哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

# 实验报告

开课学期：2021 秋季

课程名称：深度学习体系结构(实验)

实验名称：基于脉动阵列的 CNN 加速器设计

实验性质：综合设计型

实验学时：6 地点：T2608

学生班级：2019 级 4 班

学生学号：190110419

学生姓名：李怡凯

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2021 年 10 月

# 实验内容

本实验要求利用 HLS 实现脉动阵列 IP 核，并使用该 IP 核加速矩阵乘法、卷积运算和 CNN 的前向推导，具体包括：

1. 使用 HLS 编写脉动阵列，并对编写的代码进行 CSim、综合并打包成 IP 核；

2. 利用脉动阵列 IP 核搭建 Block Design，进行综合、实现、生成比特流并导出 Overlay；

3. 编写 Jupyter 程序对矩阵乘法、卷积运算和 CNN 进行测试；

4. 使用 HLS Directives 对脉动阵列进行优化。

# 2. 设计与实现

## 2.1 题目分析

（1） 脉动阵列矩阵乘法实现

使用脉动阵列完成矩阵乘法，首先需要根据脉动阵列规模对矩阵进行分块，本次实验设定的脉动阵列规模为 180*180，根据题目需求，对矩阵乘法 A*B，A 的列数和 B 的行数不会超过 180，所以将 A 按行切分为 180 行的小矩阵，将 B 按列切分为 180 列的小矩阵，每次送入 A 的 1 个小矩阵和 B 的 1 个小矩阵进入脉动阵列进行计算。

确定每次送入脉动阵列的 2 个相乘矩阵后，脉动阵列需要首先计算计算出当前两个矩阵的乘法需要多少次迭代，对矩阵乘法$C_{m\times n} \times D_{n\times k}$，根据观察可知，迭代次数=$n + m + k - 2$。之后的每次迭代，脉动阵列需要完成：

1） 更新送入阵列的数据向量 a_vec 和 b_vec；根据向量下标与迭代次数选择当前送入脉动阵列的数据。

2） 对脉动阵列的每个有效单元进行 1 次脉动；从右下角开始依次从左边和上边的脉动单元上获取数据并更新自身的数据。

脉动阵列迭代完成后，需要再将数据从脉动阵列中读出并加上 bias。

（2） 卷积操作转换为矩阵乘法

为了充分发挥脉动阵列的加速功能，避免 IO 占用过多时间，需要对特征矩阵与卷积核执行 img2col 操作，使卷积操作中卷积核在特征图上扫描执行矩阵乘法的过程转换为单次矩阵乘法。

（3） CNN 网络实现

在完成上面两步之后，CNN 网络的卷积层就能够使用硬件进行加速了，只要在 CNN 网络中添加调用硬件的函数即可。

## 2.2 实验过程

（1） 脉动阵列矩阵乘实现

首先在迭代开始对 a_vec 与 b_vec 进行更新。

```
        for (int j = 0; j < bigger; j++)
        {
            int a_index = piece_a_cell*col*SIDE_LEN + j*col + i - j;
            int b_index = (i - j)*ori_col1 + j + piece_b_cell*SIDE_LEN;
            // TODO: 逐一获取脉动阵列输入向量的各个元素
            a_vec[j] = (i >= j && i < j+col && j < row) ? din_a[a_index] :
0;
            b_vec[j] = (i >= j && i < j+col && j < col1) ? din_b[b_index] :
0;
        }
```

然后是脉动阵列脉动的过程。

```
        for (int i = shorter + longer - 2; i >= 0; i--)
        {
            // TODO: 计算每一个 step 需要处理的 PE 数
            int pe_num = (i > longer - 1)?shorter+longer-1-i:
                         (i > shorter - 1)?shorter:i+1;

            systolic_array_inner_loop:
            for (int j = 0; j < pe_num; j++)
            {
//#pragma HLS PIPELINE II=1
                // TODO: 获取当前 PE 的坐标
                int pos_y = (i >= b_size)?b_size - j - 1:i-j;
                int pos_x = (i >= b_size)?i-b_size+1+j:j;
                // TODO: 获取当前 PE 的左侧输入 a 和上方输入 b
                T a_get = (pos_y == 0)?a_vec[pos_x]:pe[pos_x][pos_y-1].a;
                T b_get = (pos_x == 0)?b_vec[pos_y]:pe[pos_x-1][pos_y].b;
                // TODO: 利用 a 和 b 更新 PE
                pe[pos_x][pos_y].val+=(a_get*b_get);
                pe[pos_x][pos_y].a = a_get;
                pe[pos_x][pos_y].b = b_get;
            }
        }
```

最后是从脉动阵列中将计算结果拷贝出来并加上 bias 的过程

```
    for(int i = 0; i < row; i++)
    {
#pragma HLS UNROLL factor=30
        for(int j = 0; j < col1; j++)
//#pragma HLS loop_flatten
            out[piece_a_cell*SIDE_LEN*ori_col1+i*ori_col1+(piece_b_cell*SIDE
_LEN+j)]
```

```
                = systolic_matrix.pe[i][j].val
                    + bias[piece_a_cell*SIDE_LEN+i];
    }
```

（2） 特征矩阵与卷积核 img2col 具体实现

```
for ch_o in range(core_num):
    for ch_i in range(channel):
        for r in range(core_w):
            for c in range(core_w):
                buf_a[ch_o][ch_i*core_size+r*core_w+c]=core[ch_o][ch_i][r][c
]

for ch_i in range(channel):
    for r in range(core_w):
        for c in range(core_w):
            for r_o in range(out_row):
                for c_o in range(out_col):
                    buf_b[ch_i*core_size+r*core_w+c][r_o*out_col+c_o] =
feature[ch_i][r_o*stride + r][c_o*stride + c]
```

## 2.3 实验结果及分析

矩阵乘法测试运行结果如图 1，卷积测试结果如图 2，CNN 测试结果如图 3。

#### 4.2 硬件矩阵乘法

In [5]:
```
pt0 = time.clock()

RunSystolic(systolic_array_ip, buf_a, buf_b, bias, buf_c)

pt1 = time.clock()
time_hw = pt1 - pt0

print("hardware-accelerated: %fs" % time_hw)
print("speedup: %.2f" % (time_sw/time_hw))
```

```
hardware-accelerated: 1.123767s
speedup: 102.62
```

#### 4.3 校验结果，计算加速比

In [6]:
```
def relative_err(ref, val):
    err = val - ref if val > ref else ref - val
    return err/ref if ref != 0 else err

flag = True

for r in range(row):
    if flag is False:
        break
    for c in range(col1):
        if relative_err(ref[r][c], buf_c[r][c]) > 0.01:
            print("Test failed at (%d, %d)" % (r, c))
            flag = False
            break

if flag:
    print("Test Passed!")

print("\nreference result: ")
print(ref)
print("\narray output:")
print(buf_c)
```

```
Test Passed!
```

图 1 矩阵乘法运行结果

#### 4.2 硬件卷积

In [6]:
```
pt0 = time.clock()

# TODO: 调整卷积核与特征图，以适应脉动阵列
# buf_a = xlnk.cma_array(shape = (core_num, channel*core_size), cacheable = 0, dtype = np.float32)
# buf_b = xlnk.cma_array(shape = (channel*core_size, out_size), cacheable = 0, dtype = np.float32)

for ch_o in range(core_num):
    for ch_i in range(channel):
        for r in range(core_w):
            for c in range(core_w):
                buf_a[ch_o][ch_i*core_size+r*core_w+c]=core[ch_o][ch_i][r][c]

for ch_i in range(channel):
    for r in range(core_w):
        for c in range(core_w):
            for r_o in range(out_row):
                for c_o in range(out_col):
                    buf_b[ch_i*core_size+r*core_w+c][r_o*out_col+c_o] = feature[ch_i][r_o*stride + r][c_o*stride + c]

# 利用硬件矩阵乘法实现卷积加速
RunSystolic(systolic_array_ip, buf_a, buf_b, bias, buf_c)

# 调整输出特征图的形状
buf_c = buf_c.reshape(core_num, out_row, out_col)

pt1 = time.clock()
time_hw = pt1 - pt0

print("hardware-accelerated: %fs" % time_hw)
print("speedup: %.2f" % (time_sw/time_hw))
```

```
hardware-accelerated: 7.301778s
speedup: 10.59
```

#### 4.3 校验结果，计算加速比

In [7]:
```
def relative_err(ref, val):
    err = val - ref if val > ref else ref - val
    return err/ref if ref != 0 else err

flag = True

for ch in range(core_num):
    if flag is False:
        break
    for r in range(out_row):
        if flag is False:
            break
        for c in range(out_col):
            if relative_err(ref[ch][r][c], buf_c[ch][r][c]) > 0.01:
                print("Test failed at (%d, %d, %d)" % (ch, r, c))
                flag = False
                break

if flag:
    print("Test Passed!")

print("\nreference result: ")
print(ref)
print("\narray output:")
print(buf_c)
```

```
Test Passed!
```

图 2 卷积运行结果

In [6]:
```
pt0 = time.clock()

# Conv1
hwConv(systolic_array_ip, KERNEL_W1, STRIDE1, RELU_EN1, PADDING1, image, W_conv1, b_conv1, h_conv1)
hwPool(pool_ip, KERNEL_W11, MODE11, h_conv1, h_pool1)
# Conv2
hwConv(systolic_array_ip, KERNEL_W2, STRIDE2, RELU_EN2, PADDING2, h_pool1, W_conv2, b_conv2, h_conv2)
hwPool(pool_ip, KERNEL_W21, MODE21, h_conv2, h_pool2)
# FC1
hwConv(systolic_array_ip, KERNEL_W3, STRIDE3, RELU_EN3, 0, h_pool2, W_fc1, b_fc1, h_fc1)
# FC2
hwConv(systolic_array_ip, KERNEL_W4, STRIDE4, RELU_EN4, 0, h_fc1, W_fc2, b_fc2, h_fc2)

pt1 = time.clock()
time_hw = pt1 - pt0

print("Hardware-accelerated inference done.")
print("Time consumed: %fs" % time_hw)
print("Speedup: %.2f" % (time_sw/time_hw))

MAX = h_fc2[0][0][0]
result = 0
for ch in range(1, OUT_CH4):
    if (h_fc2[ch][0][0] > MAX):
        MAX = h_fc2[ch][0][0]
        result = ch

print("The image was recognized as " + str(result))
```

```
Hardware-accelerated inference done.
Time consumed: 113.369650s
Speedup: 6.69
The image was recognized as 2
```

图 3 CNN 运行结果

# 3. 总结和感想

本次实验实现了 1 个脉冲阵列，用于加速矩阵乘法，并将 CNN 神经网络中卷积的部分转换为了 1 次矩阵乘，从而充分利用脉冲阵列进行加速。通过本次实验，加深了对脉冲阵列与卷积操作的理解，进一步熟悉 vivado hls 的开发与 python 搭建神经网络的过程。