中国科学院大学计算机组成原理（研讨课）

实 验 报 告

学号：2021K8009929016　　姓名：李金明　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

一、 关键代码及说明

1. custom_cpu 改动

这一部分实验我是通过 RISC_V 处理器实现的，需要增添对 MUL 指令的支持，由以下的代码给出：

```
    assign mul = R_Type & funct7[0] & ~funct3[2] & ~funct3[1] &
~funct3[0];
    assign mul_result = ALU_A * ALU_B;
```

当 mul 值为 1 时，说明传入的指令是 MUL，此时会将 ALU_A 与 ALU_B 相乘的结果的低 32 位传入 ALUOut 寄存器（传入寄存器代码更改不大不再给出），其他部分均相同。

2. 卷积（convolution）函数

```
void convolution()
{
    short *in = (short *)addr.rd_addr;
    short *weight = (short *)addr.weight_addr;
    short *out = (short *)addr.wr_addr;

    //unsigned output_offset = 0;
    //unsigned input_offset = 0;

    unsigned input_fm_w = rd_size.d3;
    unsigned input_fm_h = rd_size.d2;

    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;

    unsigned stride = KERN_ATTR_CONV_STRIDE;

    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;  //输出特征图通道数
    conv_size.d2 = conv_out_h;  //输出特征图高度
    conv_size.d3 = conv_out_w;  //输出特征图宽度

    //TODO: Please add your implementation here
    int out_position = 0;   //待填入out数组元素的位置
    int single_weight_size = 1 + mul(weight_size.d2, weight_size.d3);   //一个卷积核的大小
    int total_weight_size = mul(rd_size.d1, single_weight_size);    //一次处理所有通道输入的卷积核的大小
    int input_size = mul(input_fm_h, input_fm_w);   //一个输入数组的大小

    for (int no = 0; no < conv_size.d1; no++)   //输出特征图数量
    {
        int bias_position = mul(no, total_weight_size); //各权重图中bias的位置
        for (int ni = 0; ni < rd_size.d1; ni++) //输入特征图数量
        {
            int read_input_position = mul(ni, input_size);
            int read_weight_position = mul(ni, single_weight_size);
            for (int y = 0; y < conv_size.d2; y++)  //输出特征图高度
            {
                int y_stride = mul(y, stride);
                for (int x = 0; x < conv_size.d3; x++)  //输出特征图宽度
                {
                    int x_stride = mul(x, stride);
                    out[out_position] = weight[bias_position];
                    int out_tem = 0;
                    for (int ky = 0; ky < weight_size.d2; ky++) //权重值高度
                        for (int kx = 0; kx < weight_size.d3; kx++) //权重值宽度
                        {
                            int iw = kx + x_stride - pad;
                            int ih = ky + y_stride - pad;
                            if (iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h)   //边界用0填充
                                continue;
                            int input_position = read_input_position + mul(ih, input_fm_w) + iw;    //所需数据在input数组中的位置
                            int weight_position = bias_position + 1 + read_weight_position + mul(ky, weight_size.d3) + kx;  //所需数据在weight数组中的位置
                            out_tem += mul(in[input_position], weight[weight_position]);
                        }
                    out[out_position] += ((short)(out_tem >> FRAC_BIT) & 0x7fff) | ((short)((out_tem & 0x80000000) >> 16));
                    out_position++;
                }
            }
        }
    }
}
```

这一部分课上已经给出了如下的伪代码：

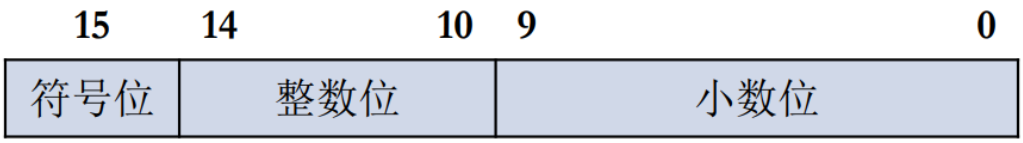
```
for(no = 0; no < Oc; no++)  → 输出特征图数量
{
  for(ni = 0; ni < Ic; ni++)  → 输入特征图数量
  {
    for(y = 0; y < Oh; y++)
      for(x = 0; x < Ow; x++)   } 输出特征图尺寸
      {
        if (ni == 0)
          output_feature_map(no, x, y) = bias(no);
        for(ky = 0; ky < K; ky++)
          for(kx = 0; kx < K; kx++)   } 权重值尺寸
          {
            iw = kx + x * S;
            ih = ky + y * S;
            output_feature_map(no, x, y) +=
                        input(ni, iw, ih) * weight(ni, no, kx, ky);
          }
      }
  }
}
```

改动的主要是：

```
int iw = kx + x_stride - pad;
int ih = ky + y_stride - pad;
if (iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h)
    continue;
```

这是因为给出的伪代码是不考虑边界填充时的算法，当填充的边界宽度为 pad 时，对 iw 和 ih 分别减去 pad，若得到的值小于 0 或大于等于输入图像的尺寸，则代表此时取值为边界填充值，即 0，故可直接跳过该循环后面未进行的步骤。

在该实验中，输入图像、卷积核、输出特征图均使用 short 类型数表示，即使用整数表示小数，表示格式如下：



对于输出值，即 out 数组里填写的内容，需要进行类型转换，该实验中，由以下的代码给出：

```
out[out_position] +=
((short)(out_tem >> FRAC_BIT) & 0x7fff) |
((short)((out_tem & 0x80000000) >> 16));
```

FRAC_BIT 规定了小数部分的位宽，输入数乘上卷积核数后，相当于第 31 位是符号位，第 30-20 位为整数位，第 19-0 位为小数位，因此转化时需将得到的结果右移FRAC_BIT 位并强制转化为 short 类型(即保留移位后的低 16 位)，再讲第 15 位（符号位）置 0，从而得到填入数值的整数位和小数位，通过所得结果的最高位确定填入数值的符号位。

3. 池化（pooling）函数

```
143  void pooling()
144  {
145      short *out = (short *)addr.wr_addr;
146
147      //unsigned output_offset = 0;
148      //unsigned input_offset = 0;
149
150      unsigned input_fm_w = conv_size.d3;
151      unsigned input_fm_h = conv_size.d2;
152
153      unsigned pad = KERN_ATTR_POOL_PAD;
154      unsigned pad_len = pad << 1;
155
156      unsigned pad_w_test = conv_size.d3 - KERN_ATTR_POOL_KERN_SIZE;
157      unsigned pad_h_test = conv_size.d2 - KERN_ATTR_POOL_KERN_SIZE;
158
159      unsigned pool_out_w = pad_w_test + pad_len;
160      unsigned pool_out_h = pad_h_test + pad_len;
161
162      unsigned stride = KERN_ATTR_POOL_STRIDE;
163
164      unsigned pad_w_test_remain = pad_w_test - mul(div(pad_w_test, stride), stride);
165      unsigned pad_h_test_remain = pad_h_test - mul(div(pad_h_test, stride), stride);
166
167      pool_out_w = div(pool_out_w, stride);
168      pool_out_h = div(pool_out_h, stride);
169      pool_out_w++;
170      pool_out_h++;
171
172      if ((!pad) && (pad_w_test_remain || pad_h_test_remain))
173      {
174          pool_out_w++;
175          pool_out_h++;
176      }
177
```

```
    for (int no = 0; no < conv_size.d1; no++)
        for (int y = 0; y < pool_out_h; y++)
            for (int x = 0; x < pool_out_w; x++)    //输出图像尺寸
            {
                short max = 0x8000; //定点表示最小的数
                for (int ky = 0; ky < KERN_ATTR_POOL_KERN_SIZE; ky++)
                    for (int kx = 0; kx < KERN_ATTR_POOL_KERN_SIZE; kx++)   //采样区域尺寸
                    {
                        int iw = kx + mul(x, stride) - pad;
                        int ih = ky + mul(y, stride) - pad;
                        short tem;
                        if (iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h)   //边界用0填充
                            tem = 0;
                        else
                        {
                            int out_read_position = mul(no, input_size) + mul(ih, input_fm_w) + iw;
                            tem = out[out_read_position];
                        }
                        if (tem > max)
                            max = tem;
                    }
                int out_write_position = mul(no, pool_size) + mul(y, pool_out_w) + x;
                out[out_write_position] = max;
            }
}
```

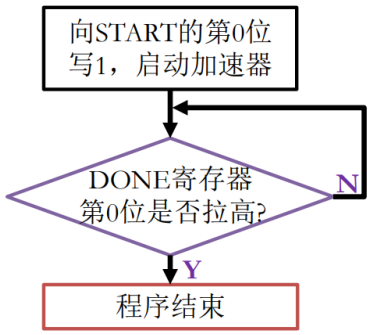
本次实验采用 Max Pooling 算法，即将特征图 2×2 区域中的最大值作为采样后的样本值，依然使用嵌套循环，以步长为 2，对各个 2×2 的区域进行处理

4. Launch_hw_accel 函数

```
#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 = (*gpio_start) | 0x1;
    while(!(*gpio_done) & 0x1);
}
#endif
```

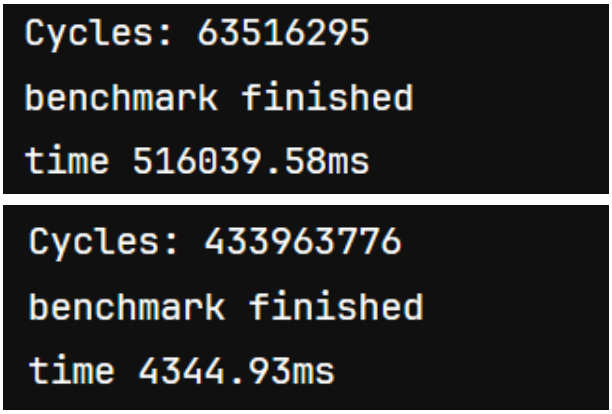
该代码依靠以下的流程图实现：



实现起来较为简单直接，不再赘述。

5. 性能计数器的设计

由于开始时发现若不进行加速，软件执行时间过长，会导致周期统计数溢出，如下图所示：



可以看出，加速后的时间大幅缩短，但统计出来的周期数甚至更多，显然是溢出导致的（性能计数器是 32 位的），因此需要在统计周期数时增加一个性能计数器来统计溢出的情况。

由于输出的数据是十进制，使用两个性能计数器计数，低位性能计数器的计数频率是高位性能计数器的一百万倍。

性能计数器的代码由以下给出：

1) 低位计数器在产生复位信号或其值达到 999,999,999 时同步赋值为 0：

```
423     always @(posedge clk) begin
424         if (rst || cycle_cnt == 32'd999_999_999) cycle_cnt <= 32'b0;
425         else cycle_cnt <= cycle_cnt + 32'b1;
426     end
427     assign cpu_perf_cnt_0 = cycle_cnt;
```

2) 高位计数器在产生复位信号时赋值为 32 位 1，每当低位计数器为 0 时就加 1：

```
477     always @(posedge clk) begin
478         if (rst) cycle_high_cnt <= ~(32'b0);
479         else if (cycle_cnt == 32'b0) cycle_high_cnt <= cycle_high_cnt + 32'b1;
480     end
481     assign cpu_perf_cnt_9 = cycle_high_cnt;
```

则相应的 main 函数代码为：

```
251   int main()
252   {
253       Result res;
254       res.msec = 0;
255       bench_prepare(&res);
256
257   #ifdef USE_HW_ACCEL
258       printf("Launching task...\n");
259       launch_hw_accel();
260   #else
261       printf("starting convolution\n");
262       convolution();
263       printf("starting pooling\n");
264       pooling();
265   #endif
266
267       int result = comparing();
268
269       bench_done(&res);
270       if(res.msec_overflow)
271           printf("Cycles: %u%09u\n", res.msec_overflow, res.msec);
272       else
273           printf("Cycles: %u\n", res.msec);
274       printf("Memory Load/Store: %u\n", res.mem_c);
275       printf("Instructions: %u\n", res.inst_c);
276       printf("Cycles Waiting totally: %u\n", res.wait_c);
277       printf("Load Instructions: %u\n", res.ld_c);
278       printf("Cycles Waiting for Fetching Instructions: %u\n", res.if_c);
279       printf("Cycles Waiting for Instructions: %u\n", res.iw_c);
280       printf("Cycles Waiting for Requesting Data: %u\n", res.memw_c);
281       printf("Cycles Waiting for Reading Data: %u\n", res.rdw_c);
282       printf("benchmark finished\n");
283
284       if (result == 0) {
285           hit_good_trap();
286       } else {
287           nemu_assert(0);
288       }
289
290       return 0;
291   }
```

当高位计数器结果不为 0 时，则打印高位计数器结果与包括前导 0 的低位计数器结果的连接，否则打印低位计数器的数值。

6. 性能计数器打印结果

a. 不使用硬件加速器且使用累加计算乘法

```
52 Cycles: 51603135800
53 Memory Load/Store: 8034454
54 Instructions: 723931544
55 Cycles Waiting totally: 852504553
56 Load Instructions: 3799358
57 Cycles Waiting for Fetching Instructions: 0
58 Cycles Waiting for Instructions: 589628354
59 Cycles Waiting for Requesting Data: 4126183
60 Cycles Waiting for Reading Data: 258751054
61 benchmark finished
62 time 516069.54ms
```

需要指出，与下一部分使用硬件加速器对比，这一部分等待周期可能也不准确也有溢出，但因为不是本实验重点不再进行处理。

b. 不使用硬件加速器但使用 verilog 的乘法计算

```
52 Cycles: 434096270
53 Memory Load/Store: 1497542
54 Instructions: 5424636
55 Cycles Waiting totally: 407237214
56 Load Instructions: 694246
57 Cycles Waiting for Fetching Instructions: 0
58 Cycles Waiting for Instructions: 358599498
59 Cycles Waiting for Requesting Data: 776055
60 Cycles Waiting for Reading Data: 47862678
61 benchmark finished
62 time 4373.66ms
```

c. 使用硬件加速器

```
Cycles: 6796892
Memory Load/Store: 34337
Instructions: 80254
Cycles Waiting totally: 6389014
Load Instructions: 16880
Cycles Waiting for Fetching Instructions: 0
Cycles Waiting for Instructions: 5510588
Cycles Waiting for Requesting Data: 17331
Cycles Waiting for Reading Data: 862153
benchmark finished
time 96.89ms
```

可以观察到三者 cycles(自己编写的)与程序运行的时间(程序给出)的比例大致相同，且与 10ns 一个周期的设计大致相符，可以互相印证。在这个实验中，使用乘法运算比使用累加计算乘法的方式运行周期少了两个数量级，使用硬件加速器又进一步少了两个数量级，提升非常明显。根据统计指令数量的性能计数器，猜测硬件加速器是通过优化有关指令的方式实现的。使用硬件加速器的指令数都比不使用时少了两个数量级。

二、 实验过程中遇到的问题、对问题的思考过程及解决方法

该部分的问题主要有两个方面

1. 刚开始实验的时候软件未执行，fpga 部分报错信息如下:

```
RUNNER_CNT = 0
Completed FPGA configuration
Launching hw_conv benchmark...
tggetattr: Inappropriate ioctl for device
reset: before MMIO access...
reset: MMIO accessed
axi_firewall_unblock: checking firewall status...
axi_firewall_unblock: firewall busy status: 00000000
axi_firewall_unblock: firewall error status: 00000000
main: before DDR accessing...
main: DDR accessed...
reset: before MMIO access...
reset: MMIO accessed
time 1000870.85ms
custom cpu running time out
```

起初不了解是哪里的问题，后来注意到报错信息中的 custom cpu running time out 才去关注 custom_cpu 修改的代码，注意到在删除立即数扩展相关的无用代码时，错误删除了有用的代码，导致 cpu 错误。

2. 性能计数器的使用

由于硬件上性能计数器为 32 位宽的，而不使用硬件加速器运行时间过久，导致统计周期数时溢出，如何处理这一部分信息成为了本部分实验思考最久的问题。

起先想到了使用两个性能计数器，但没想清楚具体实现。经过吉骏雄同学在的提示，想到数字电路课程中有关改变计数器进制有关的知识，实现了计数器同步置 0 的串联，具体代码已在前文给出。

三、 在课后，你花费了大约 10 小时完成此次实验?

四、 对于此次实验的心得、感受和建议

这部分实验对硬件部分的修改不大，集中于软件部分。

我本来对卷积、池化的理解不深，通过阅读课上资料及在网上学习理解了相关知识，并在编程时加深了理解。

本实验的难点集中于统计周期数的性能计数器的实现，因为其牵扯到溢出的问题。后来将该部分知识与数字电路计数器相关的知识结合起来，从而解决了问题，体会到了知识融会贯通使用的重要性。

该部分实验整体上并不难，理解和实现起来较为容易。感谢主讲老师与助教老师的教导以及同学在微信群中的讨论！