Accelerating AI Models with Andes Matrix Multiplication (AMM) from CNN to LLM

Pei-Hsiang Hung,
Chung-Hua Yen,
I-Wei Wu.

Andes Technology

Outline

- Introduction to Andes Matrix Multiplication (AMM)
  - Illustrations of AMM Scalability.
- AMM Code Generation in IREE for LLM deployment
  - Choosing Optimal Tiling Size
  - Generating VLEN-Agnostic code
  - Handling LLM Prefill and Decode Stages.
- Performance Estimates
- Conclusion

Taking RISC-V® Mainstream

AMM Introduction

- The Andes Matrix Multiplication(AMM) is being designed to optimize tiled matrix multiplication.
- $M \times N$ tile = AMM($M \times K$ tile, $K \times N$ tile)
- Key features:
  - The tiles are stored in the RVV vector registers
  - 2D load and store instructions facilitate data movement between memory and vector registers.
  - The scalability across RVV VLEN, LMUL and SEW.
- Understanding Tiling Size M, N and K: (fully tiled cases)
  1. *M* is always 2.
  2. *N* equals VLEN / 64.
  3. *K* is determined by LMUL and SEW.

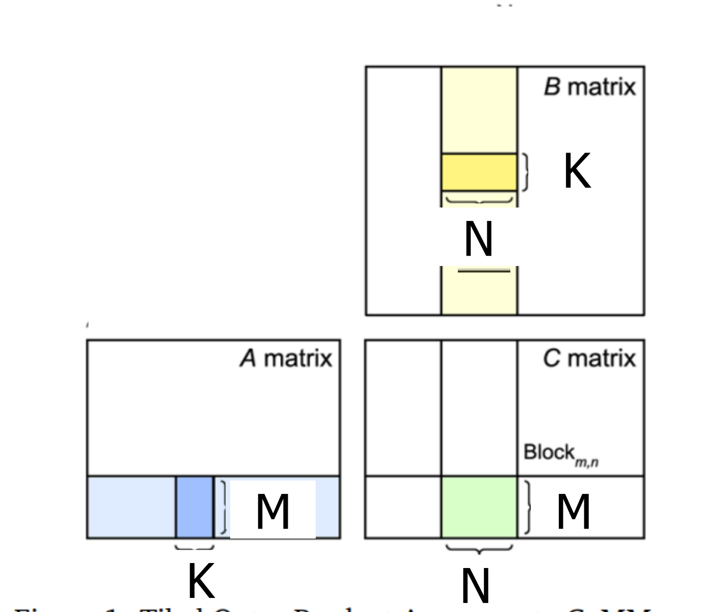
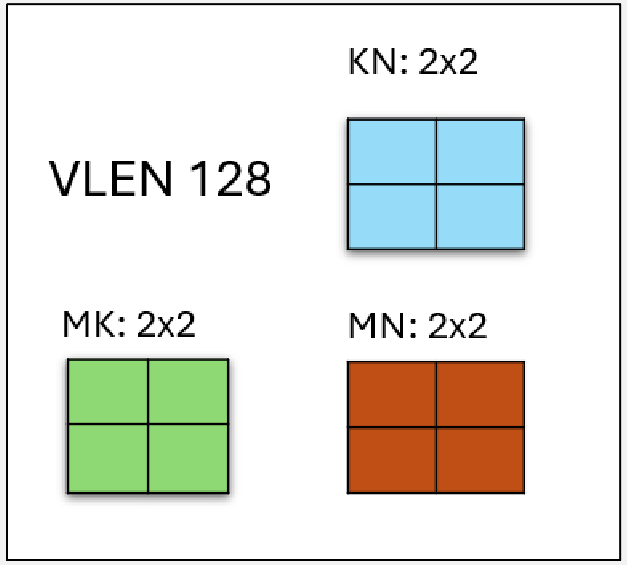


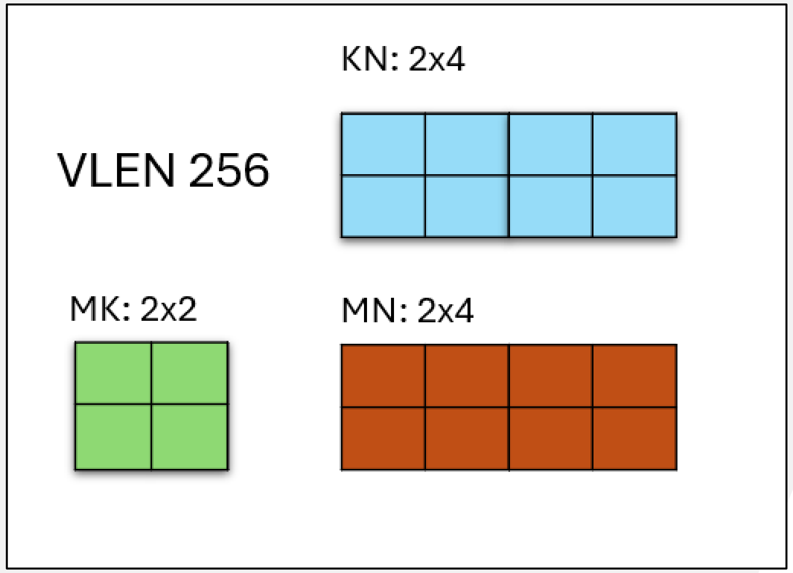
Figure 1: Tiled Outer Product Approach to GeMMs

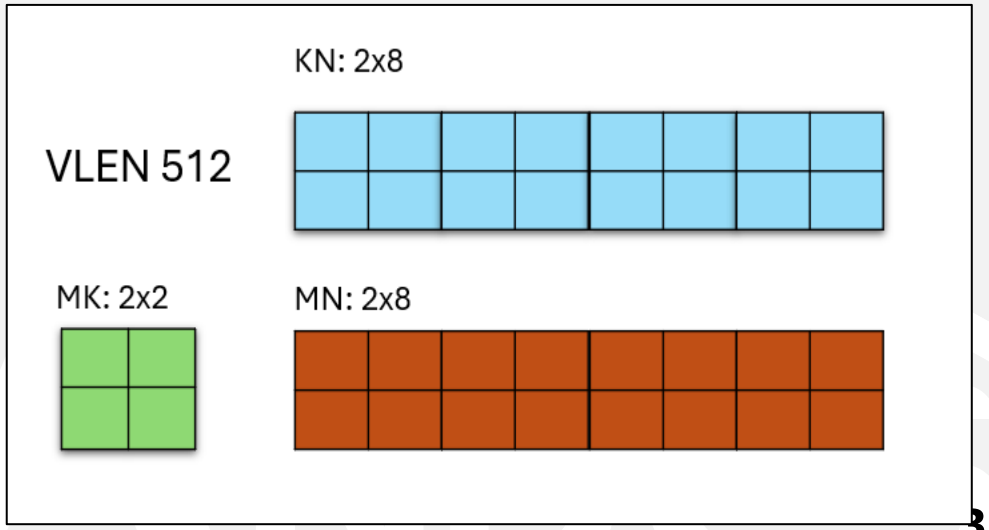
VLEN-Scalable Design

- VLEN (Vector Length) depends on the specific VPU implementation
- Conditions for the illustration below:
  - F32 * F32 -> F32
  - LMUL 1

VLEN	*M*	*N*	*K*				
			SEW \ LMUL	1	2	4	8
128	2	2	I8->I32	8	16	32	64
256	2	4	F16->F32	4	8	16	32
512	2	8	F32->F32	2	4	8	16
1024	2	16					



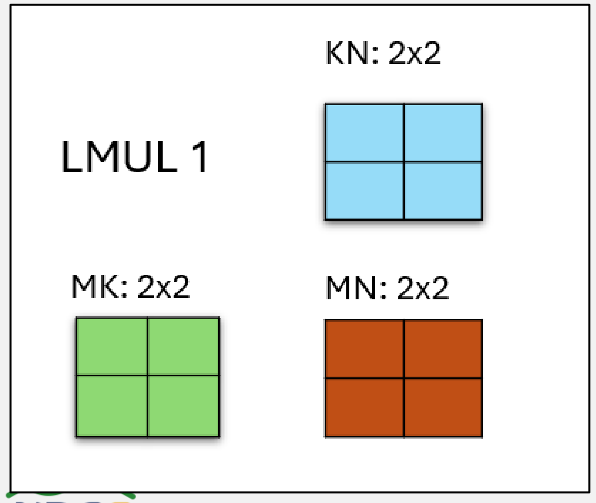


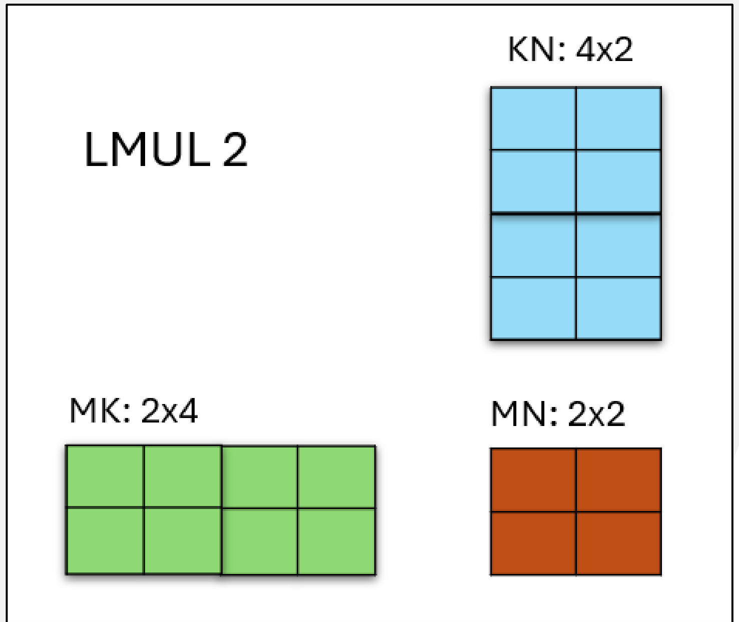


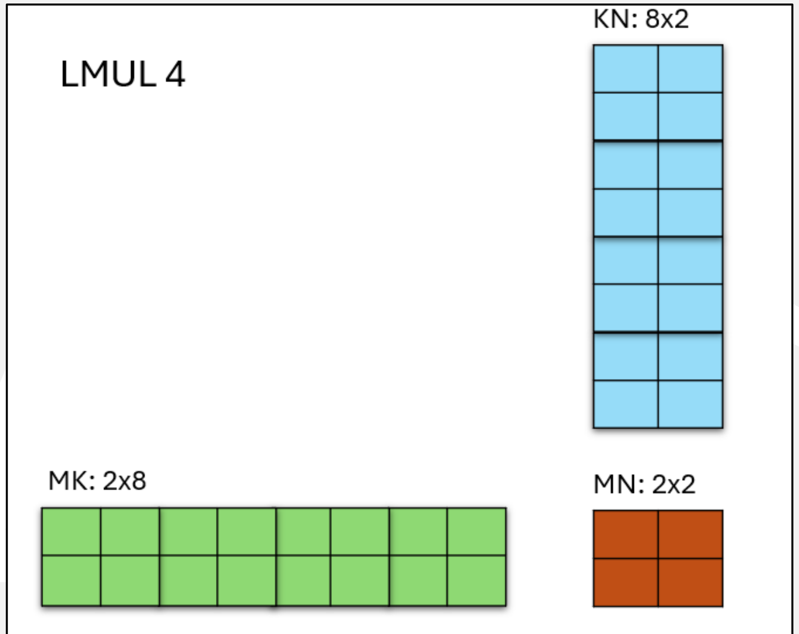
LMUL-Scalable Design

- LMUL(Vector Length Multiplier)
- AMM supports the integer LMUL values.
  - Fractional LMULs are not supported; boundary control is used instead.
- Conditions for the illustration below:
  - F32 * F32 -> F32
  - VLEN 128

VLEN	*M*	*N*	*K*				
			SEW \ LMUL	1	2	4	8
128	2	2	I8->I32	8	16	32	64
256	2	4	F16->F32	4	8	16	32
512	2	8	F32->F32	2	4	8	16
1024	2	16					



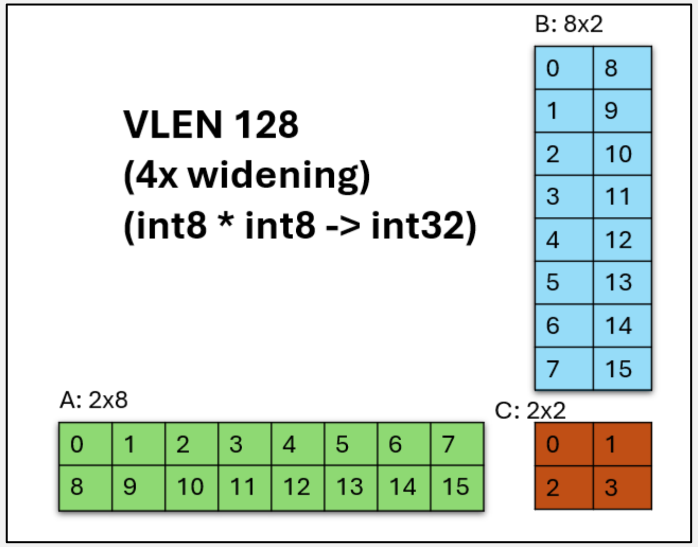


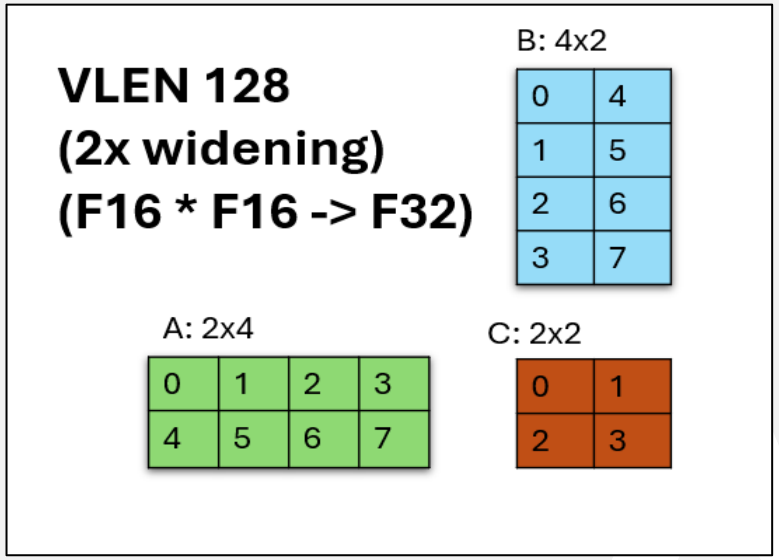


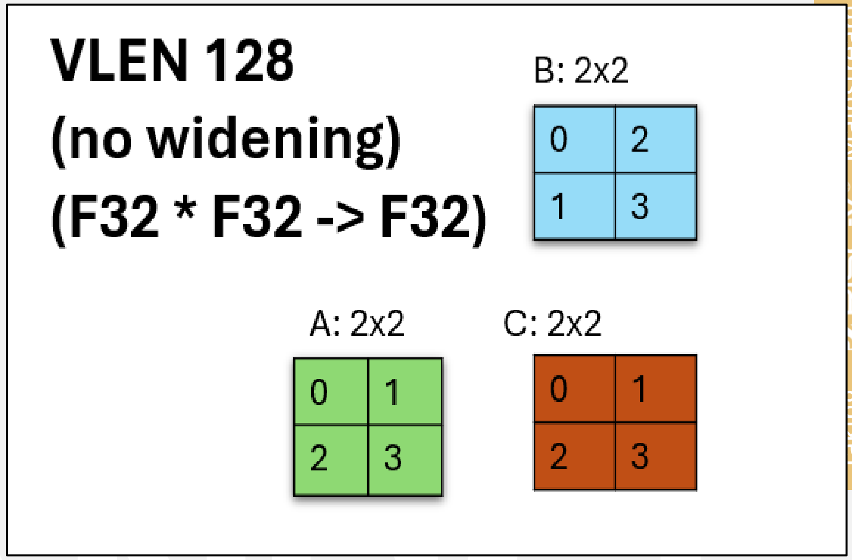
SEW-Scalable Design

- SEW(Selected Element Width)
- The results could 2x and 4x data widening
- Conditions for the illustration below:
  - VLEN 128
  - LMUL 1

VLEN	*M*	*N*	*K*				
			SEW \ LMUL	1	2	4	8
128	2	2	I8->I32	8	16	32	64
256	2	4	F16->F32	4	8	16	32
512	2	8	F32->F32	2	4	8	16
1024	2	16					







Tiled Load and Store Instructions

- 2D Load instructions
  - Load the MK tiles in row-major order.
  - Load the KN tiles in either **row-major** or **column-major** order.
    - Column-major order is helpful when the weight matrix is not stored in a transposed data layout.
  - Load the MN tiles in row-major order.
- 2D Store instructions
  - Store the MN tiles in row-major order.

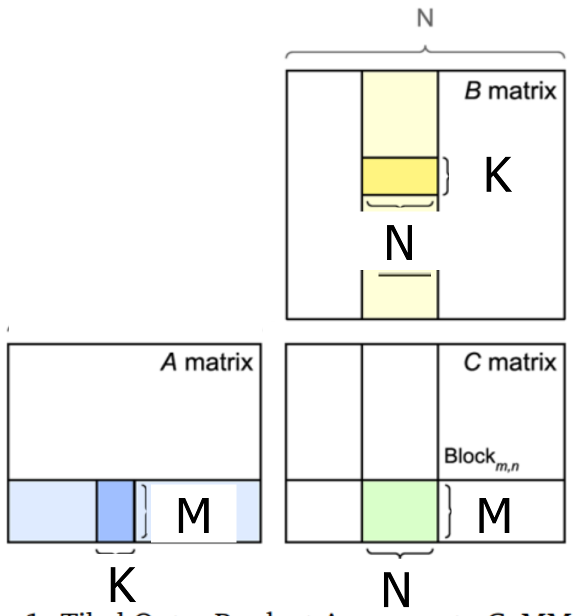
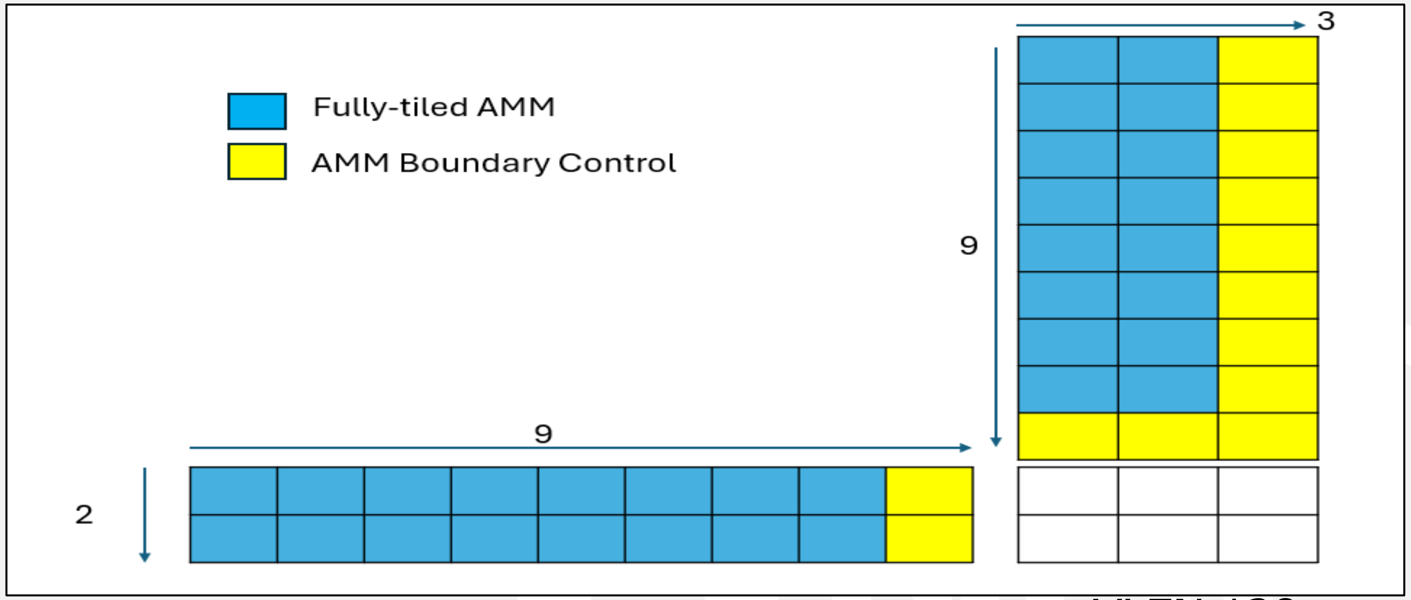


Figure 1: Tiled Outer Product Approach to GeMMs

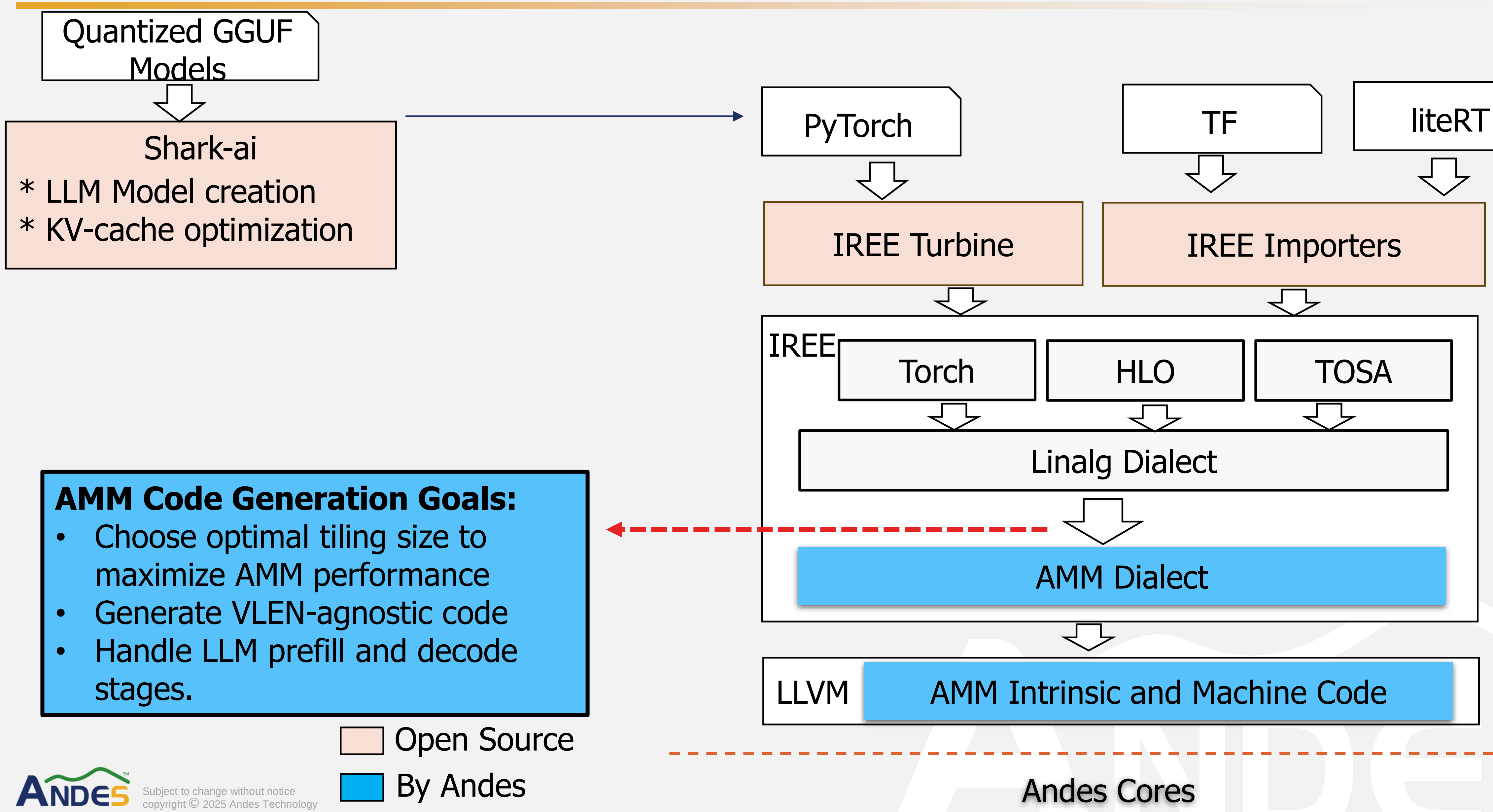
Boundary Control for the Remaining Elements

- What happens when the input shapes are not multiples of the AMM tile sizes?
- Boundary Control is managed by setting a Control and Status Register(CSR) to specify the number of active elements.
  - The main portion (blue) is computed using standard, fully tiled AMM instructions.
  - The remaining edges (yellow) are computed using AMM boundary control features, processing partial tiles."



VLEN 128

LLM Deployment Workflow With IREE



AMM Code Generation Goals:

- Choose optimal tiling size to maximize AMM performance
- Generate VLEN-agnostic code
- Handle LLM prefill and decode stages.

🟥 Open Source

🟦 By Andes

Determining Optimal Tiling Size (M, N, K) (Register level)

- **Compiler factors to decide for tiling:**
  - LMUL (1, 2, 4 or 8)
  - UNROLL_MK (Power of 2) and UNROLL_KN (1 or 2)

- **Optimization Goal**

  Maximize operations per iteration,
  subject to using <= 32 vector registers.

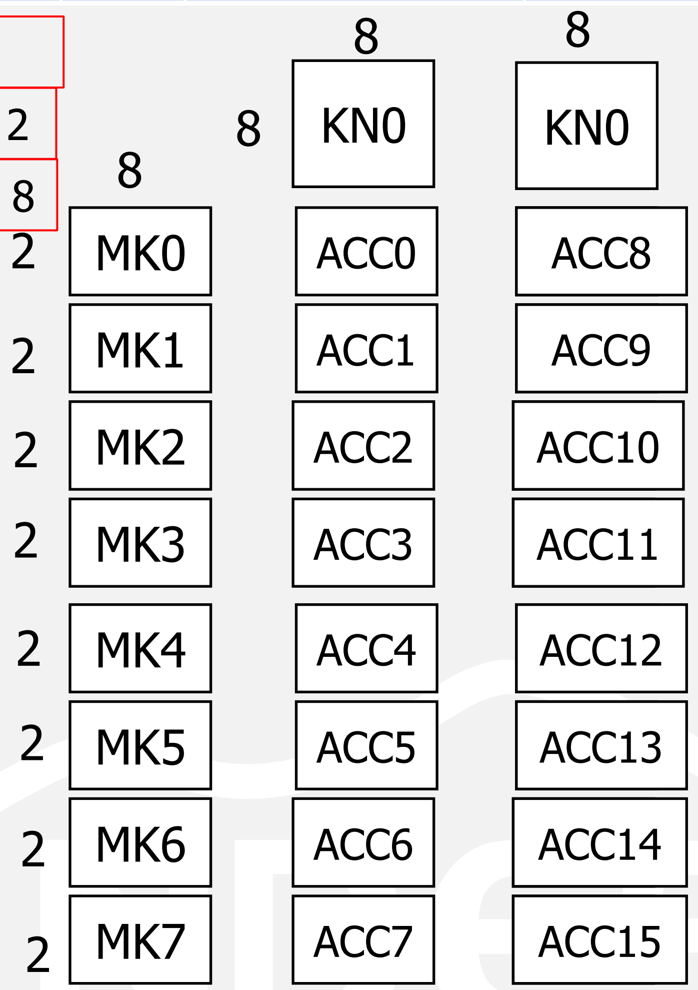
- **Note:**
  - The actual tiling size depends on the input shape and the cost model for AMM.

M	N	K	Used vector registers	Num of Operation
16	16	8	32	2048

LMUL= 4

UNROLL_KN = 2

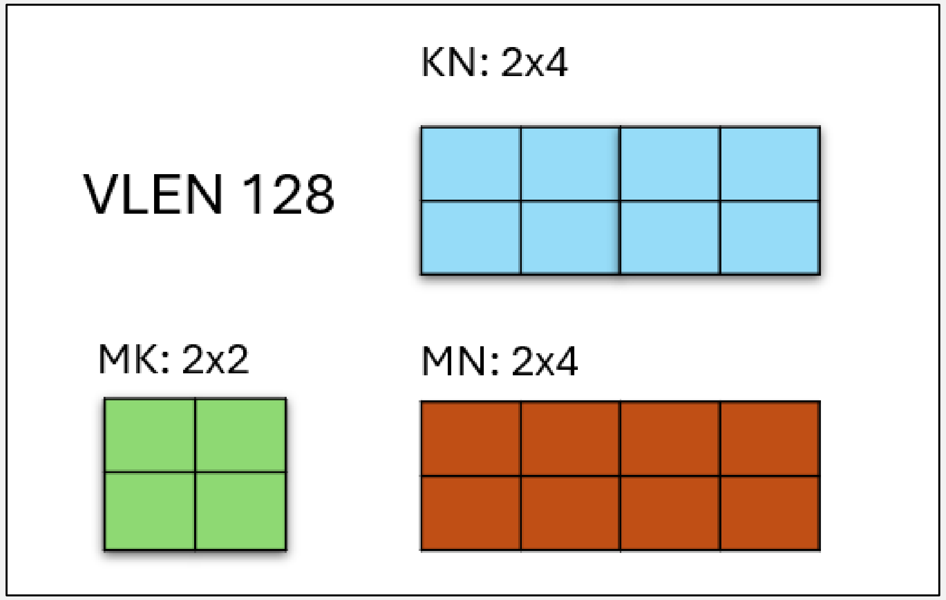
UNROLL_MK = 8

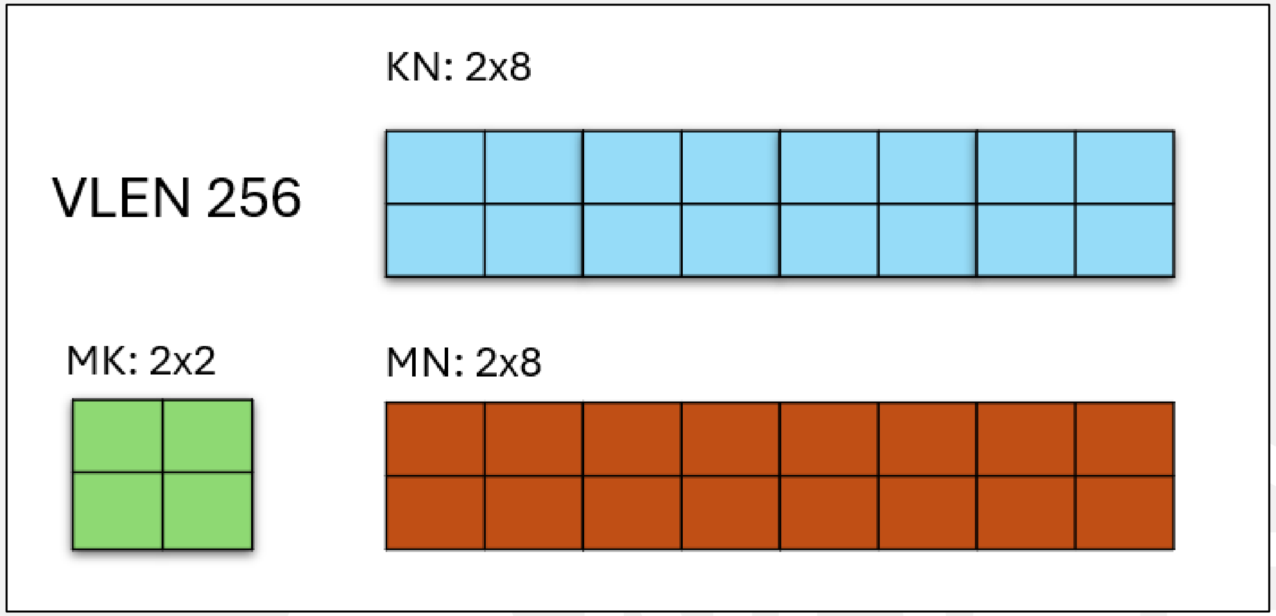


F32*F32->F32,
VLEN 512

Generate VLEN-Agnostic Code

- The tiling size of the previous example: M, N, K = [16, 16, 8]
  - This tiling size is only suitable for VLEN 512, not VLEN-scalable.
- VLEN-agnostic tiling size:
  - M, N, K = [16, [2], 8], where [2] signifies 2 × vscale.
  - Vscale is a runtime constant adaptable to different VLEN values. (vscale = VLEN/64)





LLM Prefill and Decode Stages

- Prefill stage:
  - Features a dynamic shape M, representing the token sequence length.
  - Compiler generates two kernels: an AMM kernel and an optimized RVV kernel.
    - The AMM kernel handles the majority of the computations.
    - The RVV kernel is responsible for the remaining rows when M is not a multiple of the 16.

- Decode stage:
  - The matrix multiplication shape has M equal to 1.
  - For this GEMV(Matrix-Vector multiplication) case, the compiler only generates optimized RVV code because it's more efficient than AMM.

Performance Comparison: AMM vs. RVV

- Models:
  - CNN: Mobilenet V1 (INT8*INT8->INT32)
  - LLM:tinyllama-1.1b-chat-v1.0.Q8_0.gguf. (F32*F32->F32)
- Performance estimation:
  - AMM configuration: VLEN: 512, DLEN: 512
    - 128 F32 MACs or 512 INT8 MACs
  - RVV configuration, VLEN: 512, DLEN: 512
    - 16 F32 MACs or 64 INT8 MACs
- Performance estimates: AMM speedup over RVV.

Op/Models	AMM over RVV	
	Speedup	AMM MAC efficiency
GEMM	6.4x	80%
Mobilenet V1	2.72x	34%
Tinyllama Prefill Stage	2.02x	25%

- Estimation doesn't model the complete cache behavior.

Conclusion

- The Andes Matrix Multiplication(AMM) is being designed to optimize the tiled matrix multiplication.
  - Key features are the reuse of RVV vector registers, scalability and the tiled load and store instructions.
- We have integrated IREE framework to generate optimal and scalable code for AMM.
- AMM demonstrates significant performance speedups for GEMM(up to 6.4x).
- We will continue optimizing the non-linear operators to improve the AMM speedup for the full models.

Taking RISC-V® Mainstream

Thank you!!