AAML Final Project

Group 9

412510020 高振翔
312605003 王語
312591015 張昱勛
112033616 洪丞玄

Outline

- Achievement
- Add.h
- Full_connectd.h
- Conv.h
- Comparation

Achievement

- What we modify
  - add.h
  - fully_connected.h
  - **conv.h**
  - cfu.v

```
Perf counters not enabled.
    98M (     97508659 )  cycles total
OK   Golden tests passed
```

```
100%|                                                                                | 200/200 [09:33<00:00,  2.87s/it]
Accuracy: 0.875 %
Latency: 1300795.105 us
```

Add.h

Add.h (SW)

- What we modify => AddFunc()
  - Replace the **clamp** by hardware
    - Reduce function call
    - **no significant effect**

```
inline int8_t AddFunc(int8_t x, int8_t y, const ArithmeticParams& params) {
  const int32_t input1_val = params.input1_offset + x;
  const int32_t input2_val = params.input2_offset + y;
  const int32_t shifted_input1_val = input1_val * (1 << params.left_shift);
  const int32_t shifted_input2_val = input2_val * (1 << params.left_shift);
  const int32_t scaled_input1_val =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          shifted_input1_val, params.input1_multiplier, params.input1_shift);
  const int32_t scaled_input2_val =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          shifted_input2_val, params.input2_multiplier, params.input2_shift);
  const int32_t raw_sum = scaled_input1_val + scaled_input2_val;
  const int32_t raw_output =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          raw_sum, params.output_multiplier, params.output_shift) +
      params.output_offset;
  const int32_t clamped_output =
      std::min(params.quantized_activation_max,
               std::max(params.quantized_activation_min, raw_output));
  return static_cast<int8_t>(clamped_output);
}
```

```
inline int8_t AddFunc(int8_t x, int8_t y, const ArithmeticParams& params) {
  const int32_t input1_val = params.input1_offset + x;
  const int32_t input2_val = params.input2_offset + y;
  const int32_t shifted_input1_val = input1_val * (1 << params.left_shift);
  const int32_t shifted_input2_val = input2_val * (1 << params.left_shift);
  const int32_t scaled_input1_val =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          shifted_input1_val, params.input1_multiplier, params.input1_shift);
  const int32_t scaled_input2_val =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          shifted_input2_val, params.input2_multiplier, params.input2_shift);
  const int32_t raw_sum = scaled_input1_val + scaled_input2_val;
  const int32_t raw_output =
      MultiplyByQuantizedMultiplierSmallerThanOneExp(
          raw_sum, params.output_multiplier, params.output_shift) +
      params.output_offset;
  /*
  const int32_t clamped_output =
      std::min(params.quantized_activation_max,
               std::max(params.quantized_activation_min, raw_output));
  */
  cfu_op0(6, params.quantized_activation_min, params.quantized_activation_max);
  const int32_t clamped_output = cfu_op0(7, raw_output, 0);
  return static_cast<int8_t>(clamped_output);
}
```

Add.h (HW)

- What we add
  - Use combinational circuit to implement
    - **no significant effect**
  - Add two cfu command
    - Set min、max value
    - Push value and get the result
  - Take 4 cycles to get one clamp result

```
// ---------------------------------------------------------------------------
    wire signed [31:0] value;
    reg signed [31:0] minValue;
    reg signed [31:0] maxValue;
    reg signed [31:0] clampedValue;
    assign value = cmd_payload_inputs_0;

    always @* begin
        if (value <= minValue)
            clampedValue = minValue;
        else if (value >= maxValue)
            clampedValue = maxValue;
        else
            clampedValue = value;
    end
```

```
7'd6: begin
  minValue = cmd_payload_inputs_0;
  maxValue = cmd_payload_inputs_1;
end
7'd7: begin
  rsp_payload_outputs_0 <= clampedValue;
```

Fully_connected.h

Fully_connected.h (SW1)

- What we modify
  - Replace the multiply by SIMD
    - **FullyConnectedPerChannel**()
    - **no significant effect**

```
inline void FullyConnectedPerChannel(
    const FullyConnectedParams& params, const int32_t* output_multiplier,
    const int* output_shift, const RuntimeShape& input_shape,
    const int8_t* input_data, const RuntimeShape& filter_shape,
    const int8_t* filter_data, const RuntimeShape& bias_shape,
    const int32_t* bias_data, const RuntimeShape& output_shape,
    int8_t* output_data) {
  const int32_t input_offset = params.input_offset;
  const int32_t output_offset = params.output_offset;
  const int32_t output_activation_min = params.quantized_activation_min;
  const int32_t output_activation_max = params.quantized_activation_max;
  TFLITE_DCHECK_GE(filter_shape.DimensionsCount(), 2);
  TFLITE_DCHECK_EQ(output_shape.DimensionsCount(), 2);

  TFLITE_DCHECK_LE(output_activation_min, output_activation_max);
  const int filter_dim_count = filter_shape.DimensionsCount();
  const int batches = output_shape.Dims(0);
  const int output_depth = output_shape.Dims(1);
  TFLITE_DCHECK_LE(output_depth, filter_shape.Dims(filter_dim_count - 2));
  const int accum_depth = filter_shape.Dims(filter_dim_count - 1);
  for (int b = 0; b < batches; ++b) {
    for (int out_c = 0; out_c < output_depth; ++out_c) {
      int32_t acc = 0;

      for (int d = 0; d < accum_depth; ++d) {
        int32_t input_val = input_data[b * accum_depth + d];
        int32_t filter_val = filter_data[out_c * accum_depth + d];
        acc += filter_val * (input_val + input_offset);
      }
      if (bias_data) {
        acc += bias_data[out_c];
      }
      acc = MultiplyByQuantizedMultiplier(acc, output_multiplier[out_c],
                                          output_shift[out_c]);
      acc += output_offset;
      acc = std::max(acc, output_activation_min);
      acc = std::min(acc, output_activation_max);
      output_data[out_c + output_depth * b] = static_cast<int8_t>(acc);
    }
  }
}
```

```
inline void FullyConnectedPerChannel(
    const FullyConnectedParams& params, const int32_t* output_multiplier,
    const int* output_shift, const RuntimeShape& input_shape,
    const int8_t* input_data, const RuntimeShape& filter_shape,
    const int8_t* filter_data, const RuntimeShape& bias_shape,
    const int32_t* bias_data, const RuntimeShape& output_shape,
    int8_t* output_data) {
  const int32_t input_offset = params.input_offset;
  const int32_t output_offset = params.output_offset;
  const int32_t output_activation_min = params.quantized_activation_min;
  const int32_t output_activation_max = params.quantized_activation_max;
  TFLITE_DCHECK_GE(filter_shape.DimensionsCount(), 2);
  TFLITE_DCHECK_EQ(output_shape.DimensionsCount(), 2);

  TFLITE_DCHECK_LE(output_activation_min, output_activation_max);
  const int filter_dim_count = filter_shape.DimensionsCount();
  const int batches = output_shape.Dims(0);
  const int output_depth = output_shape.Dims(1);
  TFLITE_DCHECK_LE(output_depth, filter_shape.Dims(filter_dim_count - 2));
  const int accum_depth = filter_shape.Dims(filter_dim_count - 1);
  for (int b = 0; b < batches; ++b) {
    for (int out_c = 0; out_c < output_depth; ++out_c) {
      int32_t acc = 0;
      cfu_op0(3, 0, input_offset);
      for (int d = 0; d < accum_depth; d+=4) {
        uint32_t input_val = *((uint32_t *)(input_data + (b * accum_depth + d)));
        uint32_t filter_val = *((uint32_t *)(filter_data + (out_c * accum_depth + d)));
        acc = cfu_op0(4, filter_val, input_val);
      }
      if (bias_data) {
        acc += bias_data[out_c];
      }
      acc = MultiplyByQuantizedMultiplier(acc, output_multiplier[out_c],
                                          output_shift[out_c]);
      acc += output_offset;
      acc = std::max(acc, output_activation_min);
      acc = std::min(acc, output_activation_max);
      output_data[out_c + output_depth * b] = static_cast<int8_t>(acc);
    }
  }
}
```

Fully_connected.h (SW2)

- What we modify
  - Replace the multiply by SIMD
    - **FullyConnected**()
    - **no significant effect**

```
inline void FullyConnected(
    const FullyConnectedParams& params, const RuntimeShape& input_shape,
    const int8_t* input_data, const RuntimeShape& filter_shape,
    const int8_t* filter_data, const RuntimeShape& bias_shape,
    const int32_t* bias_data, const RuntimeShape& output_shape,
    int8_t* output_data) {
  const int32_t input_offset = params.input_offset;
  const int32_t filter_offset = params.weights_offset;
  const int32_t output_offset = params.output_offset;
  const int32_t output_multiplier = params.output_multiplier;
  const int output_shift = params.output_shift;
  const int32_t output_activation_min = params.quantized_activation_min;
  const int32_t output_activation_max = params.quantized_activation_max;
  TFLITE_DCHECK_GE(filter_shape.DimensionsCount(), 2);
  TFLITE_DCHECK_GE(output_shape.DimensionsCount(), 1);

  TFLITE_DCHECK_LE(output_activation_min, output_activation_max);
  const int filter_dim_count = filter_shape.DimensionsCount();
  const int output_dim_count = output_shape.DimensionsCount();
  const int batches = FlatSizeSkipDim(output_shape, output_dim_count - 1);
  const int output_depth = output_shape.Dims(output_dim_count - 1);
  TFLITE_DCHECK_LE(output_depth, filter_shape.Dims(filter_dim_count - 2));
  const int accum_depth = filter_shape.Dims(filter_dim_count - 1);
  for (int b = 0; b < batches; ++b) {
    for (int out_c = 0; out_c < output_depth; ++out_c) {
      int32_t acc = 0;

      for (int d = 0; d < accum_depth; ++d) {
        int32_t input_val = input_data[b * accum_depth + d];
        int32_t filter_val = filter_data[out_c * accum_depth + d];
        acc += (filter_val + filter_offset) * (input_val + input_offset);
      }
      if (bias_data) {
        acc += bias_data[out_c];
      }
      acc = MultiplyByQuantizedMultiplier(acc, output_multiplier, output_
      acc += output_offset;
      acc = std::max(acc, output_activation_min);
      acc = std::min(acc, output_activation_max);
      output_data[out_c + output_depth * b] = static_cast<int8_t>(acc);
    }
  }
}
```

```
inline void FullyConnected(
    const FullyConnectedParams& params, const RuntimeShape& input_shape,
    const int8_t* input_data, const RuntimeShape& filter_shape,
    const int8_t* filter_data, const RuntimeShape& bias_shape,
    const int32_t* bias_data, const RuntimeShape& output_shape,
    int8_t* output_data) {
  const int32_t input_offset = params.input_offset;
  const int32_t filter_offset = params.weights_offset;
  const int32_t output_offset = params.output_offset;
  const int32_t output_multiplier = params.output_multiplier;
  const int output_shift = params.output_shift;
  const int32_t output_activation_min = params.quantized_activation_min;
  const int32_t output_activation_max = params.quantized_activation_max;
  TFLITE_DCHECK_GE(filter_shape.DimensionsCount(), 2);
  TFLITE_DCHECK_GE(output_shape.DimensionsCount(), 1);

  TFLITE_DCHECK_LE(output_activation_min, output_activation_max);
  const int filter_dim_count = filter_shape.DimensionsCount();
  const int output_dim_count = output_shape.DimensionsCount();
  const int batches = FlatSizeSkipDim(output_shape, output_dim_count - 1);
  const int output_depth = output_shape.Dims(output_dim_count - 1);
  TFLITE_DCHECK_LE(output_depth, filter_shape.Dims(filter_dim_count - 2));
  const int accum_depth = filter_shape.Dims(filter_dim_count - 1);
  for (int b = 0; b < batches; ++b) {
    for (int out_c = 0; out_c < output_depth; ++out_c) {
      int32_t acc = 0;
      cfu_op0(3, filter_offset, input_offset);
      for (int d = 0; d < accum_depth; d+=4) {
        uint32_t input_val = *((uint32_t *)(input_data + (b * accum_depth + d)));
        uint32_t filter_val = *((uint32_t *)(filter_data + (out_c * accum_depth + d)));
        acc = cfu_op0(4, filter_val, input_val);
      }
      if (bias_data) {
        acc += bias_data[out_c];
      }
      acc = MultiplyByQuantizedMultiplier(acc, output_multiplier, output_shift);
      acc += output_offset;
      acc = std::max(acc, output_activation_min);
      acc = std::min(acc, output_activation_max);
      output_data[out_c + output_depth * b] = static_cast<int8_t>(acc);
    }
  }
}
```

Fully_connected.h (HW)

- What we add
  - Add the SIMD hardware for multiply parallel
    - **no significant effect**
  - Add two cfu command
    - Set **filter_offset、input_offset**
    - Return cumulated product result

```
--- a/proj/AAML_final_proj/cfu.v
+++ b/proj/AAML_final_proj/cfu.v
@@ -41,6 +41,22 @@ module Cfu (
         else
             clampedValue = value;
     end
+//---------------------------------------------------------------------------
+  reg [15:0] filter_offset, input_offset;
+
+  // SIMD multiply step:
+  wire signed [16:0] prod_fc_0, prod_fc_1, prod_fc_2, prod_fc_3;
+  assign prod_fc_0 =  ($signed(cmd_payload_inputs_0[7 : 0]) + $signed(filter_offset))
+                   * ($signed(cmd_payload_inputs_1[7 : 0]) + $signed(input_offset));
+  assign prod_fc_1 =  ($signed(cmd_payload_inputs_0[15: 8]) + $signed(filter_offset))
+                   * ($signed(cmd_payload_inputs_1[15: 8]) + $signed(input_offset));
+  assign prod_fc_2 =  ($signed(cmd_payload_inputs_0[23:16]) + $signed(filter_offset))
+                   * ($signed(cmd_payload_inputs_1[23:16]) + $signed(input_offset));
+  assign prod_fc_3 =  ($signed(cmd_payload_inputs_0[31:24]) + $signed(filter_offset))
+                   * ($signed(cmd_payload_inputs_1[31:24]) + $signed(input_offset));
+
+  wire signed [31:0] sum_prods_fc;
+  assign sum_prods_fc = prod_fc_0 + prod_fc_1 + prod_fc_2 + prod_fc_3;

   // Only not ready for a command when we have a response.
   assign cmd_ready = ~rsp_valid;
@@ -65,6 +81,16 @@ module Cfu (
           InputOffset <= cmd_payload_inputs_0[15:0];
           rsp_payload_outputs_0 <= 0'b0;
         end
+        7'd3: begin
+          filter_offset <= cmd_payload_inputs_0;
+          input_offset <= cmd_payload_inputs_1;
+          rsp_payload_outputs_0 <= 0'b0;
+        end
+        7'd4: begin
+          filter_offset <= filter_offset;
+          input_offset <= input_offset;
+          rsp_payload_outputs_0 <= rsp_payload_outputs_0 + sum_prods_fc;
+        end
```

Conv.h

Conv.h (version 1)

- Use **uint64_t**
  - Get 2 uint32_t value
  - Get 8 uint8 input_data、filter_data

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+4 < input_depth; in_channel += 4) {
  uint32_t input_val = *((uint32_t *)(input_data + Offset(
      input_shape, batch, in_y, in_x, in_channel)));

  uint32_t filter_val = *((uint32_t *)(filter_data + Offset(
      filter_shape, out_channel, filter_y, filter_x, in_channel)));


  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val, /* in1= */ filter_val);
}
// left-over
```

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+4 < input_depth; in_channel += 8) {
    uint64_t input_val = *((uint64_t *)(input_data + Offset(
                input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    uint64_t filter_val = *((uint64_t *)(filter_data + Offset(
                filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val, /* in1= */ filter_val);
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val >> 32, /* in1= */ filter_val >> 32);
}
// left-over
```

Conv.h (version 1) Result

- Golden Test improve from 154M to **120M**
- Latency improve from 2034443 to **1601753**

```
12,AVERAGE_POOL_2D,52
13,RESHAPE,2
14,FULLY_CONNECTED,15
15,SOFTMAX,31
Perf counters not enabled.
   120M (    120271916 )  cycles total
Copied 3072 bytes at 0x400c3700
Running imgc

[4]+  Stopped                 make load
michael@michael-System-Product-Name:~/hw/final_project/CFU
```

```
8, 1601666, 1602331, 1601690, 1601589, 1601068, 1600871, 1601076, 1602110, 1601607, 1601422, 16
2198, 1602173, 1602233, 1602002, 1601710, 1601928, 1602575, 1601488, 1601390, 1602461, 1602138,
1601584, 1601951, 1601297, 1601613, 1601224, 1601733, 1601841, 1601738, 1601769, 1601990, 16019
7, 1602171, 1601895, 1601675, 1601154, 1601860, 1601872, 1602553, 1601977, 1602160, 1601196, 16
2286, 1601639, 1602208, 1601701, 1601498, 1601646, 1602214, 1601285, 1601406, 1602203, 1601960,
1601329, 1600878, 1601987, 1601074, 1601496, 1601646, 1601199, 1601766, 1601860, 1602312, 16014
2, 1602016, 1601482, 1601094, 1601191, 1601914, 1601917, 1602006, 1601990, 1602118, 1601824, 16
100%|
Accuracy: 0.875 %
Latency: 1601753.7 us
michael@michael-System-Product-Name:~/hw/final_project/CFU-Playground/proj/AAML_final_proj$
```

Conv.h (version 2)

- Use my_uint64_t
  - No uint128 for compiler
  - Try to save time of shift operation
    - Replace shift by typedef

```
extern long long unsigned my_cycles;
```

```
#include <stdint.h>
extern long long unsigned my_cycles;

typedef struct {
  uint32_t low;
  uint32_t high;
} my_uint64_t;
```

```
        int in_channel;
        // unrolling factor: 4
        for (in_channel = 0; in_channel+4 < input_depth; in_channel += 8) {
            uint64_t input_val = *((uint64_t *)(input_data + Offset(
                        input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

            uint64_t filter_val = *((uint64_t *)(filter_data + Offset(
                        filter_shape, out_channel, filter_y, filter_x, in_channel)));

          cfu_op0(/* funct7= */ 0, /* in0= */ input_val, /* in1= */ filter_val);
          acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val >> 32, /* in1= */ filter_val >> 32);
        }
        // left-over
        for (; in_channel < input_depth; in_channel++) {
          int8_t input_val = *((int8_t *)(input_data + Offset(
              input_shape, batch, in_y, in_x, in_channel)));

          int8_t filter_val = *((int8_t *)(filter_data + Offset(
              filter_shape, out_channel, filter_y, filter_x, in_channel)));

          acc += filter_val * (input_val + input_offset);
        }
```

```
        int in_channel;
        // unrolling factor: 4
        for (in_channel = 0; in_channel+4 < input_depth; in_channel += 8) {
            my_uint64_t input_val = *((my_uint64_t *)(input_data + Offset(
                        input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

            my_uint64_t filter_val = *((my_uint64_t *)(filter_data + Offset(
                        filter_shape, out_channel, filter_y, filter_x, in_channel)));

          cfu_op0(/* funct7= */ 0, /* in0= */ input_val.low, /* in1= */ filter_val.low);
          acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.high, /* in1= */ filter_val.high);
        }
        // left-over
        for (; in_channel < input_depth; in_channel++) {
          int8_t input_val = *((int8_t *)(input_data + Offset(
              input_shape, batch, in_y, in_x, in_channel)));

          int8_t filter_val = *((int8_t *)(filter_data + Offset(
              filter_shape, out_channel, filter_y, filter_x, in_channel)));

          acc += filter_val * (input_val + input_offset);
        }
```

Conv.h (version 2) Result

- Golden Test improve from 120M to **109M**
- Latency improve from 1601753 to **1439047**

```
14,FULLY_CONNECTED,15
15,SOFTMAX,28
Perf counters not enabled.
   109M (    109080364 )  cycles total
OK   Golden tests passed
---

Tests for imgc model
====================
 0: Run with airplane input
 1: Run with car input
 2: Run with bird input
 3: Run with cat input
 g: Run golden tests (check for expected outputs)
 x: eXit to previous menu
imgc>
```

```
0, 1438960, 1439569, 1438979, 1438847, 1438463, 1438133, 1438314, 1439477, 1438969, 1438828, 143891
9432, 1439444, 1439630, 1439287, 1439161, 1439141, 1439760, 1438770, 1438674, 1439762, 1439420, 143
1438895, 1439174, 1438632, 1438845, 1438489, 1439160, 1439086, 1439066, 1439034, 1439206, 1439244,
6, 1439441, 1439180, 1439039, 1438571, 1439080, 1439104, 1439816, 1439251, 1439426, 1438431, 143891
9502, 1439121, 1439464, 1439021, 1438764, 1438945, 1439561, 1438650, 1438866, 1439553, 1439221, 143
1438591, 1438119, 1439231, 1438403, 1438691, 1438942, 1438438, 1439153, 1439143, 1439450, 1438855,
6, 1439338, 1438713, 1438373, 1438508, 1439285, 1439197, 1439372, 1439160, 1439394, 1439136, 143910
100%|
Accuracy: 0.875 %
Latency: 1439047.975 us
michael@michael-System-Product-Name:~/hw/final_project/CFU-Playground/proj/AAML_final_proj$
```

Conv.h (version 3)

Use **my_uint128_unroll_t**

- Increase uint32_t in the typedef
  - Equal to two my_uint64_t

```
typedef struct {
  uint32_t item0;
  uint32_t item1;
  uint32_t item2;
  uint32_t item3;
} my_uint128_unroll_t;
```

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+4 < input_depth; in_channel += 8) {
    my_uint64_t input_val = *((my_uint64_t *)(input_data + Offset(
                    input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    my_uint64_t filter_val = *((my_uint64_t *)(filter_data + Offset(
                    filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.low, /* in1= */ filter_val.low);


  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.high, /* in1= */ filter_val.high);
}
// left-over
for (; in_channel < input_depth; in_channel++) {
  int8_t input_val = *((int8_t *)(input_data + Offset(
      input_shape, batch, in_y, in_x, in_channel)));

  int8_t filter_val = *((int8_t *)(filter_data + Offset(
      filter_shape, out_channel, filter_y, filter_x, in_channel)));

  acc += filter_val * (input_val + input_offset);
}
```

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+4 < input_depth; in_channel += 16) {
    my_uint128_unroll_t input_val = *((my_uint128_unroll_t *)(input_data + Offset(
                    input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    my_uint128_unroll_t filter_val = *((my_uint128_unroll_t *)(filter_data + Offset(
                    filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item0, /* in1= */ filter_val.item0);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item1, /* in1= */ filter_val.item1);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item2, /* in1= */ filter_val.item2);
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item3, /* in1= */ filter_val.item3);
}
// left-over
for (; in_channel < input_depth; in_channel++) {
  int8_t input_val = *((int8_t *)(input_data + Offset(
      input_shape, batch, in_y, in_x, in_channel)));

  int8_t filter_val = *((int8_t *)(filter_data + Offset(
      filter_shape, out_channel, filter_y, filter_x, in_channel)));

  acc += filter_val * (input_val + input_offset);
}
```

Conv.h (version 3) Result

- Golden Test improve from 109M to **106M**
- Latency improve from 1439047 to **1392803**

```
12,AVERAGE_POOL_2D,52
13,RESHAPE,3
14,FULLY_CONNECTED,9
15,SOFTMAX,29
Perf counters not enabled.
   106M (    105679173 )  cycles total
OK   Golden tests passed
---

Tests for imgc model
====================
 0: Run with airplane input
 1: Run with car input
 2: Run with bird input
 3: Run with cat input
 g: Run golden tests (check for expected outputs)
 x: eXit to previous menu
imgc>
```

```
1, 1392688, 1393554, 1392754, 1392599, 1392214, 1391947, 1392102, 1393247, 1392695, 1392
3207, 1393236, 1393367, 1393059, 1392931, 1392949, 1393543, 1392496, 1392408, 1393522, 1
1392668, 1392968, 1392374, 1392593, 1392258, 1392838, 1392886, 1392810, 1392771, 1392977
4, 1393220, 1392948, 1392751, 1392309, 1392862, 1392894, 1393620, 1393016, 1393159, 1392
3305, 1392846, 1393234, 1392769, 1392511, 1392718, 1393334, 1392408, 1392658, 1393288, 1
1392338, 1391951, 1393026, 1392173, 1392459, 1392696, 1392205, 1392891, 1392967, 1393285
0, 1393046, 1392521, 1392110, 1392204, 1393071, 1392982, 1393111, 1392950, 1393141, 1392
100%|
Accuracy: 0.875 %
Latency: 1392803.7 us
michael@michael-System-Product-Name:~/hw/final_project/CFU-Playground/proj/AAML_final_pr
```

Conv.h (version 4)

Increase uint32_t in the typedef

my_uint256_unroll_t

Equal to two my_uint128_t

```
typedef struct {
  uint32_t item0;
  uint32_t item1;
  uint32_t item2;
  uint32_t item3;
  uint32_t item4;
  uint32_t item5;
  uint32_t item6;
  uint32_t item7;
} my_uint256_unroll_t;
```

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+4 < input_depth; in_channel += 16) {

    my_uint128_unroll_t input_val = *((my_uint128_unroll_t *)(input_data + Offset(
                input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    my_uint128_unroll_t filter_val = *((my_uint128_unroll_t *)(filter_data + Offset(
                filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item0, /* in1= */ filter_val.item0);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item1, /* in1= */ filter_val.item1);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item2, /* in1= */ filter_val.item2);
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item3, /* in1= */ filter_val.item3);

}
```

```
int in_channel;
// unrolling factor: 4
for (in_channel = 0; in_channel+16 < input_depth; in_channel += 32) {
    my_uint256_unroll_t input_val = *((my_uint256_unroll_t *)(input_data + Offset(
                input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    my_uint256_unroll_t filter_val = *((my_uint256_unroll_t *)(filter_data + Offset(
                filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item0, /* in1= */ filter_val.item0);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item1, /* in1= */ filter_val.item1);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item2, /* in1= */ filter_val.item2);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item3, /* in1= */ filter_val.item3);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item4, /* in1= */ filter_val.item4);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item5, /* in1= */ filter_val.item5);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item6, /* in1= */ filter_val.item6);
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item7, /* in1= */ filter_val.item7);
}
for (; in_channel+4 < input_depth; in_channel += 16) {
    my_uint128_unroll_t input_val = *((my_uint128_unroll_t *)(input_data + Offset(
                input_shape, batch, in_y, in_x, in_channel + group * filter_input_depth)));

    my_uint128_unroll_t filter_val = *((my_uint128_unroll_t *)(filter_data + Offset(
                filter_shape, out_channel, filter_y, filter_x, in_channel)));

  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item0, /* in1= */ filter_val.item0);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item1, /* in1= */ filter_val.item1);
  cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item2, /* in1= */ filter_val.item2);
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val.item3, /* in1= */ filter_val.item3);
}
for (; in_channel+4 < input_depth; in_channel += 4) {
  uint32_t input_val = *((uint32_t *)(input_data + Offset(
      input_shape, batch, in_y, in_x, in_channel)));

  uint32_t filter_val = *((uint32_t *)(filter_data + Offset(
      filter_shape, out_channel, filter_y, filter_x, in_channel)));
  acc_cfu = cfu_op0(/* funct7= */ 0, /* in0= */ input_val, /* in1= */ filter_val);
}
```

Conv.h (version 4) Result

- Golden Test improve from 106M to **98M**
- Latency improve from 1392803 to **1300795**

```
Perf counters not enabled.
    98M (     97508659 )  cycles total
OK   Golden tests passed
```

```
100%|                                                  | 200/200 [09:33<00:00,  2.87s/it]
Accuracy: 0.875 %
Latency: 1300795.105 us
```

Comparation

Comparation

	Golden Test Latency	Script Latency	Acc
uint32_t	154M	2034443	0.875%
uint64_t	120M	1601753	0.875%
my_uint64_t	109M	1439047	0.875%
my_uint128_t	106M	1392803	0.875%
my_uint256_t	98M	**1300795**	0.875%

The End