中国科学院大学计算机组成原理实验课

实 验 报 告

学号：_2016K8009929018_　姓名：_陈灿宇_　专业：_计算机科学与技术_

实验序号:____prj5-3____　实验名称:_____深度学习算法及硬件加速_____

注 1：本实验报告请以 PDF 格式提交。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 5 的阿拉伯数字。例如：2015K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。
注 2：实验报告模板以下部分的内容供参考，可包含但不限定如下条目内容。

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、相应信号的仿真波形和信号变化的说明等）

本次实验我完成了深度学习核心算法（2D 卷积+池化）的软件实现，并为 CPU 增加了乘法指令和更多性能计数器，同时完成了加速器控制软件的编写。

首先，在原来的 CPU 的基础之上，添加了更多性能计数器，核心代码如下：

```
reg     [31:0]  cycle_cnt;
reg     [31:0]  read_mem_cnt;
reg     [31:0]  write_mem_cnt;
wire    [31:0]  mem_cnt;
reg     [31:0]  IF_cnt;
reg     [31:0]  IW_cnt;
reg     [31:0]  ID_EX_cnt;
reg     [31:0]  RDW_cnt;
reg     [31:0]  write_reg_file_cnt;
reg     [31:0]  Load_cnt;
reg     [31:0]  Store_cnt;
reg     [31:0]  MUL_cnt;
reg     [31:0]  R_type_cnt;
wire    [31:0]  wait_cnt;

always @(posedge clk) begin
    if (rst) begin
        cycle_cnt <= 32'd0;
        read_mem_cnt <= 32'd0;
        write_mem_cnt <= 32'd0;
        IF_cnt <= 32'd0;
        IW_cnt <= 32'd0;
        ID_EX_cnt <= 32'd0;
        RDW_cnt <= 32'd0;
        write_reg_file_cnt <= 32'd0;
        Load_cnt <= 32'd0;
        Store_cnt <= 32'd0;
        MUL_cnt <= 32'd0;
        R_type_cnt <= 32'd0;
    end
    else begin
        cycle_cnt <= cycle_cnt + 32'd1;
        if (state_next == IF) begin
            IF_cnt <= IF_cnt + 32'd1;
        end
        if (state_next == IW) begin
            IW_cnt <= IW_cnt + 32'd1;
        end
        if (state_next == ID_EX) begin
            ID_EX_cnt <= ID_EX_cnt + 32'd1;
            if ({Instruction[31:26]} == SPECIAL) begin
                R_type_cnt <= R_type_cnt + 32'd1;
            end
            if ({Instruction[31:26],Instruction[5:0]} == {SPECIAL2,MUL_FUNC}) begin
                MUL_cnt <= MUL_cnt + 32'd1;
            end
            if(Instruction[31:26] == LBU
```

```
                    || Instruction[31:26] == LB
                    || Instruction[31:26] == LW
                    || Instruction[31:26] == LWL
                    || Instruction[31:26] == LWR
                    || Instruction[31:26] == LH
                    || Instruction[31:26] == LHU) begin
                        Load_cnt <= Load_cnt + 32'd1;
                    end
                    if (Instruction[31:26] == SW
                    || Instruction[31:26] == SH
                    || Instruction[31:26] == SWL
                    || Instruction[31:26] == SWR
                    || Instruction[31:26] == SB) begin
                        Store_cnt <= Store_cnt + 32'd1;
                    end
                    if ((Instruction[31:26] == JAL)
                      ||(Instruction[31:26] == SPECIAL && Instruction[5:0] == JALR_FUNC)
                      ) begin
                        write_reg_file_cnt <= write_reg_file_cnt + 32'd1;
                    end
                end
                if (state_next == ST) begin
                    write_mem_cnt <= write_mem_cnt + 32'd1;
                end
                if (state_next == LD) begin
                    read_mem_cnt <= read_mem_cnt + 32'd1;
                end
                if (state_next == RDW) begin
                    RDW_cnt <= RDW_cnt + 32'd1;
                end
                if (state_next == WB) begin
                    write_reg_file_cnt <= write_reg_file_cnt + 32'd1;
                end
            end
        end

        assign mem_cnt = IF_cnt + write_mem_cnt + read_mem_cnt + IW_cnt + RDW_cnt;
        assign wait_cnt = IW_cnt + RDW_cnt;

        assign mips_perf_cnt_0 = cycle_cnt;
        assign mips_perf_cnt_1 = read_mem_cnt;
        assign mips_perf_cnt_2 = write_mem_cnt;
        assign mips_perf_cnt_3 = mem_cnt;
        assign mips_perf_cnt_4 = IF_cnt;
        assign mips_perf_cnt_5 = IW_cnt;
        assign mips_perf_cnt_6 = ID_EX_cnt;
        assign mips_perf_cnt_7 = RDW_cnt;
        assign mips_perf_cnt_8 = write_reg_file_cnt;

        assign mips_perf_cnt_9 = Load_cnt;
        assign mips_perf_cnt_10 = Store_cnt;
        assign mips_perf_cnt_11 = MUL_cnt;
        assign mips_perf_cnt_12 = R_type_cnt;
        assign mips_perf_cnt_13 = wait_cnt;
        assign mips_perf_cnt_14 = 32'd0;
        assign mips_perf_cnt_15 = 32'd0;
```

convolution()函数的核心代码如下：

```
unsigned no, ni, x, y, kx, ky;
unsigned weight_offset = 0;
unsigned weight_offset_store = 0;
unsigned product_x_stride = 0;
unsigned product_y_stride = 0;
unsigned no_offset_1 = 0;
unsigned no_offset_2 = 0;
unsigned ni_offset_1 = 0;
unsigned ni_offset_2 = 0;
unsigned y_offset = 0;
unsigned ky_offset_1 = 0;
unsigned ky_offset_2 = 0;
unsigned conv_size_area = mul(conv_size.d2,conv_size.d3);
unsigned weight_size_area = mul(weight_size.d2,weight_size.d3);
unsigned input_area = mul(input_fm_h,input_fm_w);
int out_store;

conv_out_w = div(conv_out_w, stride);
conv_out_h = div(conv_out_h, stride);

conv_out_w++;
conv_out_h++;

conv_size.d0 = wr_size.d0;
conv_size.d1 = wr_size.d1;
conv_size.d2 = conv_out_h;
conv_size.d3 = conv_out_w;
conv_size_area = mul(conv_size.d2,conv_size.d3);

for (no = 0; no < wr_size.d1; ++no)
{
    no_offset_1 = mul(no,conv_size.d0);
    no_offset_2 = mul(no, conv_size_area);
    weight_offset_store = mul(no_offset_1,(1+mul(weight_size.d2,weight_size.d3)));

    for (y = 0; y < conv_size.d2; ++y)
    {
        product_y_stride = mul(stride,y);
        y_offset = mul(y,conv_size.d3);
        for (x = 0; x < conv_size.d3; ++x)
        {
            product_x_stride = mul(stride,x);
            out_store = 0;
            output_offset = no_offset_2 + y_offset + x;
```

```
            for (ni = 0; ni < conv_size.d0; ++ni)
            {
                ni_offset_1 = mul(ni,(1+weight_size_area));
                ni_offset_2 = mul(ni,input_area);

                for (ky = 0; ky < weight_size.d2; ++ky)
                {
                    ky_offset_1 = mul(weight_size.d3,ky);
                    ky_offset_2 = mul(input_fm_w,(product_y_stride + ky - pad));
                    for (kx = 0; kx < weight_size.d3; ++kx)
                    {
                        weight_offset = weight_offset_store + ni_offset_1 + 1 + ky_offset_1 + kx;

                        input_offset = ni_offset_2 + ky_offset_2 + product_x_stride + kx - pad;

                        if (((product_x_stride + kx) >= pad) && ((product_x_stride + kx) < (pad+input_fm_w))
                         && ((product_y_stride + ky) >= pad) && ((product_y_stride + ky) < (pad+input_fm_h)))
                        {
                            out_store += mul((short)(*(in+input_offset)), (short)(*(weight+weight_offset)));
                        }

                    }
                }
            }
            out_store = (((short)(out_store >> FRAC_BIT) & 0x7fff) | ((short)(out_store >> 16) & 0x8000)) + (*(weight+weight_offset_store));
            (*(out+output_offset)) = (short)out_store;
        }
    }
}
```

pooling（）函数的核心代码如下：

```
    unsigned no,x,y,px,py;
    unsigned input_area = mul(input_fm_w,input_fm_h);
    unsigned pool_out_area = 0;
    unsigned y_offset = 0;
    unsigned no_offset_1 = 0;
    unsigned no_offset_2 = 0;
    unsigned product_y_stride = 0;
    unsigned product_x_stride = 0;
    unsigned product_2 = 0;


    pool_out_w = div(pool_out_w, stride);
    pool_out_h = div(pool_out_h, stride);
    pool_out_w++;
    pool_out_h++;

    if ( (!pad) && (pad_w_test_remain || pad_h_test_remain) )
    {
        pool_out_w++;
        pool_out_h++;
    }

    pool_out_area = mul(pool_out_w, pool_out_h);

    for (no = 0; no < wr_size.d1; ++no)
    {
        no_offset_1 = mul(no,pool_out_area);
        no_offset_2 = mul(no,input_area);

        for (y = 0; y < pool_out_h; ++y)
        {
            y_offset = mul(y,pool_out_w);
            product_y_stride = mul(y,stride);

            for (x = 0; x < pool_out_w; ++x)
            {
                product_x_stride = mul(x,stride);

                output_offset = no_offset_1 + y_offset + x;

                for (py = 0; py < KERN_ATTR_POOL_KERN_SIZE; ++py)
                {
                  product_2 = mul(input_fm_w,(product_y_stride + py-pad));
                  for (px = 0; px < KERN_ATTR_POOL_KERN_SIZE; ++px)
                  {
                      input_offset = no_offset_2 + product_2 + product_x_stride+px - pad;

                      if (((product_x_stride + px) >= pad) && ((product_x_stride + px) < (pad+input_fm_w))
                       && ((product_y_stride + py) >= pad) && ((product_y_stride + py) < (pad+input_fm_h))
                       && (((*(out+output_offset)) < (*(out+input_offset))) || ((px == 0) && (py == 0))))
                      {
                          *(out+output_offset) = *(out+input_offset);
                      }
                  }
                }
            }
        }
    }
}
```

另外我在 main()函数中添加了性能计数器的打印：

```
int main()
{
    Result res;
    res.msec = 0;
    res.read_mem_cycle = 0;
    res.write_mem_cycle = 0;
    res.mem_cycle = 0;
    res.IF_cycle = 0;
    res.IW_cycle = 0;
    res.ID_EX_cycle = 0;
    res.RDW_cycle = 0;
    res.write_reg_file_cycle = 0;
    res.Load_cycle = 0;
    res.Store_cycle = 0;
    res.MUL_cycle = 0;
    res.R_type_cycle = 0;
    res.wait_cycle = 0;
    bench_prepare(&res);
```

```
    printf("starting convolution\n");
    convolution();
    printf("starting pooling\n");
    pooling();
    bench_done(&res);
    printf("Cycle cnt=%u\n", res.msec);
    printf("read_mem_cycle cnt=%u\n", res.read_mem_cycle);
    printf("write_mem_cycle cnt=%u\n", res.write_mem_cycle);
    printf("mem_cycle cnt=%u\n", res.mem_cycle);
    printf("IF_cycle=%u\n", res.IF_cycle);
    printf("IW_cycle=%u\n", res.IW_cycle);
    printf("ID_EX_cycle=%u\n", res.ID_EX_cycle);
    printf("RDW_cycle=%u\n", res.RDW_cycle);
    printf("write_reg_file_cycle=%u\n", res.write_reg_file_cycle);
    printf("Load_cycle=%u\n", res.Load_cycle);
    printf("Store_cycle=%u\n", res.Store_cycle);
    printf("MUL_cycle=%u\n", res.MUL_cycle);
    printf("R_type_cycle=%u\n", res.R_type_cycle);
    printf("wait_cycle=%u\n",res.wait_cycle);
    return 0;
}
```

下面对于有加速器和无加速器的实现方式进行对比：

在无加速器的情况下(master branch)，上板测试 conv:01 的结果如下：

```
Evaluating convolution benchmark suite...
Launching sw_conv benchmark...
tggetattr: Inappropriate ioctl for device
Initializing read image data
Initializing weights
starting convolution
starting pooling
Cycle cnt=2153390719
read_mem_cycle cnt=1438418
write_mem_cycle cnt=675564
mem_cycle cnt=1579511466
IF_cycle=331749414
IW_cycle=1201409926
ID_EX_cycle=331749418
RDW_cycle=44238347
write_reg_file_cycle=242130067
Load_cycle=1438425
Store_cycle=168891
MUL_cycle=0
R_type_cycle=181676759
wait_cycle=1245648865
Hit good trap
pass 1 / 1
Stopping xl2tpd (via systemctl): xl2tpd.service.
[cpu_with_more_counter 5ca9627] autocmt: cloud_run USER=chencanyu HW_VAL=conv:01
 2 files changed, 30 insertions(+), 26049 deletions(-)
 rewrite run/log/cloud_run_conv_bench.log (99%)
```

在有加速器的情况下（cpu_with_acc_and_more_counter branch），上板测试 conv:02 的结果如下：

```
Evaluating convolution benchmark suite...
Launching hw_conv benchmark...
tggetattr: Inappropriate ioctl for device
Initializing read image data
Initializing weights
Cycle cnt=382592
read_mem_cycle cnt=330
write_mem_cycle cnt=60
mem_cycle cnt=381177
IF_cycle=968
IW_cycle=372061
ID_EX_cycle=972
RDW_cycle=7898
write_reg_file_cycle=663
Load_cycle=337
Store_cycle=15
MUL_cycle=0
R_type_cycle=16
wait_cycle=380510
acc_cycle=297
Hit good trap
pass 1 / 1
Stopping xl2tpd (via systemctl): xl2tpd.service.
[cpu_with_acc_and_more_counter 1732fbe] autocmt: cloud_run USER=chencanyu HW_VAL
=conv:02
```

我们可以明显地对比发现加速器对于访问内存指令的优化非常明显，体现在 IF_cycle，Load_cycle，Store_cycle，read_mem_cycle，write_mem_cycle，

mem_cycle 的明显减少，尤其是对于写内存和取指令的加速非常明显，Write_mem_cycle 加速比达到了 11259，IF_cycle 加速比达到了 342716.3；另外对于访问寄存器堆的优化也非常明显，体现在 write_reg_file_cycle，R_type_cycle 的明显减少，Write_reg_file_cycle 加速比达到了 365203.7。

	加速前	加速后	加速比
Cycle_cnt	2153390719	382592	5628.426
Read_mem_cycle	1438418	330	4358.842
Write_mem_cycle	675564	60	11259.4
Mem_cycle	1579511466	381177	4143.774
IF_cycle	331749414	968	342716.3
IW_cycle	1201409926	372061	3229.067
ID_EX_cyle	331749418	972	341306
RDW_cycle	44238347	7898	5601.209
Write_reg_file_cycle	242130067	663	365203.7
Load_cycle	1438425	337	4268.323
Store_cycle	168891	15	11259.4
R_type_cycle	181676759	16	11354797
Wait_cycle	1245648865	380510	3273.63
acc_cycle	-	297	-

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，仿真、本地上板及云平台调试过程中的难点等）

- 本次实验难点在于对于数据在内存中的摆放方式的理解，输入与输出数据在内存中表示为一个三维数组，我们在访问的时候不能直接通过三维数组来进行处理，而是要深入理解三维数组是如何转化为一维数组然后存放在内存中的，然后通过指针访问这个一维数组。
- 在软件算法实现中，需考虑如何避免出现溢出和精度损失：

这里需要考虑到图像和卷积核中的数据都是按照 16-bit 定点数格式存放的，所以在乘法和加法运算地中间结果可以利用 32 位的 int 类型来表示，然后在利用下面的算式提取最高位（符号位）和中间的 5 位整数位和 10 位小数位，共同构成 16 位 short 类型数存放回内存中。

```
out_store = (((short)(out_store >> FRAC_BIT) & 0x7fff) | ((short)(out_store >> 16) & 0x8000)) + (*(weight+weight_offset_store));
(*(out+output_offset)) = (short)out_store;
```

- 另一个难点是要深入理解各个宏定义的含义，然后在运算的过程中充分并正确地利用这些宏定义。
- 由于要考虑 padding，所以我充分利用了坐标变换的思想去计算每一个点的坐标，然后通过下面的式子判断每一个点是在 padding 中还是在图像中：

```
if (((product_x_stride + kx) >= pad) && ((product_x_stride + kx) < (pad+input_fm_w))
 && ((product_y_stride + ky) >= pad) && ((product_y_stride + ky) < (pad+input_fm_h)))
{
    out_store += mul((short)(*(in+input_offset)), (short)(*(weight+weight_offset)));
}
```

- 另一方面我利用了“空间换时间”的思想去尽可能地优化算法的计算速度。由于在计算过程中需要大量用到乘法运算，而乘法运算的复杂性和耗时远高于加法运算和移位运算，所以我们在优化算法的过程中必须尽可能地减少乘法运算地数量。

  这里我设置了较多的变量用来存储乘法运算的结果，在第一次乘法运算时将结果通过变量储存在栈中，之后只需要调用这个变量即可，不需要重复进行乘法运算。通过这种方法我减少了大量的冗余的乘法运算，确实提高了算法的运行速度。

```
unsigned no, ni, x, y, kx, ky;
unsigned weight_offset = 0;
unsigned weight_offset_store = 0;
unsigned product_x_stride = 0;
unsigned product_y_stride = 0;
unsigned no_offset_1 = 0;
unsigned no_offset_2 = 0;
unsigned ni_offset_1 = 0;
unsigned ni_offset_2 = 0;
unsigned y_offset = 0;
unsigned ky_offset_1 = 0;
unsigned ky_offset_2 = 0;
unsigned conv_size_area = mul(conv_size.d2,conv_size.d3);
unsigned weight_size_area = mul(weight_size.d2,weight_size.d3);
unsigned input_area = mul(input_fm_h,input_fm_w);
int out_store;

conv_out_w = div(conv_out_w, stride);
conv_out_h = div(conv_out_h, stride);

conv_out_w++;
conv_out_h++;

conv_size.d0 = wr_size.d0;
conv_size.d1 = wr_size.d1;
conv_size.d2 = conv_out_h;
conv_size.d3 = conv_out_w;
conv_size_area = mul(conv_size.d2,conv_size.d3);
```

三、 对讲义中思考题（如有）的理解和回答

本次实验没有思考题。

四、 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）

- 本次实验让我收获很大，第一次真正接触到深度学习算法在计算机的底层是如何运行的，以及 CNN 到底是如何构建的，深度学习算法是如何与加速器进行交互的，如何去优化一个深度学习算法。
- 本次实验我大量使用了 printf 进行调试的方法，让我对于如何调试一个“黑箱”程序理解更加深入了。
- 在这里需要感谢李奉致同学帮我解答了几个疑惑。
- 作为最后一个组成原理实验，在回首，真是有些感慨，不敢相信自己的进步竟然有这么大，再次感谢张科老师和各位助教老师的辛苦付出和帮助！