通路的实现也较为方便。

```
//其余部分的ALU代码不再展示
//计算出A乘B的结果
assign result_mul = A * B;
//根据ALUop选择输出的运算结果
assign isand = &ALUop;
assign isor = ~ALUop[0] & ALUop[1] & ALUop[2];
assign isxor = ~ALUop[0] & ~ALUop[1] & ALUop[2];
assign iscomp = ALUop[1] & ~ALUop[2];
assign issum = ~ALUop[1] & ~ALUop[2];
assign ismul = ALUop[0] & ~ALUop[1] & ALUop[2]; //乘法对应的ALUop为3'b101

assign Result = ({32{isand}} & result_and)
    | ({32{isor}} & result_or)
    | ({32{isxor}} & result_xor)
    | ({32{issum}} & sum)
    | ({32{iscomp}} & result_comp)
        | ({32{ismul}} & result_mul[31:0]);
```

• **2D 卷积池化算法设计**。

算法的实现分为三个部分，分别为卷积算法、池化算法、以及硬件加速器控制程序。卷积算法可根据讲义中的描述实现，设计的算法如下，最内层的两个循环实现的是一次卷积操作。在卷积过程中，使用整型变量保存中间结果，以防止出现精度的损失，将计算出的结果截断为十六位。根据实验中定点数的规格可知，16 位中低 10 位是小数，最高位为符号位，中间为整数位，乘法计算出的结果中，低 20 位为小数部分，因此截断时将整型变量算数右移 10 位，再将结果强制类型转换为 short 类型，存入内存中对应的位置。

```
    for(no=0;no<wr_size.d1;no++)
    {
        for(ni=0;ni<rd_size.d1;ni++)
        {
            for(y=0;y<conv_out_h;y++)
            {
                for(x=0;x<conv_out_w;x++)
                {
                    temp = 0;
                    if(ni==0)
                        (*out_array)[no][y][x] = (*weight_array)[no][0][0];
                    for(ky=0;ky<weight_size.d2;ky++)
                    {
                        for(kx=0;kx<weight_size.d3;kx++)
                        {
                            iw = kx + mul(x,stride) - pad;
                            ih = ky + mul(y,stride) - pad;
                            if(iw>=0 && iw<input_fm_w && ih>=0 && ih<input_fm_h)
                                temp +=
                                    mul((*in_array)[ni][ih][iw],(*weight_array)[no][ni][mu
                                    WEIGHT_SIZE_D3) + kx + 1]);
                        }
                    }
                    (*out_array)[no][y][x] += (short)(temp >> FRAC_BIT);
                }
            }
        }
    }
```

考虑到对输入图进行边界填充，或者是步幅不为 1 的情况，需要在运算时对 iw 和 ih 进行判断，iw 和 ih 在计算时也要考虑填充的长度，若计算出的 iw 或 ih 小于 0 或者大于未填充时的输入图宽度或高度，则说明此处超出了输入图的位置或者是处于填充值为 0 的位置，不进行计算。

池化算法的实现如下，内层的两个循环负责计算出一个 2×2 的矩阵中最大的值，与卷积算法同样，需考虑边界填充的情况，算法实现方式也类似。

```
if ((!pad) && (pad_w_test_remain || pad_h_test_remain)) //考虑池化时是否填充
{
   pool_out_w++;
   pool_out_h++;
}
    //经过上述操作，得出来的输出结果的高和宽时正确的
    //此高度和宽度说明奇数矩阵需要填充边缘值
```

```
for(no=0;no<wr_size.d1;no++)
{
        for(y=0;y<pool_out_h;y++)
        {
                for(x=0;x<pool_out_w;x++)
                {
                        max = 0x8000;                          //max赋值为short类型的最小值
                        for(ky=0;ky<KERN_ATTR_POOL_KERN_SIZE;ky++)
                        {
                                for(kx=0;kx<KERN_ATTR_POOL_KERN_SIZE;kx++)
                                {
                                        iw = mul(x,KERN_ATTR_POOL_KERN_SIZE) - pad + kx;
                                        ih = mul(y,KERN_ATTR_POOL_KERN_SIZE) - pad + ky;
                                        if(iw>=0 && iw<input_fm_w && ih>=0 && ih<input_fm_h)
                                        {
                                                if(max < (*in_array)[no][ih][iw])
                                                        max = (*in_array)[no][ih][iw];
                                        }
                                }
                        }
                        (*out_array)[no][y][x] = max;
                }
        }
}
```

硬件加速器的控制程序可参照讲义中给出的流程图实现，通过改变 START 寄存器中的值，可以控制加速器的启动，并根据 DNOE 寄存器 0 位的值来判断硬件加速是否已经完成卷积运算。

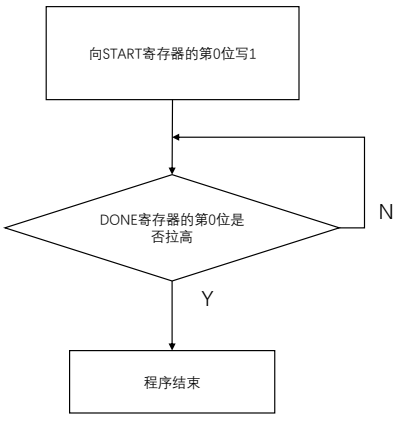

图 1: 硬件加速器的控制流程

二、 思考与总结

- **算法的优化**。

在卷积和池化算法中，我采用了最符合直觉的算法设计，在最内层的循环中算出了 iw 和 ih 的值，但是这种设计的弊端是进行了太多的乘法，在后续的优化中，可以考虑将 iw 和 ih 的计算变为加法，随着内层的两个循环不断累加，这样可以减少编译器生成的乘法指令数，减少程序运行时间。

- **性能评估**。

FPGA 仿真提供了三种执行卷积池化算法的方法，分别是不使用乘法指令进行卷积、使用处理器提供的乘法指令进行卷积、使用外部的硬件加速器进行卷积，根据在程序中添加的性能计数器，可以统计不同情况下程序执行的总时钟周期数。

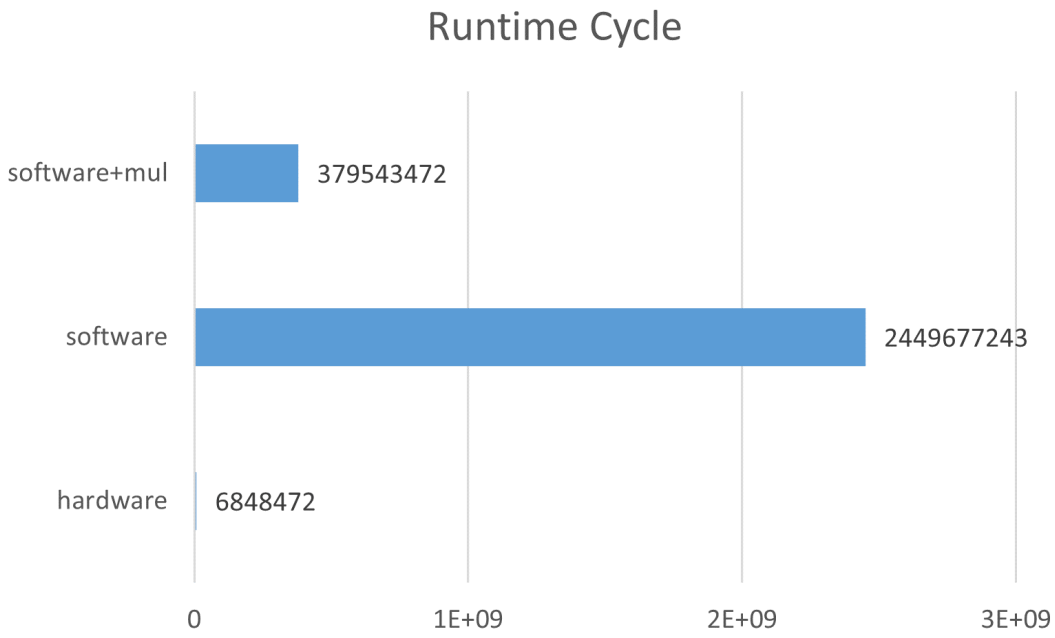

图 2: 性能对比

可以看到在实现了乘法指令之后，在不使用硬件加速器的时候，处理器进行卷积算法的速度有了一个较大的提升，但由于我未在处理器中实现一个真正的乘法器，也没有查看乘法综合出来的电路图是什么样的，我对于这两者为何会相差这么多没有一个足够清晰的认知。而硬件加速器由于是在处理器外部，所以处理器实际上是没有参与卷积的计算的，而测试程序最花时间的部分正是卷积算法，因此使用了加速器之后，程序运行速度有了一个显著的提升。

三、 实验所耗时间

在课后，你花费了大约____8____ 小时完成此次实验。