YAKUP KORAY BUDANAZ

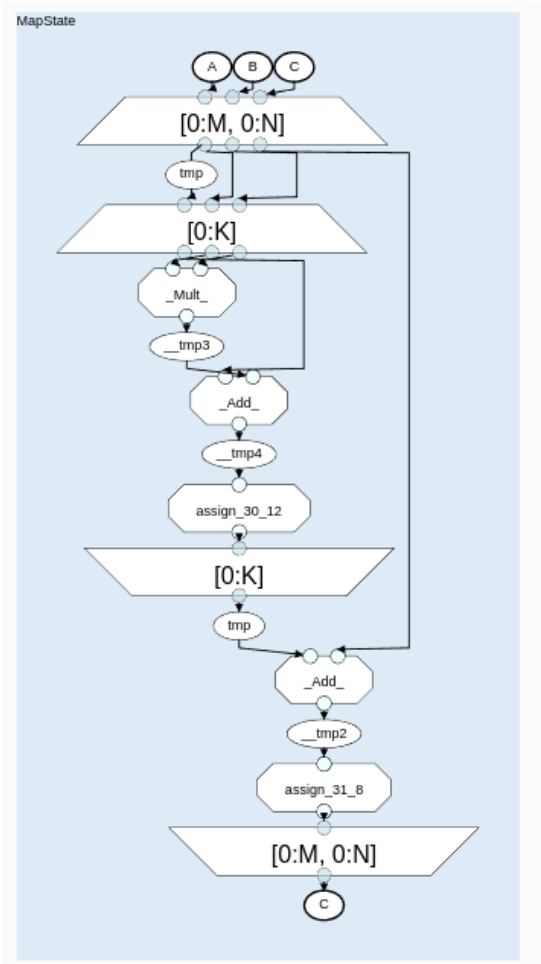
SoftHier Year 1 Sync

Overiew of the Topics:

- Tiling / Schedule Transformations for SoftHier and Ascend
- Layout Transformations
- Code-Generation and Results

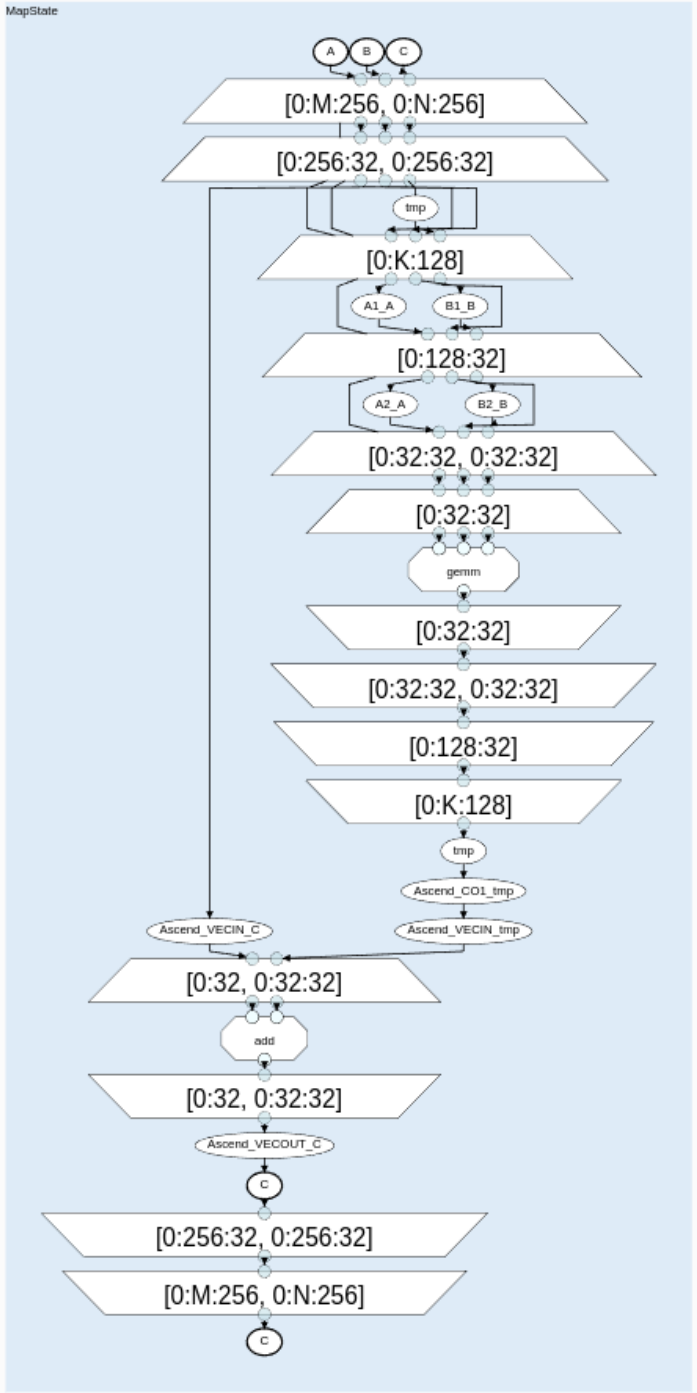
- **Tiling / Schedule Transformations for SoftHier and Ascend**

Tiling Transformations:

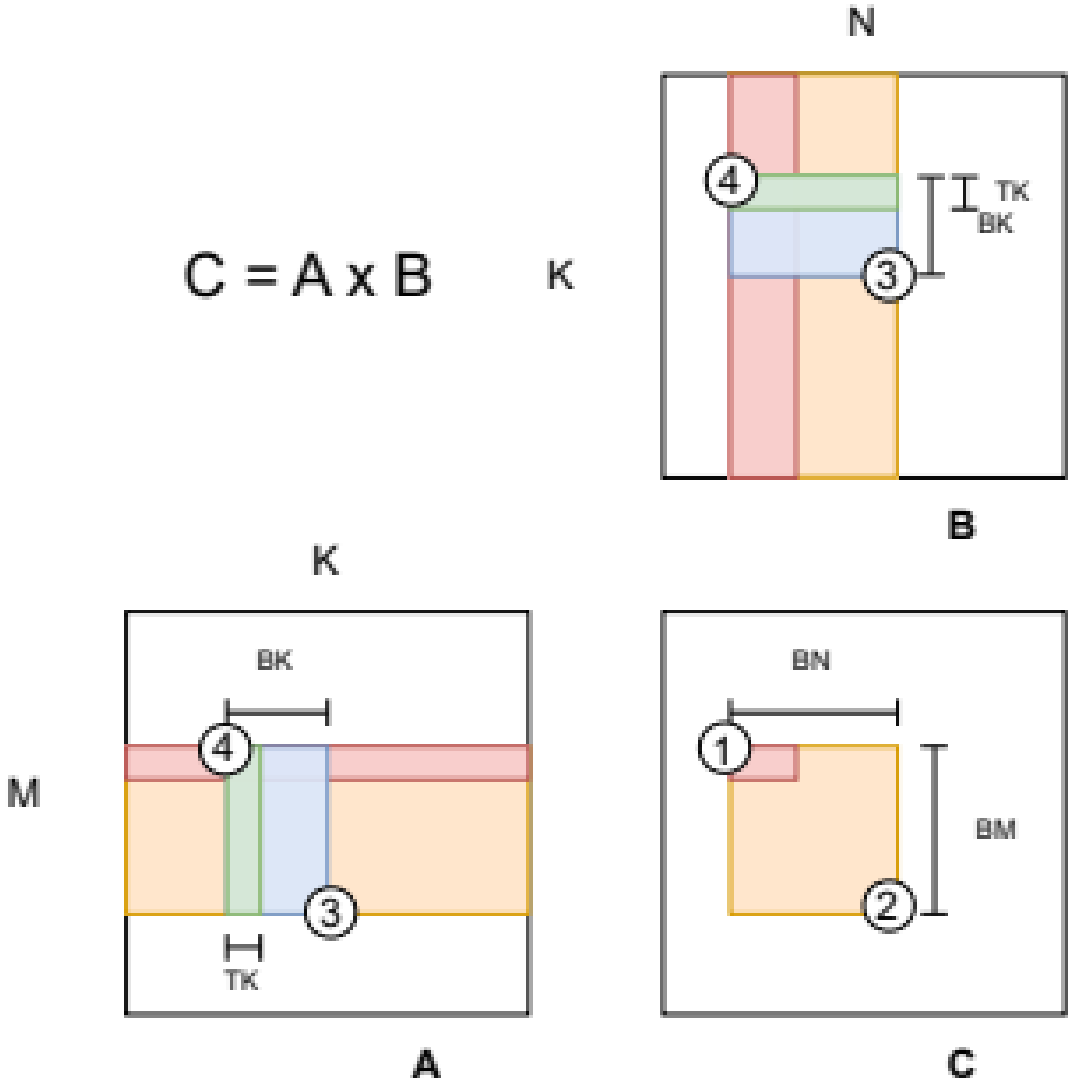


Implement fully in the tiling pass.

```
.tile(<hardware_info>)
```



Tiling Transformations:

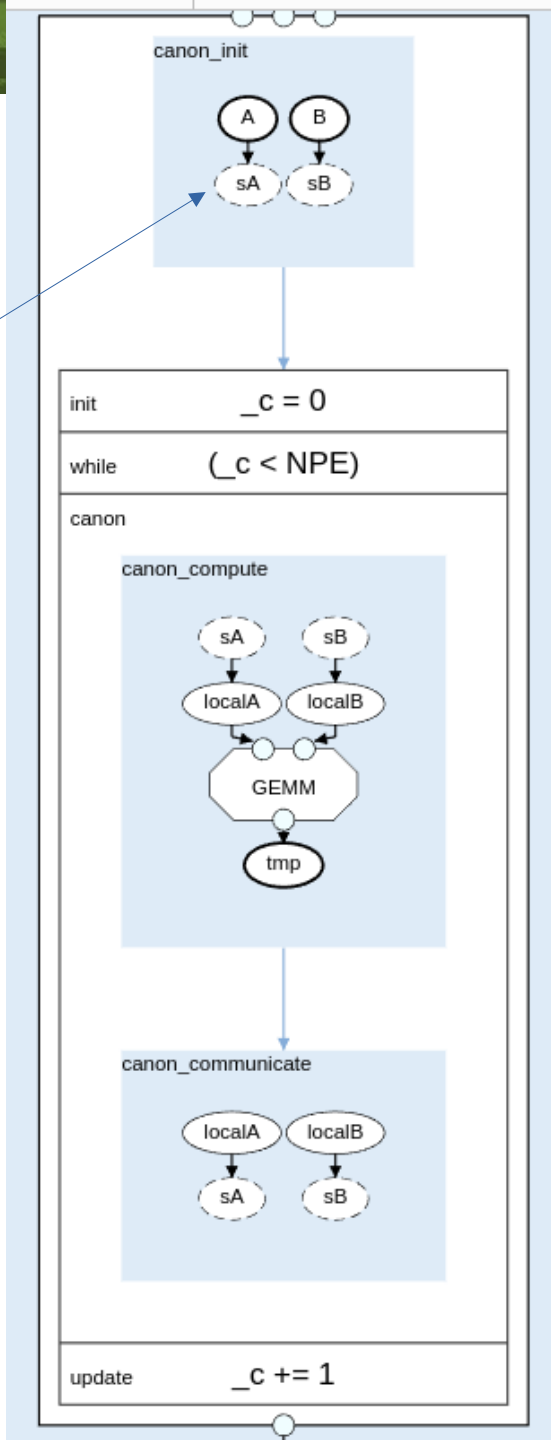


- Transformation (1) defines the per-thread computation domain (*TM* × T*N*)
- Transformation (2) establishes the computation of the core-group (e.g., 32 for 910A, 25 for 910B) domain (*BM* × *BN*).
- The first tiling transformation (3) enables explicit data movement from global memory to A1 and B1 memory locations.
- The second tiling transformation (4) orchestrates movement from A1/B1 to A2/B2 respectively.

Advanced Tiling Transformations: Double-Buffering

The access-node with dashed edges represent a stream. And enables the abstraction of producer-consumer queue to be used for double buffering.

A transformation that accepts a schedule in an input similar to BSP-model has been developed. On the right you can see the schedule that maps to *Canon's algorithm* on the right.



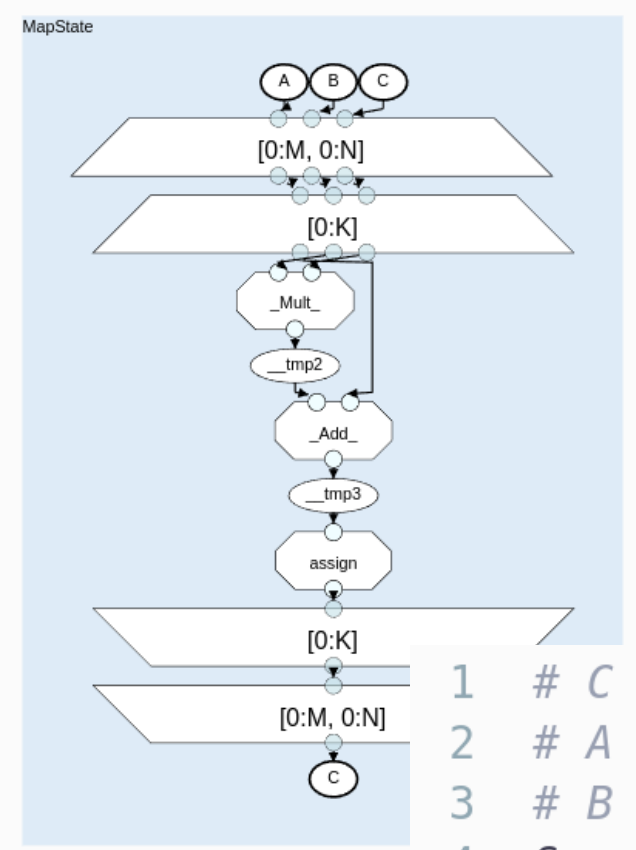
Advanced Tiling Transformations: Double-Buffering

- The transformations configured using the hardware-description of SoftHier is functional and are already in use.
- For SoftHier the streams are utilized to enable DMA-initiated memory transfers for both from global memory and from the the memory of other Pes.
- The integration of streams to SoftHier backend is completed and their integration to Ascend backend is under work.

- **Layout Transformations**

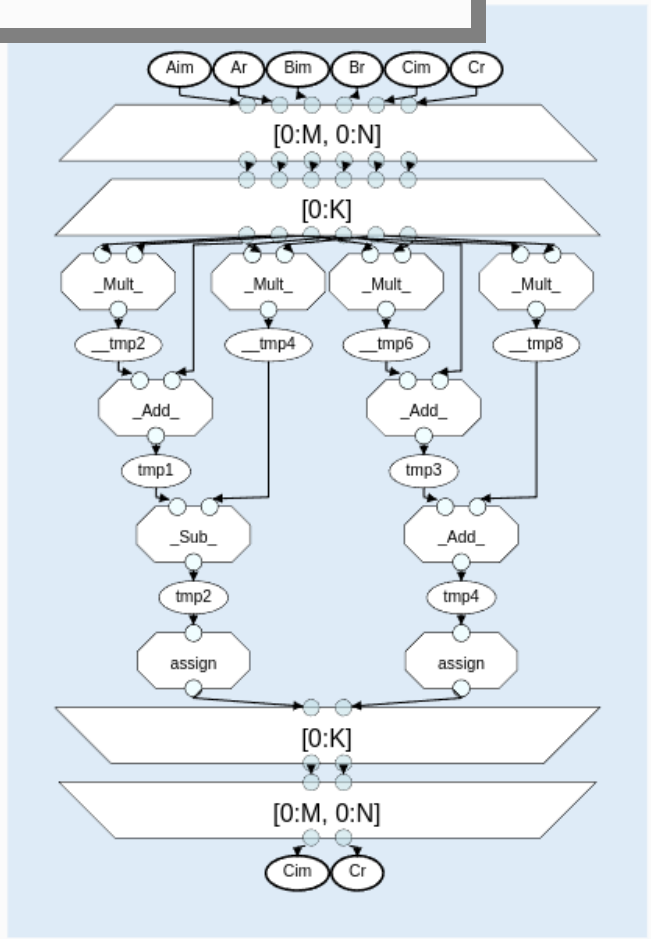
Layout Transformations: AoS-to-SoA Flattening

```
# C_real, C_imag : float32
# A_real, A_imag : float32
# B_real, B_imag : float32
C_real = A_real @ B_real - A_imag @ B_imag
C_imag = A_real @ B_imag + A_imag @ B_real
```




```
# C : complex64
# A : complex64
# B : complex64
C = A  @ B
```

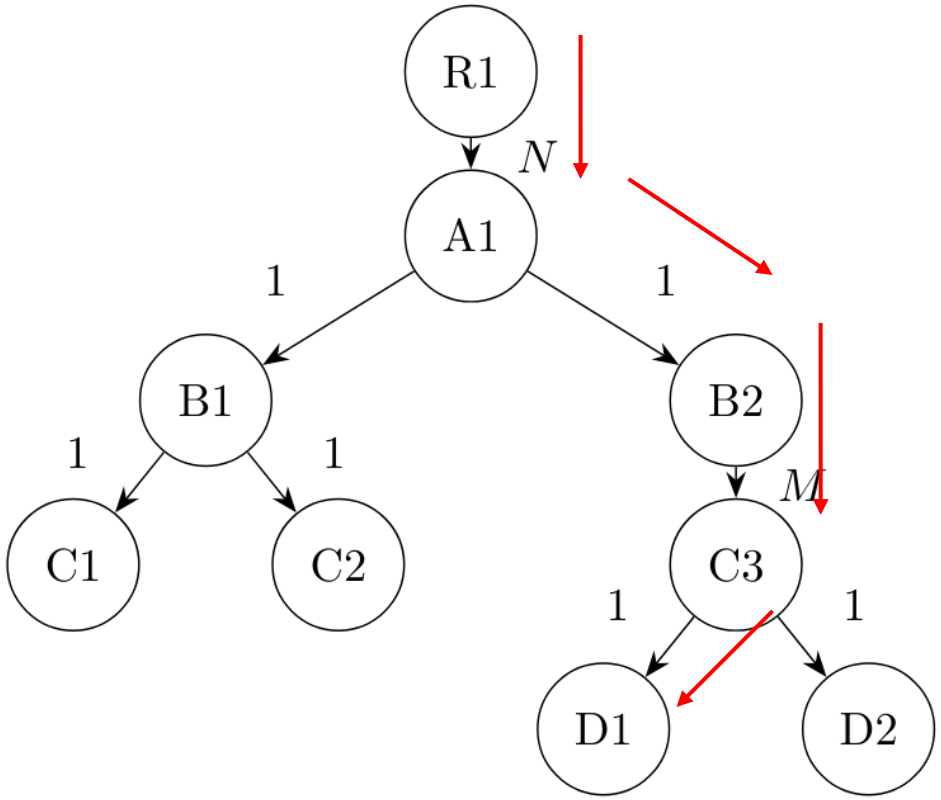
`.apply_pass(Flattening)`



Layout Transformations: AoS-to-SoA Flattening

- Struct-of-Arrays formats are often more suitable for SIMD processing units (e.g. Ascend's Vector Unit) and for dedicated Mat-Mul on hardware (e.g., Ascend's Cube Unit)

- This process determines the dimensions of the new arrays required



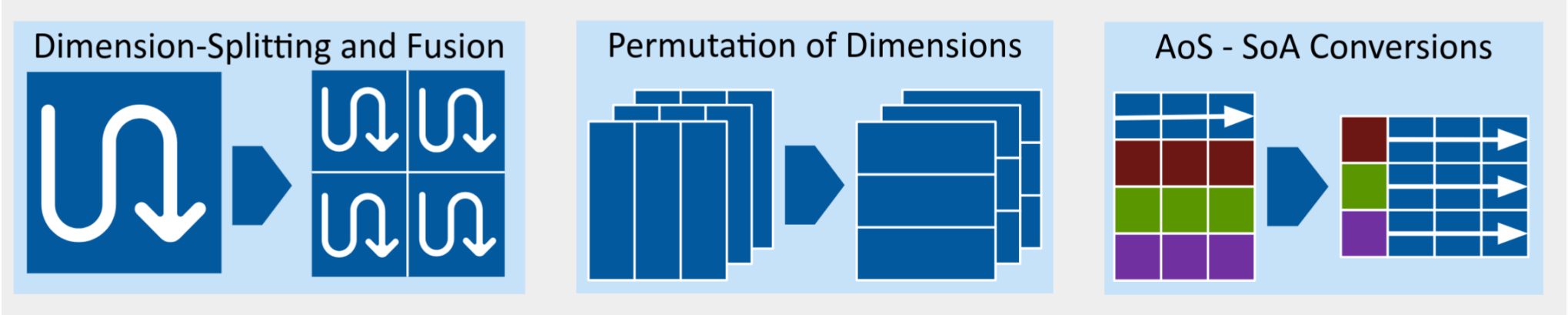
Layout Transformations: AoS-to-SoA Flattening

- The previous struct hierarchy results in four arrays of native C types as they are four paths from the root node to leaf nodes.

Container Name	Dimensions
__CG_R1__CA_A1_CG_B1__m_C1	N
__CG_R1__CA_A1__CG_B1__m_C2	N
__CG_R1__CA_A1__CG_B1__CA_C3__m_D1	N × M
__CG_R1__CA_A1__CG_B1__CA_C3__m_D2	N × M

For example, the array corresponding to a previously shown path is highlighted in red

Layout Transformations:



- Transposing Matrices is a type of Dimension Permutation:
- **B[K, N] → B[N, K]**

- Block-Tiled Storage is a type of Splitting the Dimensions:
- **A[M, K] → A[M//32][K//32][32][32]**

- Flattening is useful to support the programming language abstract without impacting performance:
- **C[M, N] : complex64 → C_real[M, N] : float32, C_imag[M, N] : float32**

Overiew of the Topics:

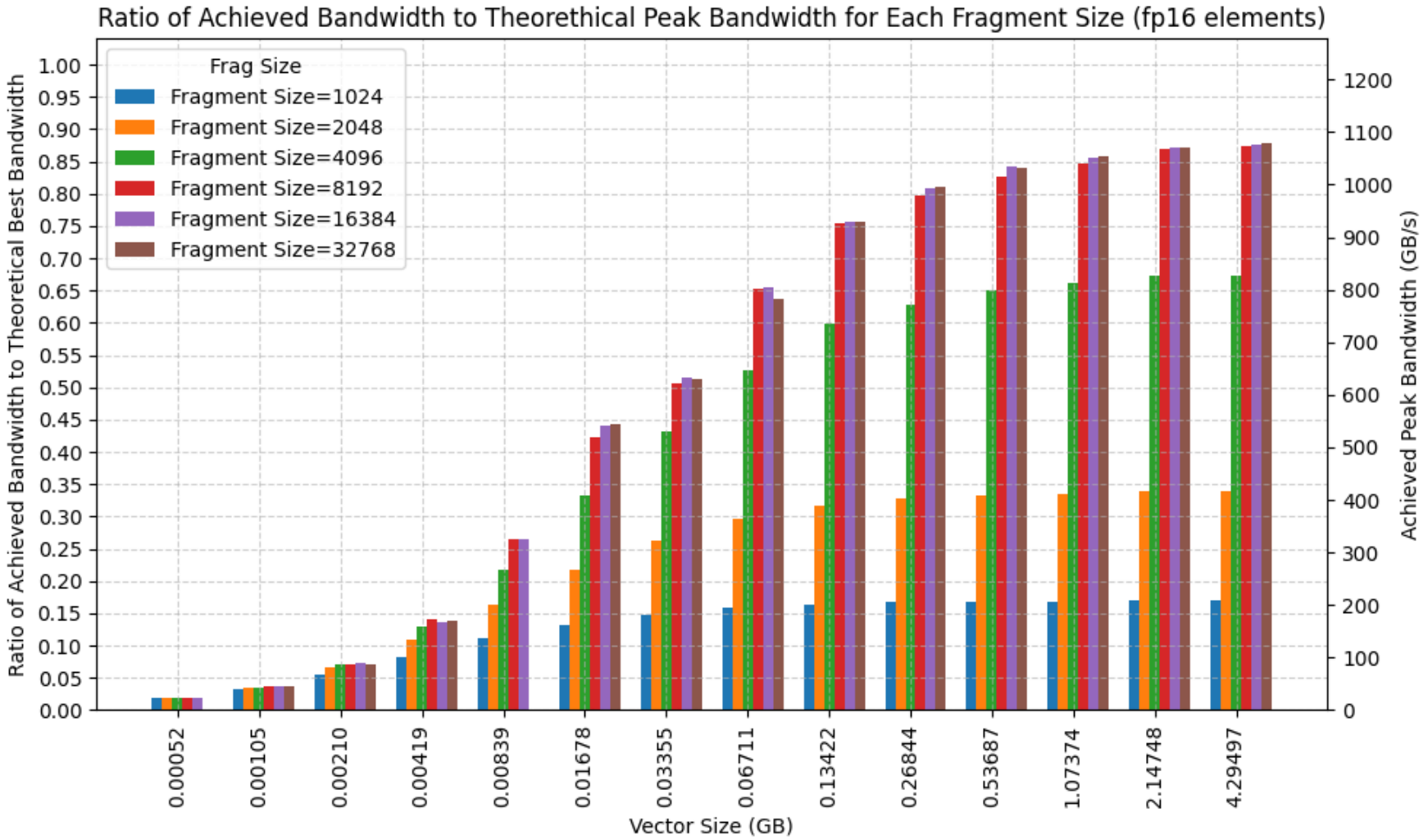
- **Code-Generation and Results**

Code-Generation and Results

- Code-generation for Vector-Units are functional and ready.
- Code-generation for Cube units will be ready by June.

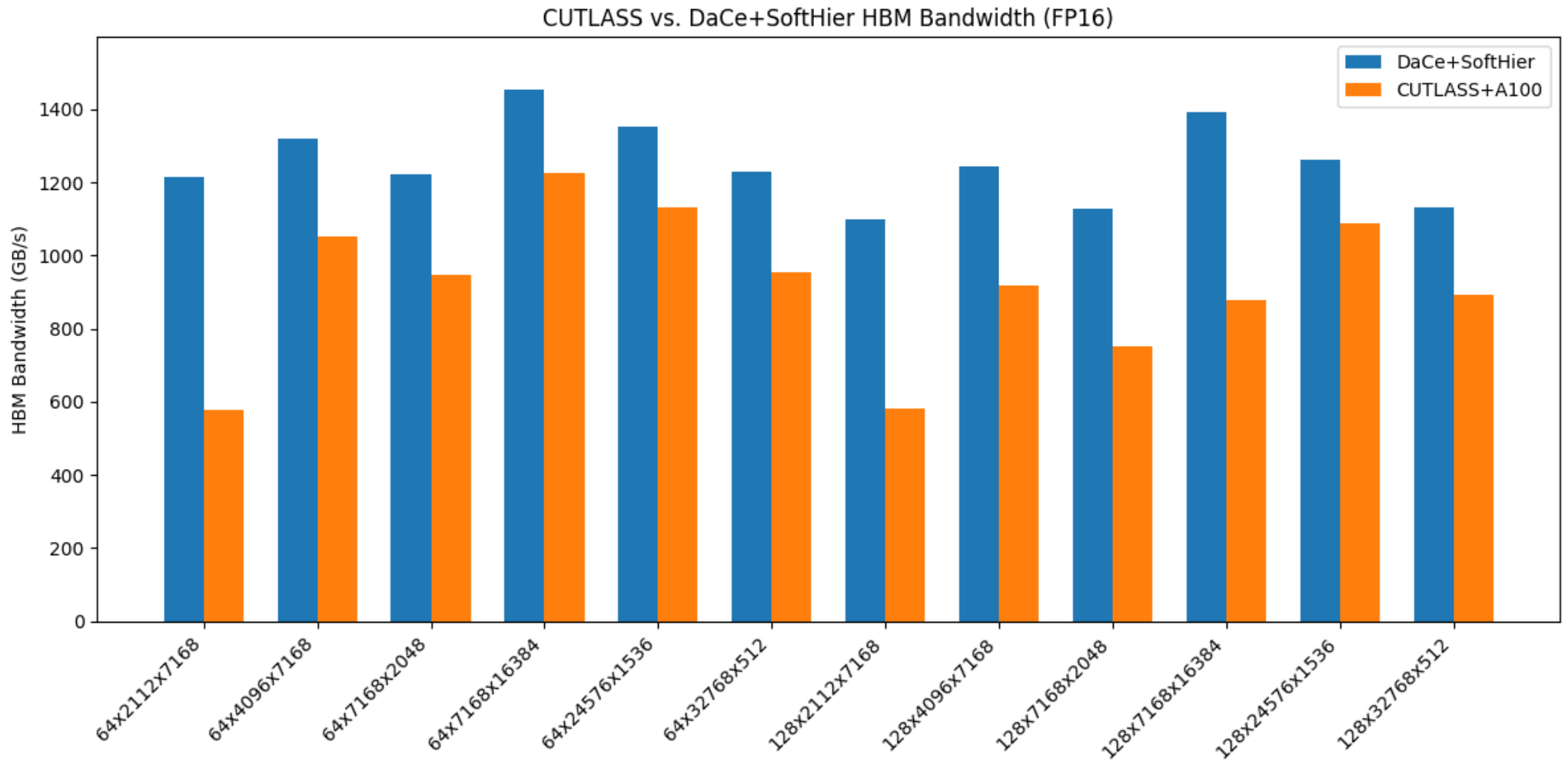
Code-Generation and Results

- Vector-copy benchmark running on Ascend 910A:



Code-Generation and Results

- Bandwidth utilization for GEMM with varying dimensions (MxNxK) comparing SoftHier to A100.



Transformation Updates

- DaCe is able to generate efficient code on multiple hardware.
  - By implementing the transformation to take hardware configurations as inputs, it is possible to reuse the whole (and if not most of the) components on different hardware.
- From the initial results, the optimal tiling strategy for Ascend might be closer to multi-core CPUs where threads do not need to coalesce access to global memory to achieve peak bandwidth, but we optimize for cache locality.
  - GEMM baseline and results are planned to be completed by June.