Difference of quantization behavior between onednn and cldnn

Felix Kim

Intro

- A unit test failed since accuracy issue
  - TestName
    - deconv_scale_actv_quant_u8_eltw_scale_actv_quant_i8.basic/15
  - Error Message
    - The difference between ref[i] and output_ptr[i] is 28, which exceeds tolerance, where ref[i] evaluates to 25, output_ptr[i] evaluates to 53, and tolerance evaluates to 2.099999046325684.

Operations Review

Quantization

- Meaning
  - Quantization is the process of mapping input values from a large set (often a continuous set) to output values in a (countable) smaller set, often with a finite number of elements
- Formula

$$output = \frac{\lfloor (clamp(input; input\_low, input\_high) - input\_low) * s \rceil}{s} + input\_low$$

$$s = \frac{levels - 1}{input\_high - input\_low}$$

$$x_{f32}[:] = scale_x \cdot (x_{int8}[:] - zp_x)$$

- Usecase
  - Mixed precision model?

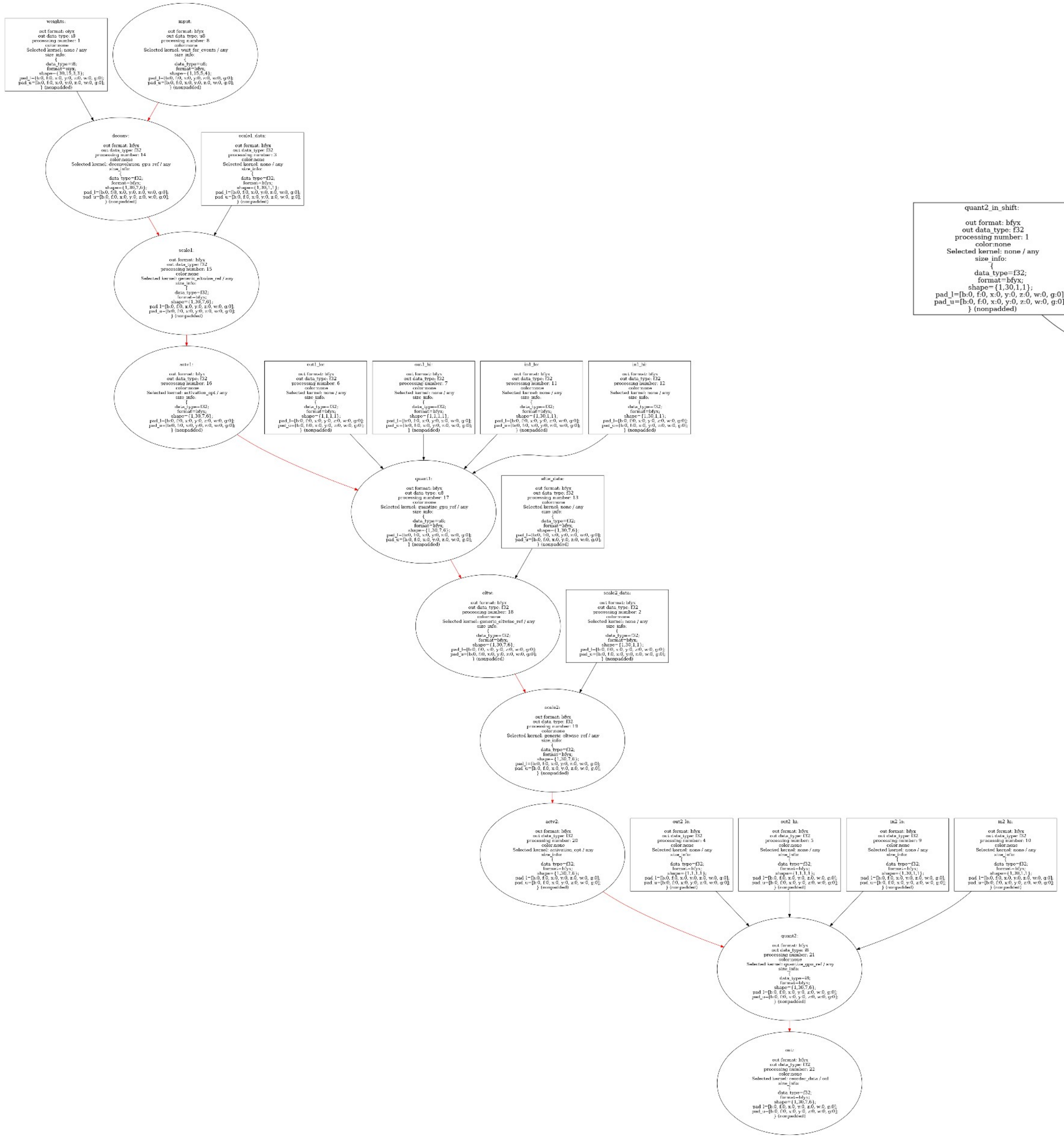
Observations

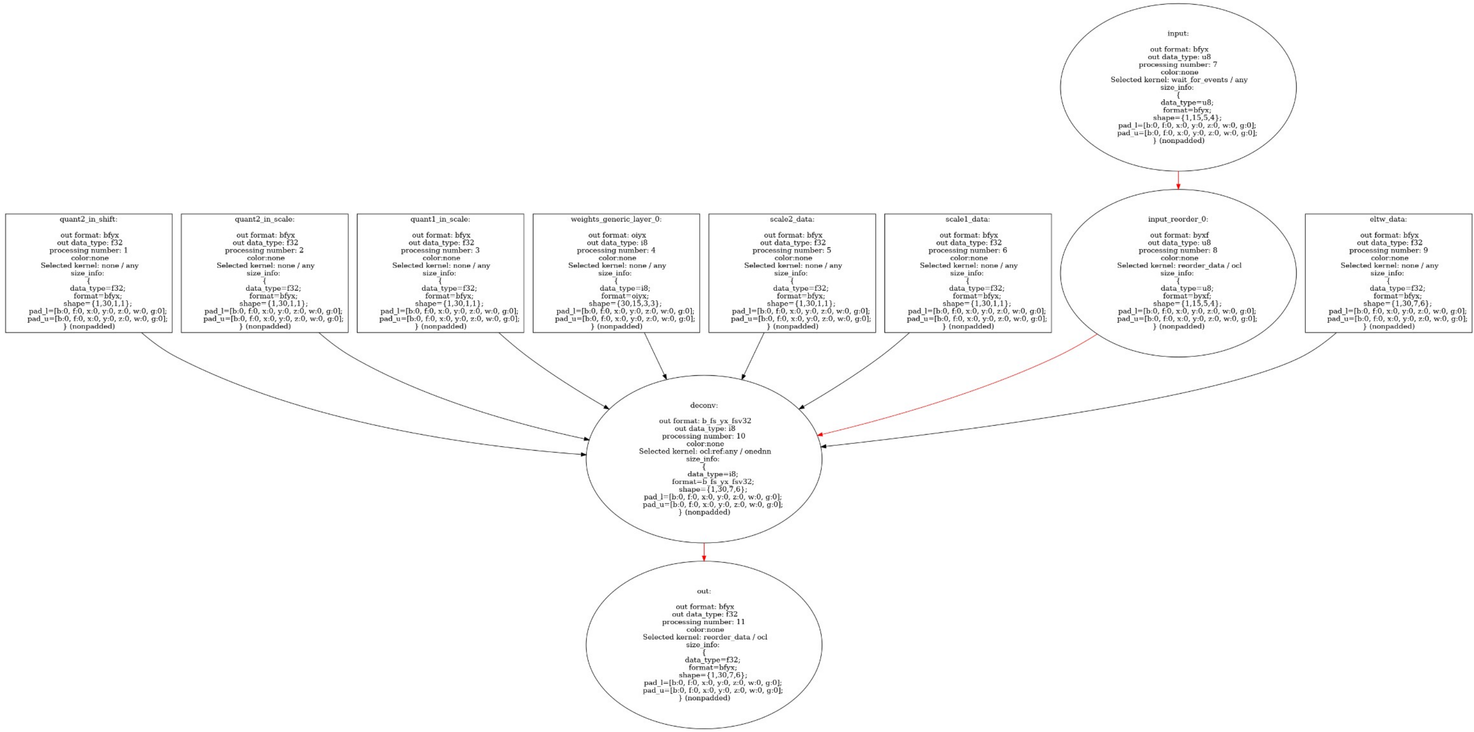
Topology

```
input_layout("input", get_input_layout(p)),
data("weights", get_mem(get_weights_layout(p))),
data("scale1_data", get_mem(get_per_channel_layout(p), 1.f / p.kernel.count())),
data("in1_lo", get_mem(get_per_channel_layout(p), 0)),
data("in1_hi", get_mem(get_per_channel_layout(p), 1, max_random)),
data("out1_lo", get_mem(get_single_element_layout(p), 0)),
data("out1_hi", get_mem(get_single_element_layout(p), 255)),
data("eltw_data", get_mem(layout(p.default_type, p.input_format, p.out_shape))),
data("scale2_data", get_mem(get_per_channel_layout(p), 1.f / p.kernel.count())),
data("in2_lo", get_mem(get_per_channel_layout(p), min_random, 0)),
data("in2_hi", get_mem(get_per_channel_layout(p), 1, max_random)),
data("out2_lo", get_mem(get_single_element_layout(p), -127)),
data("out2_hi", get_mem(get_single_element_layout(p), 127)),
```

```
deconvolution("deconv", input_info("input"), { "weights" }, p.groups, p.stride, p.pad),
eltwise("scale1", { input_info("deconv"), input_info("scale1_data") }, eltwise_mode::prod),
activation("actv1", input_info("scale1"), activation_func::relu),
quantize("quant1", input_info("actv1"), input_info("in1_lo"), input_info("in1_hi"),
        input_info("out1_lo"), input_info("out1_hi"), 256, data_types::u8),
eltwise("eltw", { input_info("quant1"), input_info("eltw_data") }, eltwise_mode::sum, p.default_type),
eltwise("scale2", { input_info("eltw"), input_info("scale2_data") }, eltwise_mode::prod),
activation("actv2", input_info("scale2"), activation_func::relu),
quantize("quant2", input_info("actv2"), input_info("in2_lo"), input_info("in2_hi"),
        input_info("out2_lo"), input_info("out2_hi"), 255, data_types::i8),
reorder("out", input_info("quant2"), p.default_format, data_types::f32)
```

Graph

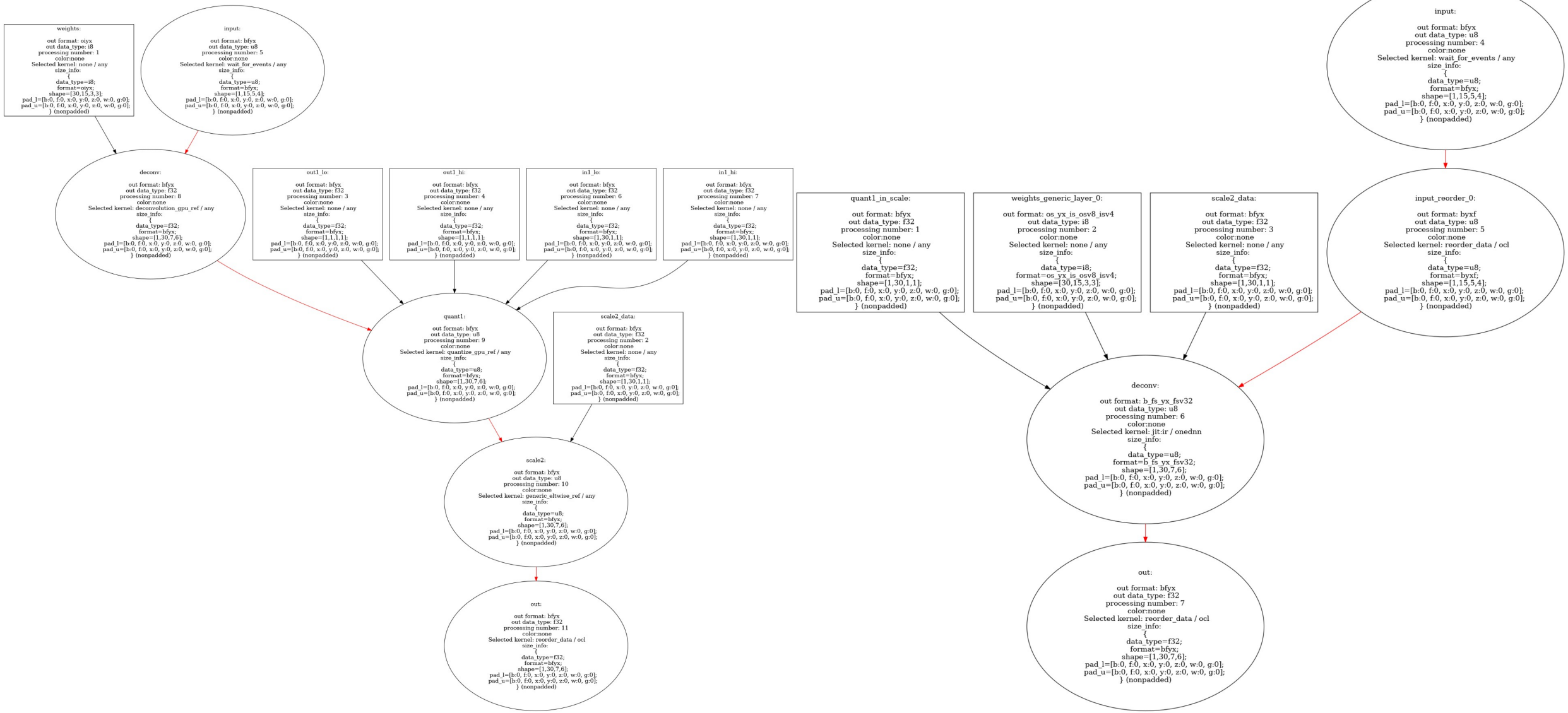



input:
out format: bfyx
out data_type: u8
processing number: 7
color:none
Selected kernel: wait_for_events / any
size_info:
{
data_type=u8;
format=bfyx;
shape={1,15,5,4};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
quant2_in_shift:
out format: bfyx
out data_type: f32
processing number: 1
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,1,1};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
quant2_in_scale:
out format: bfyx
out data_type: f32
processing number: 2
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,1,1};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
quant1_in_scale:
out format: bfyx
out data_type: f32
processing number: 3
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,1,1};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
weights_generic_layer_0:
out format: oiyx
out data_type: i8
processing number: 4
color:none
Selected kernel: none / any
size_info:
{
data_type=i8;
format=oiyx;
shape={30,15,3,3};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
scale2_data:
out format: bfyx
out data_type: f32
processing number: 5
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,1,1};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
scale1_data:
out format: bfyx
out data_type: f32
processing number: 6
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,1,1};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
input_reorder_0:
out format: byxf
out data_type: u8
processing number: 8
color:none
Selected kernel: reorder_data / ocl
size_info:
{
data_type=u8;
format=byxf;
shape={1,15,5,4};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
eltw_data:
out format: bfyx
out data_type: f32
processing number: 9
color:none
Selected kernel: none / any
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,7,6};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
deconv:
out format: b_fs_yx_fsv32
out data_type: i8
processing number: 10
color:none
Selected kernel: ocl:ref:any / onednn
size_info:
{
data_type=i8;
format=b_fs_yx_fsv32;
shape={1,30,7,6};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)
out:
out format: bfyx
out data_type: f32
processing number: 11
color:none
Selected kernel: reorder_data / ocl
size_info:
{
data_type=f32;
format=bfyx;
shape={1,30,7,6};
pad_l=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
pad_u=[b:0, f:0, x:0, y:0, z:0, w:0, g:0];
} (nonpadded)

Minimal Topology

```
input_layout("input", get_input_layout(p)),
data("weights", get_mem(get_weights_layout(p))),
data("in1_lo", get_mem(get_per_channel_layout(p), 0)),
data("in1_hi", get_mem(get_per_channel_layout(p), 1, max_random)),
data("out1_lo", get_mem(get_single_element_layout(p), 0)),
data("out1_hi", get_mem(get_single_element_layout(p), 255)),
data("scale2_data", get_mem(get_per_channel_layout(p), 1.f / p.kernel.count())),
      You, now • Uncommitted changes
deconvolution("deconv", input_info("input"), { "weights" }, p.groups, p.stride, p.pad),
quantize("quant1", input_info("deconv"), input_info("in1_lo"), input_info("in1_hi"),
        input_info("out1_lo"), input_info("out1_hi"), 256, data_types::u8),
eltwise("scale2", { input_info("quant1"), input_info("scale2_data") }, eltwise_mode::prod),
reorder("out", input_info("scale2"), p.default_format, data_types::f32)
```

Minimal Graph



Output

- Non-optimized
  - 28 28 28 28 28 28 ...
- Optimized
  - 255 255 255 255 255 255 ...
- 255 = max value of u8
- 28 = 255 * Scale

Guesses & Conclusion

Guess: Order of Quantize-Scale is flipped

- Check onednn post-op order
  - DNNL_VERBOSE=1
  - convert_dnnl_verbose.py
- No problem in post-op order
  - eltw_linear + eltw_prod
- Interestingly, there is no clip operation

Root Cause: Clip(Clamp) optimization issue

- For some reason, clip is not added to post-op
  - Has_clamp turned off in prepare_quantize_fusing pass
  - Clamp can be optimized(turned off) when
    - Output datatype = u8/i8
    - Level=256
  - In this case
    - Cldnn use type conversion instead clamp
    - Onednn does not support type conversion post-op

```
bool need_clamp = levels != 256 || out_is_fp;
bool need_min_clamp = need_clamp;
bool need_max_clamp = need_clamp;
```

Why other quantize test couldn’t catch this?

- Prev primitive = tanh activation
  - Blocked that Quantize being fused
- Level != 256
- Quant is last post-op
- …

Related codes

Cldnn round code

```
// Output clamp
if (p->has_clamp) {      Ilya Znamenskiy, 15 months ago • [GPU] Added post-ops support for OneDNN primitive…
    if (p->has_min_clamp && p->has_max_clamp)
        op_decls += "\\\n\t" + tmp_var + " = clamp(" + tmp_var + ", " + out_lo + ", " + out_hi + ");";
    else if (p->has_min_clamp)
        op_decls += "\\\n\t" + tmp_var + " = max(" + tmp_var + ", " + out_lo + ");";
    else
        op_decls += "\\\n\t" + tmp_var + " = min(" + tmp_var + ", " + out_hi + ");";
}

// Output conversion with rounding and saturation
op_decls += "\\\n\t" + GetOutputType(vec_size) + " " + out_var + " = " + ConvertToOutputTypeSat(tmp_var, vec_size) + ";";
break;
```

Fix

- program_node::init_onednn_primitive_attributes()

```
 if (q_param->has_clamp || idx < cldnn_post_ops.size() - 1) {
     float out_lo = q_param->has_min_clamp ? q_param->out_lo :
in<float>(out_dt);
     float out_hi = q_param->has_max_clamp ? q_param->out_hi :
ax<float>(out_dt);
     post_ops.append_eltwise(1.0f, dnnl::algorithm::eltwise_clip, out_lo,
```