中国科学院大学计算机组成原理（研讨课）

实 验 报 告

学号：2021K8009929034 姓名：侯汝垚 专业：计算机科学与技术

实验序号：5.1 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 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

1. 新增乘法数据通路

本实验采用了比较简陋的乘法实现，直接通过“*”让综合器自动生成乘法电路使其在一个周期就获得乘法结果。

```
wire [31:0] mul_result = RF_rdata1 * RF_rdata2;
```

根据乘法指令的特性，即 MUL 为 R-type 且 func7 字段第一位为 1，用一个 MUL 信号选择 ALUOut 寄存器的数据来源。

```
wire MUL    = R_type && func7[0];

always @(posedge clk) begin
        if (current_state[5] || current_state[6] || current_state[7])
                ALUOut <= ALUOut;
        else
                ALUOut <= {32{ShifterValid &&  current_state[4]}} & Shifter_result
                        | {32{MUL && current_state[4]}} & mul_result
                        | {32{((~ShifterValid && ~MUL) || ~current_state[4])}} & ALU_result;
end
```

2. 卷积和池化的 C 程序代码

为了减少程序执行时间，代码多次采用了强度削弱（乘法变加法）和代码外提的优化方法，尽量少做乘法运算，将循环内重复的相同乘法尽量提到外层循环，例如

weight_no_offset、in_ni_offset 分别表示第 no 个输出图像的权值的第一个元素的偏移量和第 ni 个输入图像的第一个元素的偏移量，将其分别在 no、ni 对应的那层循环中计算。优化后的代码缺点是不易理解。

```
int in_square = mul(input_fm_h, input_fm_w);
int weight_square = 1 + mul(WEIGHT_SIZE_D2, WEIGHT_SIZE_D3);
int weight_no_offset, weight_offset;
int temp;
for (no = 0; no < conv_size.d1; no++) {
        weight_no_offset = mul(no, mul(rd_size.d1, weight_square));
        bias = weight[weight_no_offset];
        for (ni = 0; ni < rd_size.d1; ni++) {
                int in_ni_offset = mul(ni, in_square);
                int weight_ni_offset = weight_no_offset + mul(ni, weight_square) + 1;
                for (y = 0; y < conv_out_h; y++) {
                        stride_y = mul(stride, y);
                        stride_x = 0;
                        for (x = 0; x < conv_out_w; x++) {
                                temp = 0;
                                for (ky = 0; ky < WEIGHT_SIZE_D2; ky++) {
                                        ih = ky + stride_y - pad;
                                        int in_line_offset = mul(ih, input_fm_w);
                                        int weight_line_offset = mul(ky, WEIGHT_SIZE_D3);
                                        for (kx = 0; kx < WEIGHT_SIZE_D3; kx++) {
                                                iw = kx + stride_x - pad;
                                                if (iw >= 0 && ih >= 0 && iw < input_fm_w && ih < input_fm_h) {
                                                        input_offset = in_ni_offset + in_line_offset + iw;
                                                        weight_offset = weight_ni_offset + weight_line_offset + kx;
                                                        temp += mul(in[input_offset], weight[weight_offset]);
                                                }
                                        }
                                }
                                out[output_offset] = (temp >> FRAC_BIT) + bias;
                                output_offset++;
                                stride_x += stride;
                        }
                }
        }
}
```

池化算法采用了相同的优化思路，减少乘法计算的次数。

```
// TODO: Please add your implementation here
int in_square = mul(input_fm_h, input_fm_w);
int y, x, ky, kx, no, iw, ih;
short max, value;
int stride_y, stride_x, in_line_offset, input_no_offset;
for (no = 0; no < conv_size.d1; no++) {
        input_no_offset = mul(no, in_square);
        for (y = 0; y < pool_out_h; y++) {
                stride_y = mul(y, stride);
                stride_x = 0;
                for (x = 0; x < pool_out_w; x++) {
                        max = 1 << 15;
                        for (ky = 0; ky < KERN_ATTR_POOL_KERN_SIZE; ky++) {
                                ih = ky + stride_y - pad;
                                in_line_offset = mul(ih, input_fm_w);
                                for (kx = 0; kx < KERN_ATTR_POOL_KERN_SIZE; kx++) {
                                        iw = kx + stride_x - pad;
                                        if (iw < 0 || ih < 0 || ih >= input_fm_h || iw >= input_fm_w) {
                                                value = 0;
                                        } else {
                                                input_offset = input_no_offset + in_line_offset + iw;
                                                value = out[input_offset];
                                        }
                                        max = (max < value) ? value : max;
                                }
                        }
                        out[output_offset] = max;
                        output_offset++;
                        stride_x += stride;
                }
        }
}
```

3. 硬件加速器驱动程序

该程序较为简单，只需要用掩码 0x1 通过或位运算将 gpio_start 的第一位置为 1 即可启动硬件加速器，之后同样用掩码 0x1 通过与位运算获得 gpio_done 的第一位即可判断是否计算完毕。

```c
#ifdef USE_HW_ACCEL
void launch_hw_accel()
{
        volatile int *gpio_start = (void *)(GPIO_START_ADDR);
        volatile int *gpio_done = (void *)(GPIO_DONE_ADDR);

        // TODO: Please add your implementation here
        *gpio_start |= 0x1;
        while (!(*gpio_done) & 0x1)
                ;
}
#endif
```

4. 性能计数器对比

a. SW

```
--------------------------------------total cycle      : 1841402783
IW cycle          : 496681695
jump times        : 429396
branch times      : 54762969
branchValid times: 51436379
memRead cycle     : 88185303
memWrite cycle    : 2424494
instr num         : 327370290
```

b. SW_MUL

```
--------------------------------------total cycle      : 371760497
IW cycle          : 311951384
jump times        : 16
branch times      : 1300026
branchValid times: 308381
memRead cycle     : 42692812
memWrite cycle    : 58598
instr num         : 4593309
benchmark finished
time 3781.46ms
```

c. HW

```
--------------------------------------total cycle      : 1062243
IW cycle          : 944851
jump times        : 6
branch times      : 5340
branchValid times: 5281
memRead cycle     : 79478
memWrite cycle    : 373
instr num         : 10954
benchmark finished
time 91.21ms
```

如果只考虑周期数，使用乘法指令相比软件加法实现乘法快了约五倍，根据 mul.h 中乘法的实现可知，软件实现乘法方法需要检测乘数的每一位，这样一次乘法需要消耗上百个周期，速度差距明显。在不考虑上述卷积池化代码优化的情况下差距应该会更加明显。

使用硬件加速器比使用乘法指令快了 350 倍，有了质的飞跃，由此可见专用的硬件加速器对于性能提升帮助非常大。

5. 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出

现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

a. 如何实现乘法?

假如采用 booth 乘法算法，一次 n 位乘法乘法需要多次加法和移位才能完成，一些乘法的优化方法例如华莱士树看不太懂，根据微信群中同学提示，为了能够在一个周期内算出结果，采用了比较简单的直接星号乘。但是验收过程中叶从容老师提到如果采用星号乘法，那么会生成一个特别庞大的电路，使得组合逻辑延迟会特别长，对性能造成很大影响。

b. 程序无限打印。

最开始以为是乘法运算耗时太长而超时，多次强度削弱（乘改加）和代码外提后依然出错，并且不知道为什么一直打印“starting convolution”。经吉骏雄同学提醒了发现问题在于 CPU 设计出错，在 ALUOut 的选择逻辑中各条件不互斥，导致跳转地址出错，每次跳转都回到 printf 函数处，因此无线打印字符。

c. 计算结构精度不够

程序用十六位定点数表示小数，运算结果需要通过移位舍去多余的小数位数，因此也会损失精度。对于输出图像的每一个像素，假如每次运算后都直接移位，那么这部分精度会直接损失，因此导致答案出错。为了保留多余的小数位数，采用 int 型变量存储中间值，直到算出最终结果后再进行移位并加上偏移量。，

6. 思考题

a. 如果使用边界填充，算法应如何修改?

考虑边界填充后，首先输入图像的实际尺寸应加上填充部分，即高和宽都加上（pad*2），因此实际输出图像的高和宽也要额外加上(pad*2)。这部分代码已经提前写好。

```
unsigned pad = KERN_ATTR_CONV_PAD;
unsigned pad_len = pad << 1;

unsigned conv_out_w = rd_size.d3 - weight_size.d3 + pad_len;
unsigned conv_out_h = rd_size.d2 - weight_size.d2 + pad_len;
```

b. 如何避免出现溢出和精度损失?

乘法、加法运算的中间结果可使用 32-bit 定点数来表示，保留多余的小数部分，知道计算出一个像素的结果后再移位。

7. 在课后，你花费了大约 10 小时完成此次实验。

8. 对于此次实验的心得、感受和建议

本次实验相较于其他选作实验比较容易，重点在于理解实验中三个宏对应的意思以及卷积和池化算法的 c 语言实现。因为 ppt 已经给出卷积和池化的伪代码，只需依次翻译为 c 语言即可。实验框架中似乎没有对边界有填充的情况进行测试，希望可以改进。