Vulkanised 2026
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

Enhancing Usability of Coop Matrices by Vector-Matrix Conversions.

Amir Momeni, Qualcomm

Cooperative Matrix Extension

- Clean, efficient, concise code for simple cases

```
for (step = 0; step < TOTAL_K; step += TILE_K)
{
  subMatrixAStart = row * STRIDE_A + step;
  subMatrixBStart = col * STRIDE_B + step;

  coopMatLoad(matA, inputA, subMatrixAStart, STRIDE_A, ROW_FIRST); // Global to CoopMat
  coopMatLoad(matB, inputB, subMatrixBStart, STRIDE_B, COL_FIRST); // Global to CoopMat

  matC = coopMatMulAdd(matA, matB, matC);
}
```

- Limitations
  - Convolution
  - Large Language Models

Convolution

- Requires im2col
- Explicit in Global Memory (not efficient)
- Implicit on the path from Global to Shared; suggested CoopMat code below

Problems:
- Imposition on the application to use Shared Memory
- Direct load from Global CoopMat can be more efficient

```
for (step = 0; step < TOTAL_K; step += TILE_K)
  for (filter_row = 0; filter_row < FILTER_HEIGHT; filter_row++)
    for (filter_col = 0; filter_col < FILTER_WIDTH; filter_col++)
    {
      // Load A matrix data into Shared Memory
      // Loading global to shared can be complex
      //  --> coordinate calculation for im2col transformation
      //  --> handle stride, dilation
      //  --> zero padding for tensor oob handling

      subMatrixAStart = row * STRIDE_A + step;
      subMatrixBStart = (col + filter_col + filter_row*FILTER_WIDTH) * STRIDE_B + step;

      coopMatLoad(matA, sharedA, subMatrixAStart, STRIDE_A, ROW_FIRST); // Shared to CoopMat
      coopMatLoad(matB, inputB,  subMatrixBStart, STRIDE_B, COL_FIRST); // Buffer to CoopMat

      matC = coopMatMulAdd(matA, matB, matC);
    }
```

Data Conversion

Problems:
- Inefficient interop between vector and matrix processing units
- Vector to Opaque transform imposes passing through Shared Memory

- How to convert input data prior to sending it to matrix multiply hardware
- Suggested CoopMat code below (unpack weights from 4bit to FP16)

```
for (step = 0; step < TOTAL_K; step += TILE_K)
{
  subMatrixAStart = row * STRIDE_A + step;
  subMatrixBStart = col * STRIDE_B + step;

  // Pre-Process input data for B matrix
  // 1. Load B data from Global Buffer to Function
  // 2. Convert format, unpack as needed using the ALU
  // 3. Move B data to Shared Memory

  coopMatLoad(matA, inputA,  subMatrixAStart, STRIDE_A, ROW_FIRST); // Buffer to Function
  coopMatLoad(matB, sharedB, subMatrixBStart, STRIDE_B, COL_FIRST); // Shared to Function

  matC = coopMatMulAdd(matA, matB, matC);
}
```

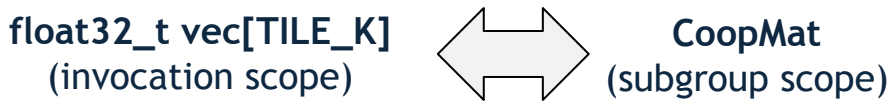
Global □ Function

Function □ Shared

Shared □ CoopMat

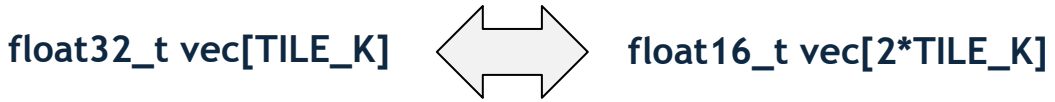
QCOM_cooperative_matrix_conversion

- vectorToCoopmatQCOM
- coopmatToVectorQCOM



- extractSubArrayQCOM

- bitcastQCOM



VectorToCoopmatQCOM

```
vectorToCoopmatQCOM(SRC_TYPE vec[SVecLen],
                    coopmat<DST_TYPE, gl_ScopeSubgroup, M, K, CoopMatUse> coopmat);
```

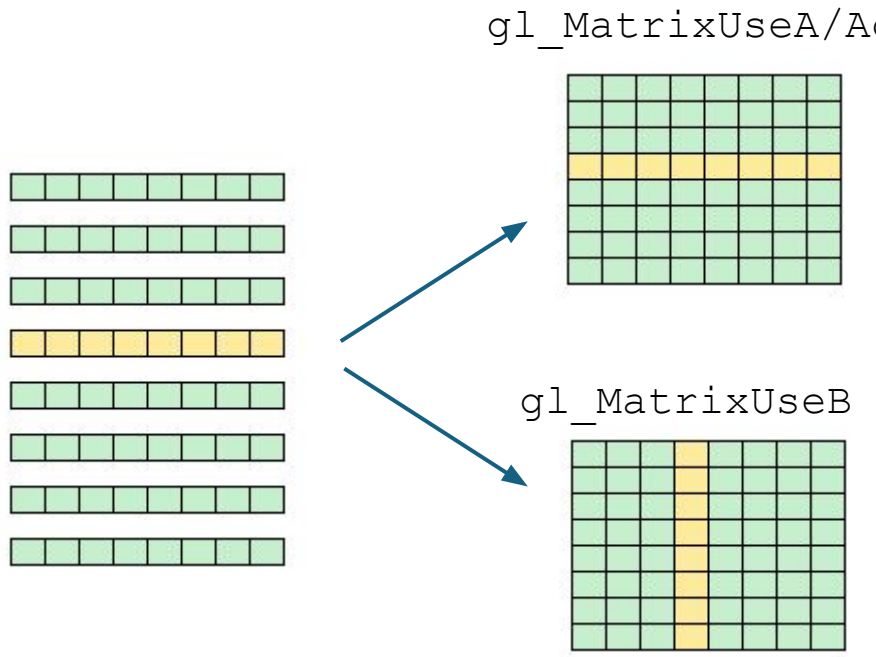
- SRC_TYPE must be uint32_t or the same as DST_TYPE
- DST_TYPE is one of float32_t, float16_t, int8_t, and uint8_t
- SVecLen is K if SRC_TYPE == DST_TYPE
- SVecLen is K/NUM_PACK if SRC_TYPE is uint32_t;
  NUM_PACK = BitWidth(uint32_t)/BitWidth(DST_TYPE)
- M is the number of rows
- K is the number of columns

CoopMatUse : gl_MatrixUseA/Acc

The vector from the ith invocation goes to the ith **row** of the resulting matrix

CoopMatUse : gl_MatrixUseB

The vector from the ith invocation goes to the ith **column** of the resulting matrix



VectorToCoopmatQCOM

F32_example

```
coopmat<float32_t, gl_ScopeSubgroup, 64, 8, gl_MatrixUseA> f32_matA;
float32_t fvec[8];
vectorToCoopmatQCOM(fvec, f32_matA);
```

INT8_example

```
coopmat<int8_t, gl_ScopeSubgroup, 64, 32, gl_MatrixUseA> si8_matA;
int8_t sivec[32];
vectorToCoopmatQCOM(sivec, si8_matA);

uint32_t uivec[8];
vectorToCoopmatQCOM(uivec, si8_matA);
```

coopmatToVectorQCOM

Extracts the ith row or ith column (depending on CoopMatUse) of the cooperative matrix to the invocation with gl_SubgroupInvocationID == i.

```
coopmatToVectorQCOM(coopmat<SRC_TYPE, gl_ScopeSubgroup, M, K, CoopMatUse> coopmat,
                    DST_TYPE vec[DVecLen]);
```

- SRC_TYPE is one of float32_t, float16_t, int8_t, and uint8_t
- DST_TYPE must be uint32_t or the same as ELT_TYPE
- DVecLen is K if DST_TYPE == SRC_TYPE
- DVecLen is K/NUM_PACK if SRC_TYPE is uint32_t;
  NUM_PACK = BitWidth(uint32_t)/BitWidth(DST_TYPE)
- M is the number of rows
- K is the number of columns

extractSubArrayQCOM

Extracts a subarray from a source array. If the result is to be used as inputs to any subsequent cooperative matrix operations, the source array must be one of NSize variants supported by the VkCooperativeMatrixPropertiesKHR device query and index must be a multiple of one of the lengths supported for input channel K.

```
extractSubArrayQCOM( uint32_t SArr[NSize], int index,  uint32_t DArr[KSize]);
extractSubArrayQCOM(  int32_t SArr[NSize], int index,   int32_t DArr[KSize]);
extractSubArrayQCOM(float32_t SArr[NSize], int index, float32_t DArr[KSize]);
extractSubArrayQCOM(float16_t SArr[NSize], int index, float16_t DArr[KSize]);
```

Pseudocode:

```
 for (j in 0…KSize-1 ) DArr[j] = SArr[index+j];
```

Example

```
float32_t uvecAcc[32];
float32_t uvecA[8];
extractSubArrayQCOM(uvecAcc, 0, uvecA);
…
extractSubArrayQCOM(uvecAcc, 24, uvecA);
```

bitcastQCOM

Bit casts an input source array to an output destination array.

```
bitcastQCOM(SrcTy SArr[SrcLen], DstTy DArr[DstLen]);
```

- SArr is an array variable with SrcTy as the element type and SrcLen as length
- DArr is an array variable with DstTy as the element type and DstLen as length
- SrcTy and DstTy are one of int32_t, uint32_t, float32_t, and float16_t
- BitWidth(SrcTy) * SrcLen == BitWidth(DstTy) * DstLen

Examples

```
uint32_t uvecA[8];
float    vecB[8];
bitcastQCOM(vecB, uvecA);

float16_t f16_vecB[16];
bitcastQCOM(f16_vecB, uvecA);
```

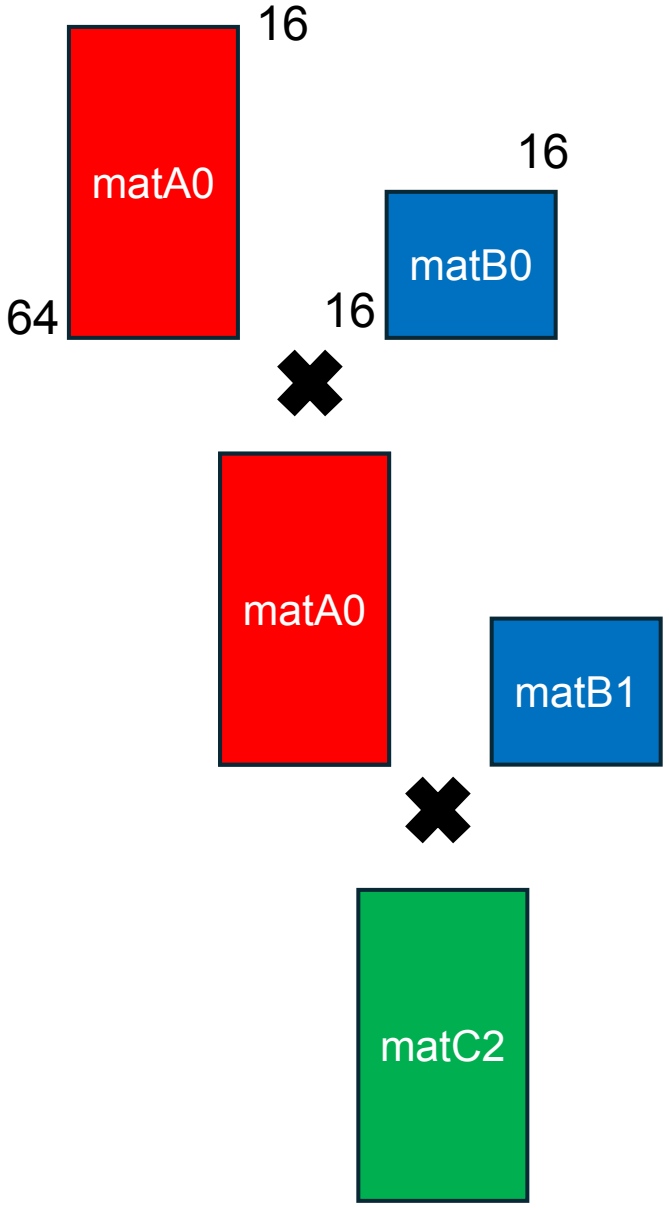
QCOM_cooperative_matrix_conversion

- Use cases
  - Convolution (tensor loading, edge zero fill or setting to constant)
  - Neural Texture Decompression
  - Layer Merging (output activation of one layer -> input activation of next layer)
  - Unpacking weights/activation (decouple load/unpack from matrix creation)

Convolution Using Implicit im2col (NHWC)

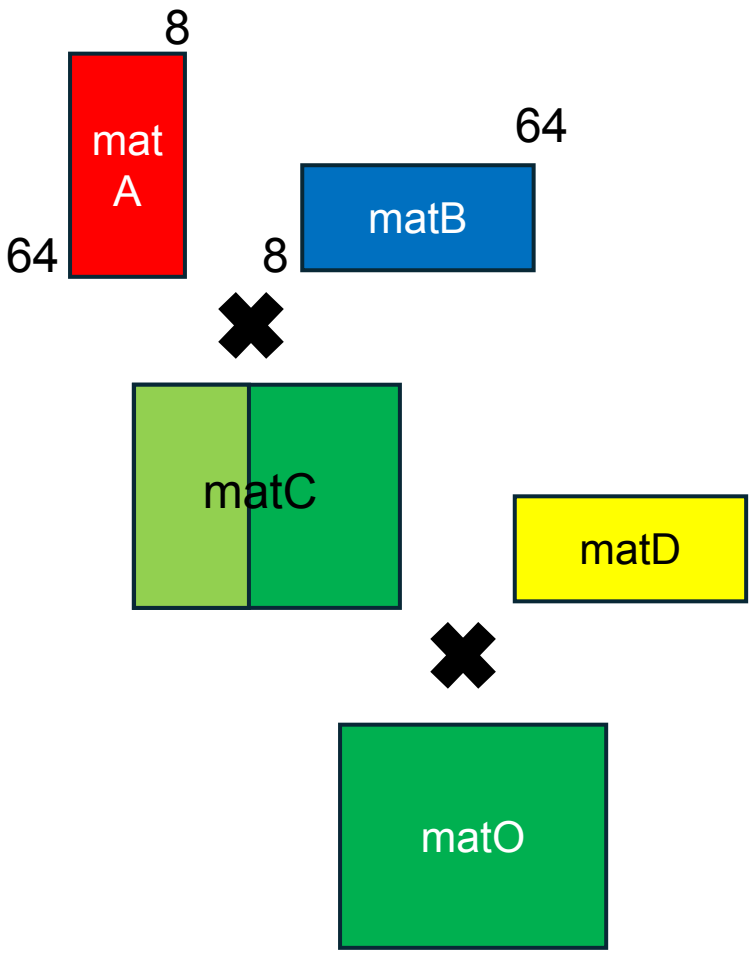
```
for (uint32_t step = 0; step < TOTAL_K; step += TILE_K)
{
    uint32_t subMatrixBStartInElements = col * FILTER_H * FILTER_W * strideBinElements + step;
    for (uint32_t filter_row = 0; filter_row < FILTER_H; filter_row++)
        for (uint32_t filter_col = 0; filter_col < FILTER_W; filter_col++)
        {
            // load matB input data using coop_mat extension
            coopmat<float, gl_ScopeSubgroup, TILE_K, TILE_N, gl_MatrixUseB> matB;
            coopMatLoad(matB, inputB, subMatrixBStartInElements, FILTER_H * FILTER_W * strideBinElements, LAYOUT_K_FIRST);

            // load vecA input data as vectors using regular vector load
            float vecA[TILE_K];
            uint32_t input_row = STRIDE * out_row + DILATION * (filter_row - FILTER_H/2);
            uint32_t input_col = STRIDE * out_col + DILATION * (filter_col - FILTER_W/2);
            for (int i=0; i<TILE_K; i++)
              vecA[i] = inputA[(input_row * INPUT_W + input_col) * strideAinElements + step + i];

            // zero fill vecA vector data for out of boundary cases
            if ((input_row < 0) || (input_row >= INPUT_H) || (input_col < 0) || (input_col >= INPUT_W))
              for (int i=0; i<TILE_K; i++) vecA[i] = float(0);

            // convert vecA to matA and perform matrix multiplication
            coopmat<float, gl_ScopeSubgroup, TILE_M, TILE_K, gl_MatrixUseA> matA;
            vectorToCoopmatQCOM(vecA, matA);
            matC = coopMatMulAdd(matA, matB, matC);

            subMatrixBStartInElements += strideBinElements;
        }
}
```

Neural Texture Decompression




```
coopmat<float16_t, gl_ScopeSubgroup, 64, 16, gl_MatrixUseA>           matA0, matA1;
coopmat<float16_t, gl_ScopeSubgroup, 16, 16, gl_MatrixUseB>           matB0, matB1;
coopmat<float16_t, gl_ScopeSubgroup, 64, 16, gl_MatrixUseAccumulator> matC1, matC2;
matC1 = coopmat<float16_t, gl_ScopeSubgroup, 64, 16, gl_MatrixUseAccumulator>(0.0);
matC2 = coopmat<float16_t, gl_ScopeSubgroup, 64, 16, gl_MatrixUseAccumulator>(0.0);

// load matB0 and matB1 matrix input data using coopmat extension
coopMatLoad(matB0, inputB0, subMatrixBStartInElements, strideBinElements, LAYOUT_K_FIRST);
coopMatLoad(matB1, inputB1, subMatrixBStartInElements, strideBinElements, LAYOUT_K_FIRST);

// load vecA0 (input features) any way you like
float16_t vecA0[16];
for (int i=0; i<16; i++) vecA0[i] = inputA.x[inputCoord + i];

// convert vecA0 to matA0, then execute MatMul (first layer)
vectorToCoopmatQCOM(vecA0, matA0);
matC1 = coopMatMulAdd(matA0, matB0, matC1);

// convert matC1 to matA1, then execute MatMul (second layer)
float16_t vecA1[16];
coopmatToVectorQCOM(matC1, vecA1);
vectorToCoopmatQCOM(vecA1, matA1);
matC2 = coopMatMulAdd(matA1, matB1, matC2);

// convert matC2 to vecC2 (which holds the output features)
float16_t vecC2[16];
coopmatToVectorQCOM(matC2, vecC2);
```

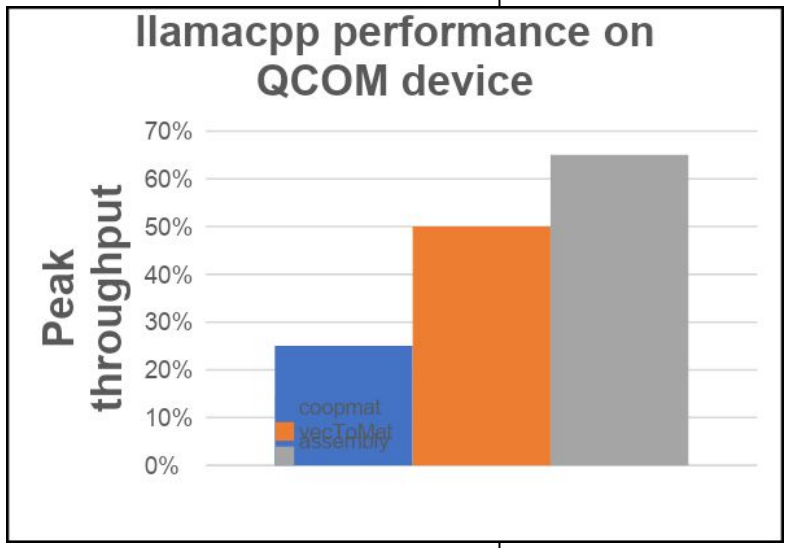
Layer Merging (Flash Attention)




```
coopmat<float, gl_ScopeSubgroup, 64,  8, gl_MatrixUseA>           matA;
coopmat<float, gl_ScopeSubgroup, 8,  64, gl_MatrixUseB>           matB;
coopmat<float, gl_ScopeSubgroup, 64, 64, gl_MatrixUseAccumulator> matC;
coopmat<float, gl_ScopeSubgroup, 64,  8, gl_MatrixUseA>           matCtoA;
coopmat<float, gl_ScopeSubgroup, 8,  64, gl_MatrixUseB>           matD;
coopmat<float, gl_ScopeSubgroup, 64, 64, gl_MatrixUseAccumulator> matO;

for (uint stepN = 0; stepN < TOTAL_N; stepN += TILE_N64)
{
    for (uint stepK = 0; stepK < TOTAL_K; stepK + =TILE_K8) {
        coopMatLoad(matA, inputA, subMatrixAStart, STRIDE_A, LAYOUT_A);
        coopMatLoad(matB, inputB, subMatrixBStart, STRIDE_B, LAYOUT_B);
        matC = coopMatMulAdd(matA, matB, matC);
    }

    float vecC[64];
    coopmatToVectorQCOM(matC, vecC);
    vecC = online_softmax(vecC); // apply softmax on vecC using rowsum and rowmax
reduction operations

    coopMatLoad(matD, inputD, subMatrixDStart + 0*STRIDE_D, STRIDE_D, LAYOUT_D);
    float vecCtoA[8];
    extractSubArrayQCOM(vecC,  0, vecCtoA); // first 8 elements of matC
    vectorToCoopmatQCOM(vecCtoA,  matCtoA);
    matO = coopMatMulAdd(matCtoA, matD, matO);
    ....
    coopMatLoad(matD, inputD, subMatrixDStart + 7*STRIDE_D, STRIDE_D, LAYOUT_D);
    extractSubArrayQCOM(vecC, 56, vecCtoA); // last 8 elements of matC
    vectorToCoopmatQCOM(vecCtoA, matCtoA);
    matO = coopMatMulAdd(matCtoUseA, matD, matO);
}
```

Llamacpp q4 weight dequantization

```
coopmat <float16_t, gl_ScopeSubgroup, TILE_M64, TILE_K16, gl_MatrixUseA>           matA;
coopmat <float16_t, gl_ScopeSubgroup, TILE_K16, TILE_N64, gl_MatrixUseB>           matB;
coopmat <float16_t, gl_ScopeSubgroup, TILE_M64, TILE_N64, gl_MatrixUseAccumulator> matC;

float16_t vecAh_block[32], vecAh[TILE_K16], vecBh[TILE_K16];
for (uint step = 0; step < TOTAL_K; step += TILE_K16)
{
    for (int k=0; k<TILE_K16; k++)
        vecBh[k] = float16_t(data_bf[(pos_b + step) + (gl_SubgroupInvocationID * p.stride_b) + k]);

    if (step%32 == 0){
        const uint ib = (pos_a + step)/32 + gl_SubgroupInvocationID * p.stride_a/32;
        float d = float(data_a_packed16[ib].d);
        for (uint i = 0; i < 4; i++) {
            uint vui = uint(data_a_packed16[ib].qs[2 * i]) | (uint(data_a_packed16[ib].qs[2 * i + 1]) << 16);
            const vec4 v0 = (vec4(unpack8((vui >> 0) & 0x0F0F0F0F)) - 8.0f) * d;
            const vec4 v1 = (vec4(unpack8((vui >> 4) & 0x0F0F0F0F)) - 8.0f) * d;
            vecAh_block[4*i+ 0] = float16_t(v0.x);
            vecAh_block[4*i+ 1] = float16_t(v0.y);
            vecAh_block[4*i+ 2] = float16_t(v0.z);
            vecAh_block[4*i+ 3] = float16_t(v0.w);
            vecAh_block[4*i+16] = float16_t(v1.x);
            vecAh_block[4*i+17] = float16_t(v1.y);
            vecAh_block[4*i+18] = float16_t(v1.z);
            vecAh_block[4*i+19] = float16_t(v1.w);
        }
    }

    extractSubArrayQCOM(vecAh_block, step%32, vecAh); // extract first 16 or second 16 channels
    vectorToCoopmatQCOM(vecAh, matA);
    vectorToCoopmatQCOM(vecBh, matB);
    matC = coopMatMulAdd(matA, matB, matC);
}
```



Links to the extension

- VK_QCOM_cooperative_matrix_conversion
- GLSL_QCOM_cooperative_matrix_conversion
- SPV_QCOM_cooperative_matrix_conversion

Questions
?