GEMM/GEMV in MLLM

A: q8_0(BSHD), B:q4_0x4(BSHD) -> C:f32(BSHD)

A: q8_0x4(BSHD), B:q4_0x4(BSHD) -> C:f32(BSHD)

for ARMv8.2+ devices

chenghua.wang.edu@gmail.com

The Quantization Data Types in llama.cpp

Q8_0	GH	8-bit round-to-nearest quantization (`q`). Each block has 32 weights. Weight formula: `w = q * block_scale`. Legacy quantization method (not used widely as of today).
Q8_1	GH	8-bit round-to-nearest quantization (`q`). Each block has 32 weights. Weight formula: `w = q * block_scale + block_minimum`. Legacy quantization method (not used widely as of today)
Q8_K	GH	8-bit quantization (`q`). Each block has 256 weights. Only used for quantizing intermediate results. All 2-6 bit dot products are implemented for this quantization type. Weight formula: `w = q * block_scale`.
I8	GH	8-bit fixed-width integer number.
Q6_K	GH	6-bit quantization (`q`). Super-blocks with 16 blocks, each block has 16 weights. Weight formula: `w = q * block_scale(8-bit)`, resulting in 6.5625 bits-per-weight.
Q5_0	GH	5-bit round-to-nearest quantization (`q`). Each block has 32 weights. Weight formula: `w = q * block_scale`. Legacy quantization method (not used widely as of today).
Q5_1	GH	5-bit round-to-nearest quantization (`q`). Each block has 32 weights. Weight formula: `w = q * block_scale + block_minimum`. Legacy quantization method (not used widely as of today).
Q5_K	GH	5-bit quantization (`q`). Super-blocks with 8 blocks, each block has 32 weights. Weight formula: `w = q * block_scale(6-bit) + block_min(6-bit)`, resulting in 5.5 bits-per-weight.
Q4_0	GH	4-bit round-to-nearest quantization (`q`). Each block has 32 weights. Weight formula: `w = q * block_scale`. Legacy quantization method (not used widely as of today).

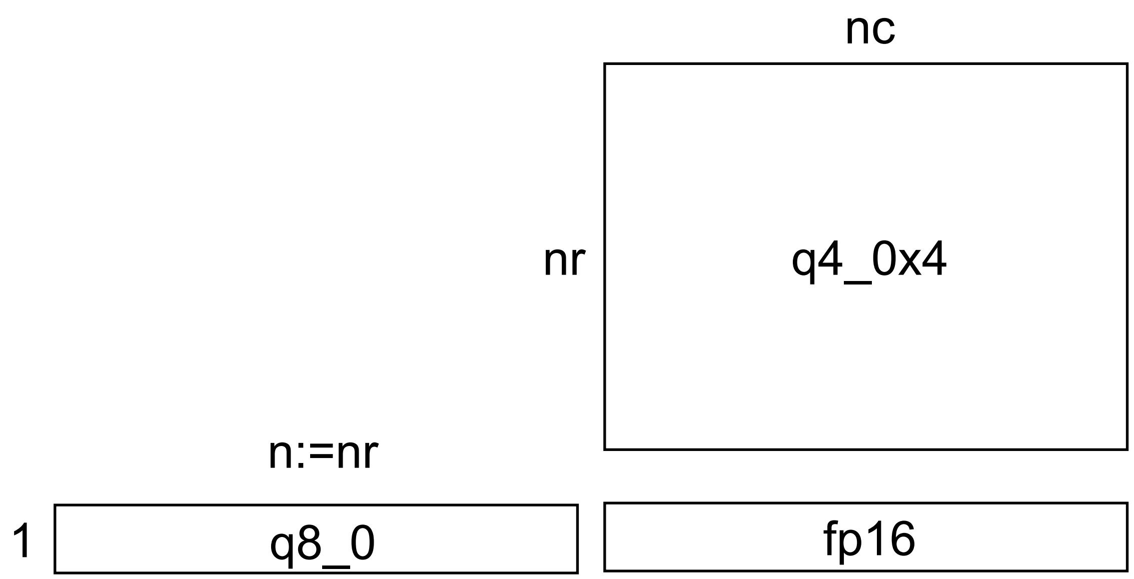
```c
typedef struct {
    mllm_fp16_t d;     // delta
    int8_t qs[QK8_0]; // quants QK8_0 = 32
} block_q8_0;
```

```c
// QK4_0 = 32
typedef struct {
    mllm_fp16_t d;          // delta
    uint8_t qs[QK4_0 / 2]; // nibbles / quants
} block_q4_0;
```

```c
typedef struct {
    mllm_fp16_t d[4];        // deltas for 4 q4_0 blocks
    uint8_t qs[QK4_0 * 2]; // nibbles / quants for 4 q4_0 blocks
} block_q4_0x4;
```

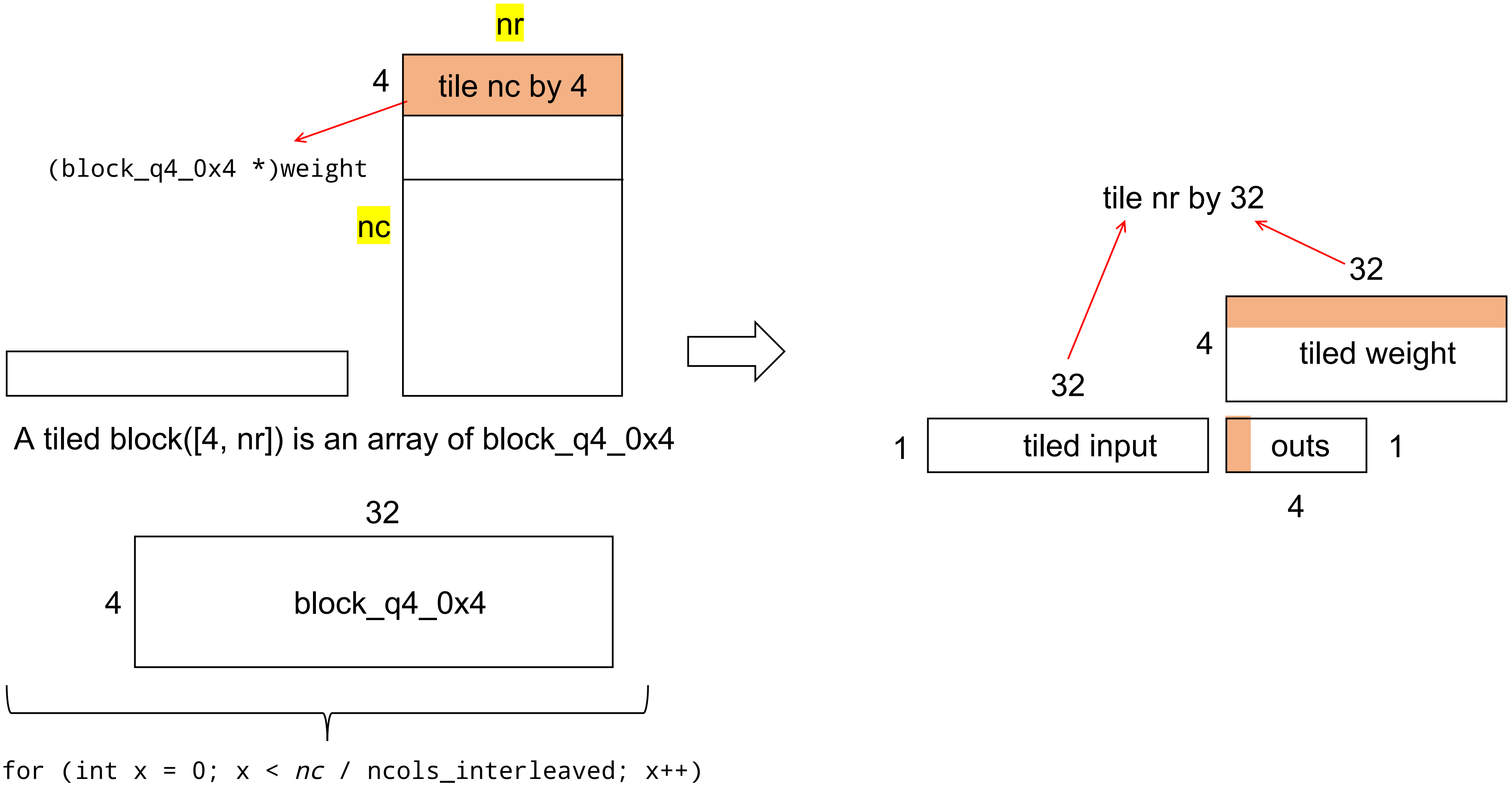
https://huggingface.co/docs/hub/en/gguf#quantization-types

The GEMV impl in llama.cpp

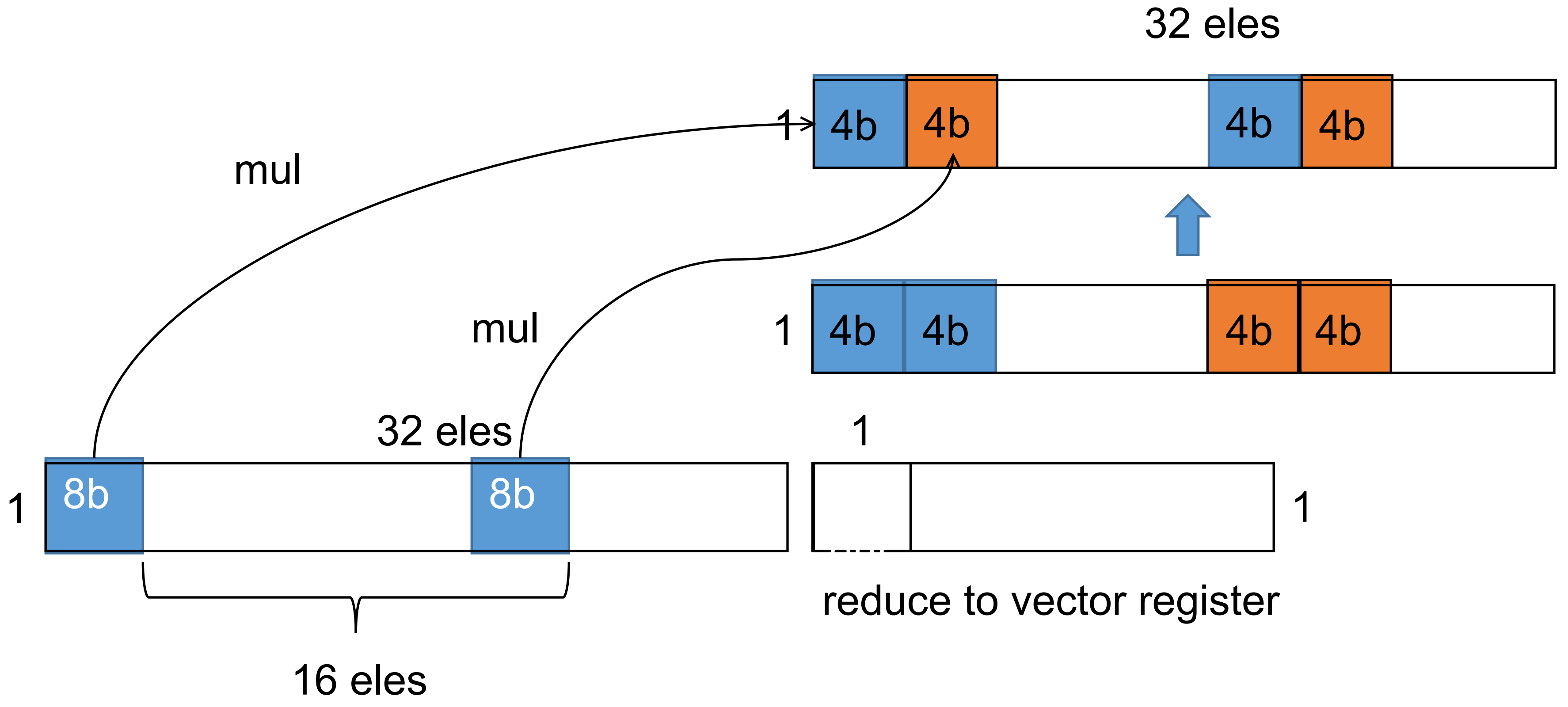


When doing V Projection. The input has shape [1, in_feature], and the weight's shape is [out_feature, in_feature].
Due to the quantization of weights is q4_0x4, we tile the column dimension of the weight matrix by 4.

The GEMV impl in llama.cpp

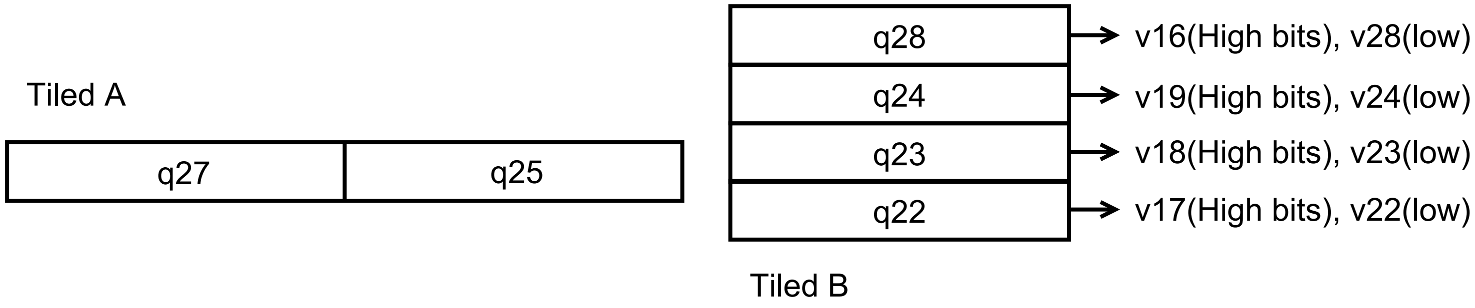


The GEMV impl in llama.cpp



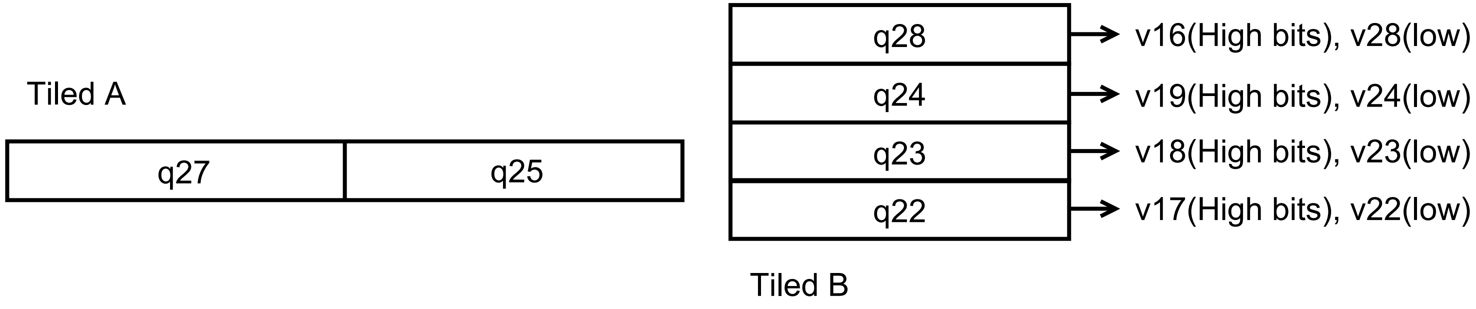
The GEMV ASM impl in llama.cpp

```
"movi v31.16b, #0x4\n"               // for sshl. to get high bits.
"movi v30.16b, #0xf0\n"              // for mask. to get low bits.
"add %x[b_ptr], %x[b_ptr], #0x8\n"   // to qs
"1:"                                 // Column loop
"add x22, %x[a_ptr], #0x2\n"         // to qs
"movi v29.16b, #0x0\n"               // acc is on register v29(16x8bits). Set to 0.
"mov x21, %x[nb]\n"                  // move num of blocks to register x21
"2:"                                 // Block loop
"ldr q28, [%x[b_ptr], #0x0]\n"       // load 128 bits from b matrix
"ldr q27, [x22, #0x0]\n"             // load 128 bits from a matrix
"movi v26.4s, #0x0\n"                // acc is on register v26(4x32bits). Set to 0.
"sub x20, x22, #0x2\n"               // to get scalar
"ldr q25, [x22, #0x10]\n"            // load 128 bits to q25. offsets is 16B
"ldr q24, [%x[b_ptr], #0x10]\n"      // load 128 bits to q24. offsets is 16B
"sub x21, x21, #0x1\n"               // nb = nb - 1
"add x22, x22, #0x22\n"              // a_ptr = aptr + 34B
"ldr q23, [%x[b_ptr], #0x20]\n"      // load 128 bits to q23. offset is 32
"ldr q22, [%x[b_ptr], #0x30]\n"      // load 128 bits to q22. offset is 48
"ld1r { v21.8h }, [x20]\n"           // scalar 4x16bit
"ldr q20, [%x[b_ptr], #-0x8]\n"      // scalar 1x16bit
"sshl v16.16b, v28.16b, v31.16b\n"   // get high bits in q4_0x4
"and v28.16b, v28.16b, v30.16b\n"    // get low bits in q4_0x4
```



The GEMV ASM impl in llama.cpp

```
"sshl v19.16b, v24.16b, v31.16b\n"  // get high bits in q4_0x4
"and v24.16b, v24.16b, v30.16b\n"   // get low bits in q4_0x4
"add %x[b_ptr], %x[b_ptr], #0x48\n" // b_ptr = b_ptr + 72
"sshl v18.16b, v23.16b, v31.16b\n"  // get high bits in q4_0x4
"and v23.16b, v23.16b, v30.16b\n"   // get low bits in q4_0x4
".inst 0x4f9be21a  // sdot v26.4s, v16.16b, v27.4b[0]\n"
"sshl v17.16b, v22.16b, v31.16b\n" // get high bits in q4_0x4
"and v22.16b, v22.16b, v30.16b\n"  // get low bits in q4_0x4
"fcvtl v21.4s, v21.4h\n"           // cvt 8x16b to 4x32b, scalar of a matrix
"fcvtl v16.4s, v20.4h\n"           // cvt 8x16b to 4x32b, scalar of b matrix. reuse v16 register
".inst 0x4f99e39a  // sdot v26.4s, v28.16b, v25.4b[0]\n"
"fmul v16.4s, v16.4s, v21.4s\n"                              // v16 = v16 * v21, scalar a * scalar b
".inst 0x4fbbe27a  // sdot v26.4s, v19.16b, v27.4b[1]\n" // v19(8 bits) + v27(32bit, 1B) to v26(32bit)
".inst 0x4fb9e31a  // sdot v26.4s, v24.16b, v25.4b[1]\n"
".inst 0x4f9bea5a  // sdot v26.4s, v18.16b, v27.4b[2]\n"
".inst 0x4f99eafa  // sdot v26.4s, v23.16b, v25.4b[2]\n"
".inst 0x4fbbea3a  // sdot v26.4s, v17.16b, v27.4b[3]\n"
".inst 0x4fb9eada  // sdot v26.4s, v22.16b, v25.4b[3]\n"
"scvtf v26.4s, v26.4s, #0x4\n"          // cvt int to float. the #0x4 is scale factor
"fmla v29.4s, v26.4s, v16.4s\n"         // v29 = v26 * v16 + v29
"cbnz x21, 2b\n"                        // is x21 is not zero, jmp to label 2. num block loop.
"sub %x[nc], %x[nc], #0x4\n"            // sub col by 4
"str q29, [%x[res_ptr], #0x0]\n"        // store value to res_ptr
"add %x[res_ptr], %x[res_ptr], #0x10\n" // res_ptr move 16B. 4xf32.
"cbnz %x[nc], 1b\n"                     // if nc is not zero, jump to label 1. num col loop.
```



The GEMM impl in llama.cpp