中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2018K8009929046　　姓名：何咏哲　　专业：计算机科学与技术

实验序号：5　　实验名称：深度学习算法及硬件加速

注 1：请在实验项目个人本地仓库中创建顶层目录 doc。撰写此 Word 格式实验报告后以 PDF 格式保存在 doc 目录下。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 4 的阿拉伯数字。例如：2018K8009929000-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：学号-prj5-projectname.pdf，例如：2018K8009929000-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 doc 目录下的实验报告 PDF 文件添加到本地仓库，并通过 git push 推送提交。

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、相应信号的仿真波形和信号变化的说明等）

```
59 assign Result = {32{op_and}}    &   {A&B}   |
60                 {32{op_or }}    &   {A|B}   |
61                 {32{op_times}}  &   {A*B}   |
```

在 ALU 中添加了乘法运算。

```
323 assign ALU_control = (opcode[5:3]==`LOAD||opcode==`ADDIU||opcode[5:3]==`STORE||(opcode==`SPECIAL&&func==`ADDU))?`ADD:
324            ( ((opcode==`SPECIAL && (func==`SUBU || func==`MOVZ || func==`MOVN))  || (opcode[5:2]==`BRANCH && opcode!=`BLEZ))?`SUB:
325            ( (opcode==`ANDI || (opcode==`SPECIAL && func==`ANDU))?`AND:
326            ( (opcode==`ORI  || (opcode==`SPECIAL && func==`ORU))?`OR:
327            ( (opcode==`XORI || (opcode==`SPECIAL && func==`XORU))?`XOR:
328            ( (opcode==`SPECIAL && (func==`SLL  || func==`SLLV))?`SL:
329            ( (opcode==`SPECIAL && (func==`SRAU || func==`SRAV))?`SRA:
330            ( (opcode==`SPECIAL && (func==`SRLU || func==`SRLV))?`SRL:
331            ( (opcode==`SLTI || opcode==`BLEZ   || (opcode==`REGIMM && rt[4:1]==`BRANCH_Z) || (opcode==`SPECIAL && (func[5:1]==`MOVE || func==`SLT_U)))?`SLT:
332            ( (opcode==`SPECIAL && func==`NORU)?`NOR:
333            ( (opcode==`SPECIAL2 && func==`MUL)? `TIMES :`SLTU))))))))));
```

同时在 MIPS_CPU 中修改了 MUL 指令相关的控制信号，使得乘法指令也能通过 ALU 正确计算。

```
    int stride_x,stride_y;
    int /*ConvSize, */ WeightSize, WeightNumSize,InputSize;
    int out_set=0;

    //ConvSize = conv_out_h * conv_out_w;
    WeightSize = 1 + weight_size.d2 * weight_size.d3;
    WeightNumSize = rd_size.d1 * WeightSize;
    InputSize = input_fm_w * input_fm_h;

    for(int no=0;no<conv_size.d1;no++){
        for(int y=0;y<conv_out_h;y++){
            stride_y = y*stride;
            for(int x=0;x<conv_out_w;x++){
                stride_x = x*stride;
                int temp = 0;
                out[out_set] = weight[no*WeightNumSize];
                for(int ni=0;ni<rd_size.d1;ni++){
                    for(int ky=0;ky<weight_size.d2;ky++)
                        for(int kx=0;kx<weight_size.d3;kx++){
                            int iw = kx + stride_x - pad;
                            int ih = ky + stride_y - pad;
                            if(iw<0 || ih<0 || iw>=input_fm_w || ih>=input_fm_h)
                                continue;
                            temp += (int)in[ni*InputSize + ih*input_fm_w + iw] * (int)weight[no*WeightNumSize + ni*WeightSize + 1 + ky*weight_size.d3 + kx];
                        }
                }
                out[out_set] += ((short)(temp>>FRAC_BIT)&0x7fff)|((short)(temp>>16)&0x8000);
                out_set++;
            }
        }
    }
}
```

卷积算法，其中优化的部分将在后面做出解释。

```
    int input_size,pool_size;
    int stride_x,stride_y;

    input_size = input_fm_h * input_fm_w;
    pool_size = pool_out_h * pool_out_w;

    for(int no=0;no<conv_size.d1;no++){
        for(int y=0;y<pool_out_h;y++){
            stride_y = y*stride;
            for(int x=0;x<pool_out_w;x++){
                stride_x = x*stride;
                short max = 0x8000;
                for(int ky=0;ky<KERN_ATTR_POOL_KERN_SIZE;ky++)
                    for(int kx=0;kx<KERN_ATTR_POOL_KERN_SIZE;kx++){
                        int iw = kx + stride_x - pad;
                        int ih = ky + stride_y - pad;
                        short now;
                        now = (iw<0 || ih<0 || iw>=input_fm_w || ih>=input_fm_h)? 0 : out[no*input_size + ih*input_fm_w + iw];
                        if(now>max)
                            max = now;
                    }
                out[no*pool_size + y*pool_out_w + x] = max;
            }
        }
    }
}
```

池化算法(max pool)。

```
int main()
{
    //TODO: Please add your own software to control hardware accelerator

    Result res;
    res.msec=0;
    bench_prepare(&res);

    volatile char* base = (void*)HW_BASE_ADDR;
    *(base + HW_ACC_START) = (*(base + HW_ACC_START)) | 0x01;
    int i=0;
    while(1){
        if((*(volatile char*)(base + HW_ACC_DONE)) & 0x01)
            break;
        i++;
    }

    bench_done(&res);
    printf("Cycle Counts: %d\n",res.msec);
    return 0;
}
```

硬件加速器控制访问及性能统计，代码逻辑将在后面进行解释。

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）

问题：上板运行时间超时，导致 bad trap。

解决：首先通过在 sw_conv.c 文件的最外层循环中添加打印变量，发现上板时并没有跑到第 20 个（编号 19）输出通道就结束了。通过讨论区中贾志杰同学的建议，减少了不必要的乘法运算以及寻址操作，大大提高了运行速度，上板因此得以通过。

```
98    for(int no=0;no<conv_size.d1;no++){
99        for(int y=0;y<conv_out_h;y++){
100           stride_y = y*stride;
101           for(int x=0;x<conv_out_w;x++){
102               stride_x = x*stride;
103               int temp = 0;
104               out[out_set] = weight[no*WeightNumSize];
105               for(int ni=0;ni<rd_size.d1;ni++){
106                   for(int ky=0;ky<weight_size.d2;ky++)
107                       for(int kx=0;kx<weight_size.d3;kx++){
108                           int iw = kx + stride_x - pad;
109                           int ih = ky + stride_y - pad;
110                           if(iw<0 || ih<0 || iw>=input_fm_w || ih>=input_fm_h)
111                               continue;
112                           temp += (int)in[ni*InputSize + ih*input_fm_w + iw] * (int)weight[no*WeightNumSize + ni*WeightSize + 1 + ky*weight_size.d3 + kx];
113                       }
114               }
115               out[out_set] += ((short)(temp>>FRAC_BIT)&0x7fff)|((short)(temp>>16)&0x8000);
116               out_set++;
117           }
118       }
119   }
120
121}
```

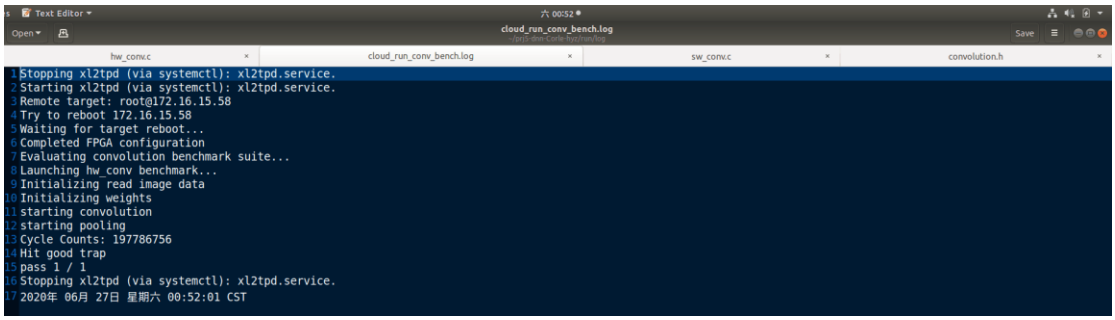
例如 out 数组在地址空间中是顺序存储的，讲义中也提到了是行主序存储形式，因此没有必要每次都计算 out 的偏移量，只需要使用一个 out_set 变量来记录偏移量，并且在需要访问下一个地址时+1 就行。另外，x*stride 和 y*stride 也不用每次都在最内层循环进行计算，只需要在 x 和 y 的值会产生变化时计算即可，并通过 stride_x 和 stride_y 对结果分别进行保存。循环中也有许多常量的乘积每次都被重复计算了，这是不必要的，因此我在计算开始前就将这部分常量的乘积暂存到其他的变量中，如下图所示：

```
89      int stride_x,stride_y;
90      int /*ConvSize, */ WeightSize, WeightNumSize,InputSize;
91      int out_set=0;
92
93      //ConvSize = conv_out_h * conv_out_w;
94      WeightSize = 1 + weight_size.d2 * weight_size.d3;
95      WeightNumSize = rd_size.d1 * WeightSize;
96      InputSize = input_fm_w * input_fm_h;
```

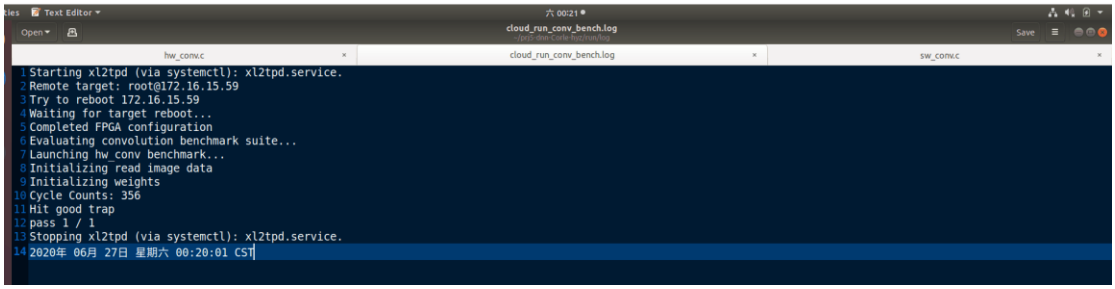
以及按照讲义中的循环顺序会带来很多不必要的计算，可以看到 x、y、no 的变化频率是低于 ni 的，并且 out 数组的偏移量只与前三者有关，因此可以调整一下，将 ni 置于内层循环，减少计算。

问题：在比较软件算法实现和算法硬件加速控制软件实现的性能时，对后者的实现不正确，得到了违背直觉的结果。

例如在使用了硬件加速器时，还将自己写的卷积池化算法通过头文件导入到 hw_conv.c 文件中，导致加速所需的时钟周期比不加速的还要多；再比如在编写硬件加速器时，把 bench_prepare 和 bench_done 放在了一起，统计量没有经过循环，得到的结果与不加速的结果相差了 6 个数量级，仅有 356 个周期，这显然是不正确的。



加速算法错误范例 1

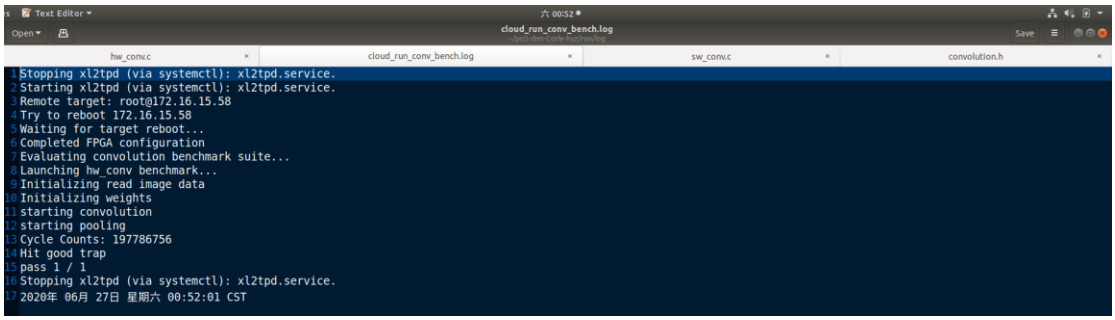


加速算法错误范例 2

解决：上述两种错误方式都 pass 了，只是所得到的周期数不正常，仔细思考其中的联系后发现，使用硬件加速算法时并不需要再额外地进行卷积和池化，该部分算法此时会被自动完成。另外，统计量应该横跨整个循环过程，即在循环开始前创建统计量并通过 bench_prepare 初始化，循环结束后再通过 bench_done 统计消耗的周期数。修改后加速算法所需的周期数变为了 10^6 量级，与不加速的情况相差了 3 个数量级，较为正常。

```
#define HW_ACC_START    0x0000
#define HW_ACC_DONE     0x0008
#define HW_BASE_ADDR    0x40040000


int main()
{
    //TODO: Please add your own software to control hardware accelerator

    Result res;
    res.msec=0;
    bench_prepare(&res);

    volatile char* base = (void*)HW_BASE_ADDR;
    *(base + HW_ACC_START) = (*(base + HW_ACC_START)) | 0x01;
    int i=0;
    while(1){
        if((*(volatile char*)(base + HW_ACC_DONE)) & 0x01)
            break;
        i++;
    }

    bench_done(&res);
    printf("Cycle Counts: %d\n",res.msec);
    return 0;
}
```

修改后的正确加速算法



正确的加速结果

三、 对讲义中思考题（如有）的理解和回答

1、在软件算法实现中，需考虑如何避免出现溢出和精度损失。

输入图像和卷积核的数据文件已按 16-bit 定点数格式存放数据，采用的是 C 语言中的 short 类型。在进行加法运算时不用考虑溢出和精度损失，只有当两个定点数相乘时需要考虑。可以使用一个 int 型的变量暂存乘法结果，由于定点数的小数位数为 10 位，因此寄存于 int 变量的结果中包含了 20 位小数，而我们最后需要写入 out 的也应该是 16-bit 定点数，因此要舍弃末 10 位小数。可以采取截断处理、“0 舍 1 入法”、“恒置 1 法”等舍入方式，此处根据实验要求，采取的应该是简单的截断处理。

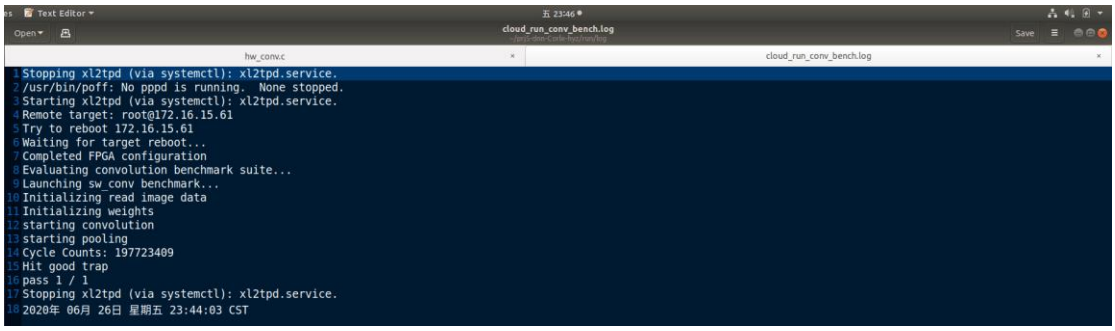
实现方法如下：

```
}
out[out_set] += ((short)(temp>>FRAC_BIT)&0x7fff)|((short)(temp>>16)&0x8000);
out_set++;
```

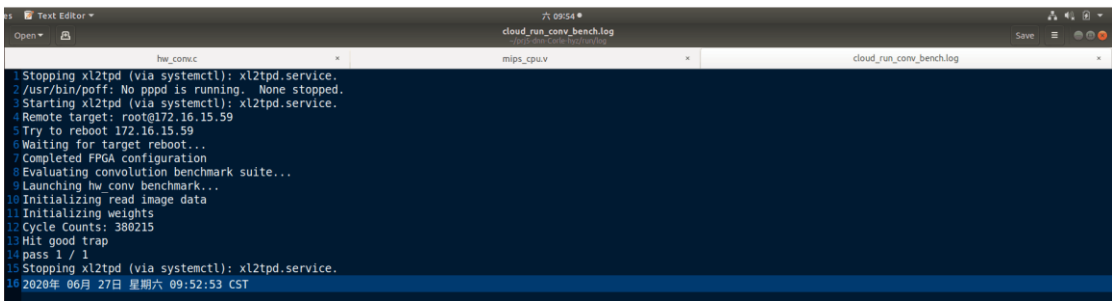
在选取了 temp 变量的有效 15 位的同时，还将原来 temp 的最高位作为结果的符号位。

2、对不同实现方式的处理性能进行对比

I. 算法软件实现：



II. 算法硬件加速控制软件实现：



可以看到，两种实现方式所需的时钟周期相差了 2-3 个数量级，直接使用算法软件实现消耗的周期数是使用算法硬件加速控制软件实现的 520 倍！

四、 在课后，你花费了大约____6____小时完成此次实验。

阅读讲义：1h
编写代码：2h
Debug：1h
等待上板：2h

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）

本次实验较为简单，可以适当增加难度。讲义中并没有提到硬件加速器已经包括了卷积池化的计算，因此我在对比两种实现方式的性能时得到了一些奇怪的结果，希望后续教学中能够把这一条给加上。最后感谢刘听雨同学在 temp 变量的舍弃和符号位处理时对我的帮助。