中国科学院大学计算机组成原理（研讨课）

实 验 报 告

学号： 2021K8009925006 姓名：冯浩瀚 专业：计算机科学与技术

实验序号：05 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（比如关键 RTL 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，复制到 word 中}、相应信号的仿真波形和信号变化的说明等）

1. 在基于 RISC-V 的 custom_cpu 中添加 MUL 指令通路：

实验中并没有采用直接修改 ALU 模块的做法，原因是原来 alu.v 的译码器是 3-8 译码器，而原有 alu 已经拥有 8 种操作（如图），故再加入 MUL 操作需要改动的地方过多，因此实验中采用了另外加入一个 MUL 的指令类型（type），与 Ralu 类型指令并列的做法。

```
`define ALUOP_AND  3'b000
`define ALUOP_OR   3'b001
`define ALUOP_XOR  3'b100
`define ALUOP_NOR  3'b101
`define ALUOP_ADD  3'b010
`define ALUOP_SUB  3'b110
`define ALUOP_SLT  3'b111
`define ALUOP_SLTU 3'b011 // define ALU operation code
```

```
wire [3:0] type = {4{opcode == 7'b1100011}} & `BType|
       {4{opcode == 7'b1100111}} & `IType |    // jalr
       {4{opcode == 7'b0010011 && {funct3[1:0]} != 2'b01}} & `IAlu |
       {4{opcode == 7'b0000011}} & `ILoad |
       {4{opcode == 7'b0010011 && {funct3[1:0]} == 2'b01}} & `IShift |
       {4{opcode == 7'b1101111}} & `JType |    // jal
       {4{opcode == 7'b0110011 && {funct3[1:0]} != 2'b01 && funct7 != 7'b0000001}} & `RAlu |
       {4{opcode == 7'b0110011 && funct3 == 3'b000 && funct7 == 7'b0000001}} & `MUL |
       {4{opcode == 7'b0110011 && {funct3[1:0]} == 2'b01}} & `RShift |
       {4{opcode == 7'b0100011}} & `SType |
       {4{{opcode[6],opcode[4:0]} == 6'b010111}} & `UType; // lui, auipc
// decode the opcode
```

```
// mul
wire [63:0] MUL_Res_full;
wire [31:0] MUL_result;
assign MUL_Res_full = RF_rdata1 * RF_rdata2;
assign MUL_result = (funct3 == 3'b000)? MUL_Res_full[31:0]: MUL_Res_full[63:32];
```

```
assign RF_wdata = (type == `RAlu || type == `IAlu)? ALU_result:
       (type == `MUL)? MUL_result:
       (type == `RShift || type == `IShift)? SHIFT_result:
       (type == `UType && opcode[5])? U_imm:
       (type == `UType && ~opcode[5])? PC_auipc:
       (type == `JType || type == `IType)? PC_pre + 4:
       (type == `ILoad)? load_result:32'b0; // write data
```

2. 修改测试软件代码

（1） 卷积算法

用多层嵌套循环的方式，将二维的卷积分成若干次的一维卷积，并设置累加数组（conv_res），将结果累加：

```
//TODO: Please add your implementation here
typedef short IN_TYPE[rd_size.d1][rd_size.d2][rd_size.d3];
IN_TYPE *IN = (IN_TYPE *) (in + input_offset);
typedef short WEIGHT_TYPE[weight_size.d0][weight_size.d1][mul(weight_size.d2, weight_size.d3) + 1];
WEIGHT_TYPE *WEIGHT = (WEIGHT_TYPE *) weight;
typedef short OUT_TYPE[conv_size.d1][conv_size.d2][conv_size.d3];
OUT_TYPE *OUT = (OUT_TYPE *) (out + output_offset);

for(int no = 0; no < wr_size.d1; no++)
{
        for (int ni = 0; ni < rd_size.d1; ni++)
        {
                for(int y = 0; y < conv_size.d2; y++)
                {
                        int ybase = mul(y, stride) - pad;
                        for(int x = 0; x < conv_size.d3; x++)
                        {
                                int xbase = mul(x, stride) - pad;
                                if(ni == 0)
                                {
                                        (*OUT)[no][y][x] = (*WEIGHT)[no][0][0];
                                }
                                int conv_res = 0;
                                for (int ky = 0; ky < weight_size.d2; ky++)
                                {
                                        int yoffset = mul(ky, weight_size.d3);
                                        int ih = ky + ybase;
                                        for (int kx = 0; kx < weight_size.d3; kx++)
                                        {
                                                int iw = kx + xbase;
                                                if (iw >= 0 && iw < input_fm_w && ih >= 0 && ih < input_fm_h)
                                                {
                                                        conv_res += mul((*IN)[ni][ih][iw], (*WEIGHT)[no][ni][yoffset + kx + 1]);
                                                }
                                        }
                                }
                                (*OUT)[no][y][x] += conv_res >> FRAC_BIT;
                        }
                }
        }
}
```

（2） 池化算法

池化算法与卷积算法思路大致相同，只是将关键的累加步骤替换为最大值的更新，因此还需要提前定义一个用于取最大值的 MAX()函数：

```
short MAX(short a, short b) {
    return a > b ? a : b;
}
```

```
//TODO: Please add your implementation here
short *in = (short *)addr.wr_addr;

typedef short IN_TYPE[conv_size.d1][conv_size.d2][conv_size.d3];
IN_TYPE *IN = (IN_TYPE *) (in + input_offset);
typedef short OUT_TYPE[wr_size.d1][pool_out_h][pool_out_w];
OUT_TYPE *OUT = (OUT_TYPE *) (out + output_offset);

for (int no = 0; no < wr_size.d1; no++)
{
        for (int y = 0; y < pool_out_h; y++)
        {
                int ybase = mul(y, stride) - pad;
                for (int x = 0; x < pool_out_w; x++)
                {
                        int xbase = mul(x, stride) - pad;
                        short max = 0x8000;
                        for (int ky =0 ; ky < KERN_ATTR_POOL_KERN_SIZE; ky++)
                        {
                                int ih = ky + ybase;
                                for (int kx = 0; kx < KERN_ATTR_POOL_KERN_SIZE; kx++)
                                {
                                        int iw = kx + xbase;
                                        if (iw >= 0 && iw < input_fm_w && ih >= 0 && ih < input_fm_h)
                                        {
                                                max = MAX(max, (*IN)[no][ih][iw]);
                                        }
                                }
                        }
                        (*OUT)[no][y][x] = max;
                }
        }
}
```

二、 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）

1. TIME OUT!!!

初期的代码存在很多重复计算的问题，将只需要计算一次的操作嵌套进了多次循环中，因此每次上板测试都出现超时。

另外，还有一种解决方法是将三维数组的运算用指针与地址取代，化三维为一维，大大缩短时钟周期数。

2. 关于性能计数器

给出的代码模板没有引用 perf_cnt.h 头文件，需手动添加；打印性能计数操作需要在 hit_good_trap()之前完成，否则将出现错误。

```
#endif
        bench_done(&res);
        printf("total cycle: %d\n", res.msec);
        int result = comparing();
        printf("benchmark finished\n");
        if (result == 0) {
                hit_good_trap();
        } else {
                nemu_assert(0);
        }

        return 0;
```

三、 对讲义中思考题（如有）的理解和回答

1. 请思考如果使用边界填充，算法应如何修改。

答：在对每个通道卷积或池化时，均考虑了 pad 和 stride 的影响，只需要根据需要在代码最前边的宏定义中修改二者的值。

```
for(int y = 0; y < conv_size.d2; y++)
{
        int ybase = mul(y, stride) - pad;
        for(int x = 0; x < conv_size.d3; x++)
        {
                int xbase = mul(x, stride) - pad;
```

2. 在软件算法实现中，需考虑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，请同学们思考如何扩展?

答：在对 conv_res 累加的时候，不考虑移位，在所有结果累加后跳出循环，再进行移位。

3. 分析不同实现方法的性能差异

答：hw_conv：

```
RUNNER_CNT = 1
Completed FPGA configuration
Launching hw_conv benchmark...
tggetattr: Inappropriate ioctl for device
reset: before MMIO access...
reset: MMIO accessed
axi_firewall_unblock: firewall error status: 00000000
main: before DDR accessing...
main: DDR accessed...
reset: before MMIO access...
reset: MMIO accessed
Launching task...
total cycle: 1018318
Passed!
benchmark finished
time 71.96ms
reset: before MMIO access...
reset: MMIO accessed
./software/workload/ucas-cod/benchmark/simple_test/dnn_test/riscv32/elf/hw_conv passed
Hit good trap
pass 1 / 1
```

sw_conv：

```
RUNNER_CNT = 0
Completed FPGA configuration
Launching sw_conv benchmark...
tggetattr: Inappropriate ioctl for device
reset: before MMIO access...
reset: MMIO accessed
axi_firewall_unblock: firewall error status: 00000000
main: before DDR accessing...
main: DDR accessed...
reset: before MMIO access...
reset: MMIO accessed
starting convolution
starting pooling
total cycle: -1353221291
Passed!
benchmark finished
time 201259.65ms
reset: before MMIO access...
reset: MMIO accessed
./software/workload/ucas-cod/benchmark/simple_test/dnn_test/riscv32/elf/sw_conv passed
Hit good trap
pass 1 / 1
```

sw_conv_mul:

```
RUNNER_CNT = 0
Completed FPGA configuration
Launching sw_conv_mul benchmark...
tggetattr: Inappropriate ioctl for device
reset: before MMIO access...
reset: MMIO accessed
axi_firewall_unblock: firewall error status: 00000000
main: before DDR accessing...
main: DDR accessed...
reset: before MMIO access...
reset: MMIO accessed
starting convolution
starting pooling
total cycle: 413496526
Passed!
benchmark finished
time 4176.88ms
reset: before MMIO access...
reset: MMIO accessed
./software/workload/ucas-cod/benchmark/simple_test/dnn_test/riscv32/elf/sw_conv_mul passed
Hit good trap
pass 1 / 1
```

可以看到，不使用 mul 指令实现乘法的软件方法的时钟周期数发生了溢出，说明其时钟周期数远大于另外两个方法。而使用 mul 指令乘法的软件方法所需的时钟周期数是硬件方法的 406 倍，由此可见利用硬件加速器可以较好提高深度学习算法的效率。

四、 在课后，你花费了大约 10 小时完成此次实验。

五、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

本次实验资料讲义讲解较为详尽，因此总体难度不大。只是由于平台提供的调试工具有限，一旦代码出现 bug 需要花费较长的时间进行调试。通过本次实验我了解了深度学习的基础算法，并且对硬件加速器有了更深的理解。