Yakup Koray Budanaz

SoftHier May 5 Update

Overiew of the Topics:

- Restarted Work on AscendC Backend
- Plan until June
- Cleaned-up / Completed AD of the Layout Paper

- **Restarted Work on AscendC Backend**

Restarting Work on the AscendC Backend:

- Cleaned-up Repositories (DaCe Ascend branch, Workspace Repository)
- Updated the CANN version from 8.0_RC1 to 8.1_RC1 on the 910A Server
- Updated the DaCe branches
- Had no connection to 910B server (fixed it with Renzo today)

Restarting Work on the AscendC Backend: Current Issues

- Default GCC's glibc and libstdc++ is too old for vscode-server on 910A server.
- Should I only work on 910B server? Are there plans to update the archaic GCC version on 910A – if not I will pass a spack-compiled GCC to bashrc/bash_profile.

The following issues were detected in your most recent Remote - SSH session
Cmd+click on an issue to continue in Copilot Chat.

Status	Message	Mitigations	Resources
LinuxPrereqs	The remote host may not meet VS Code Server's prerequisites for glibc and libstdc++ (The remote host does not meet the prerequisites for running VS Code Server)	• https://aka.ms/vscode-remote/faq/old-linux	• https://aka.ms/vscode-remote/linux-prerequisites

See verbose information in the Output Log...

Deliverables: Stencil Microkernels on Vector Processors

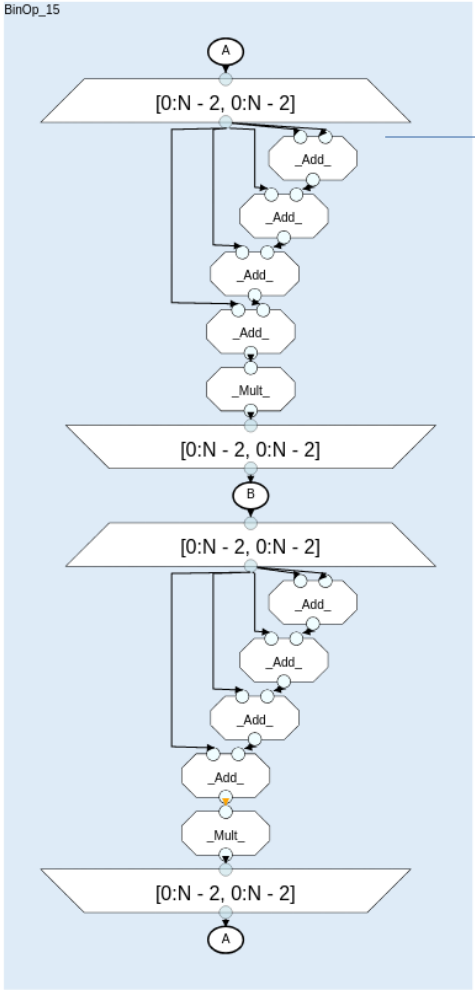
- Initial target is to have the simplest stencil benchmark run on Ascend vector processors. E.g. computing 2D Jacobi-stencil:

```
B[i, j] = 0.2 * (A[i, j] + A[i, j-1] + A[i, j+1] + A[i+1, j] + A[i-1, j])
```

```
@dace.program
def dace_jacobi_kernel(TSTEPS: dace.int32,
                       A: dace.float32[N, N] @ dtypes.StorageType.GPU_Global,
                       B: dace.float32[N, N] @ dtypes.StorageType.GPU_Global):
    for _ in range(0, TSTEPS):
        B[1:-1, 1:-1] = 0.2 * (A[1:-1, 1:-1] + A[1:-1, :-2] + A[1:-1, 2:] +
                               A[2:, 1:-1] + A[:-2, 1:-1])
        A[1:-1, 1:-1] = 0.2 * (B[1:-1, 1:-1] + B[1:-1, :-2] + B[1:-1, 2:] +
                               B[2:, 1:-1] + B[:-2, 1:-1])
```

Deliverables: Stencil Microkernels on Vector Processors

- The Jacobi Microkernel in SDFG IR looks as follows (1 time-step):



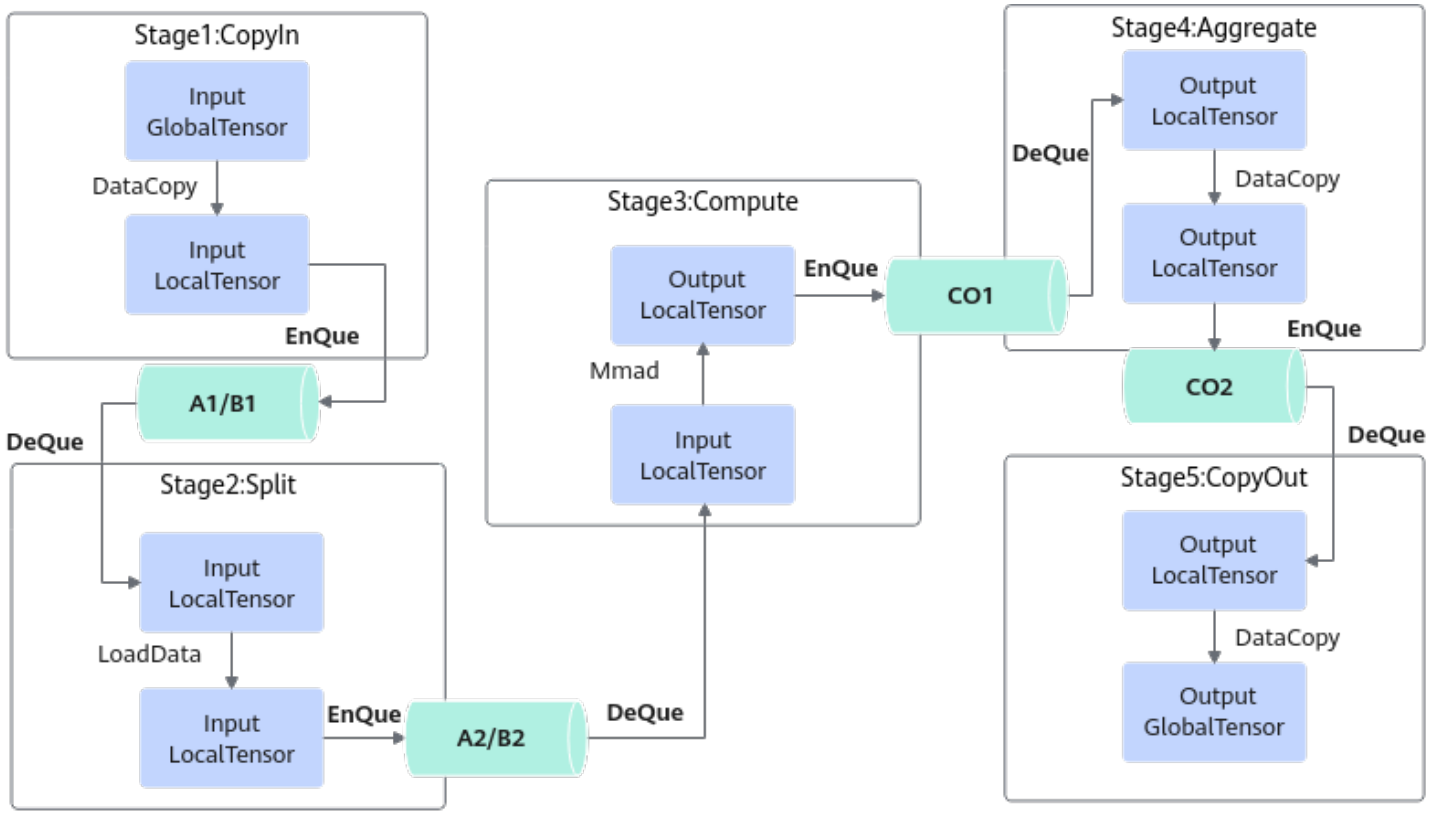
Current tasklet code: `_out = _in1 + _in2`

The idea is as follows:
- Tile the map in one dimension
- Replace scalar tasklet code with vectorized tasklets compatible with Ascend
- `_out = _in1 + _in2 //dace.int32@register)`
- `_out = Add(_in1, _in2) //dace.int32[<size>]@vecin)`

- Will work combined with the pass that inserts memory transfers between maps.

Issues: 910B Documentation?

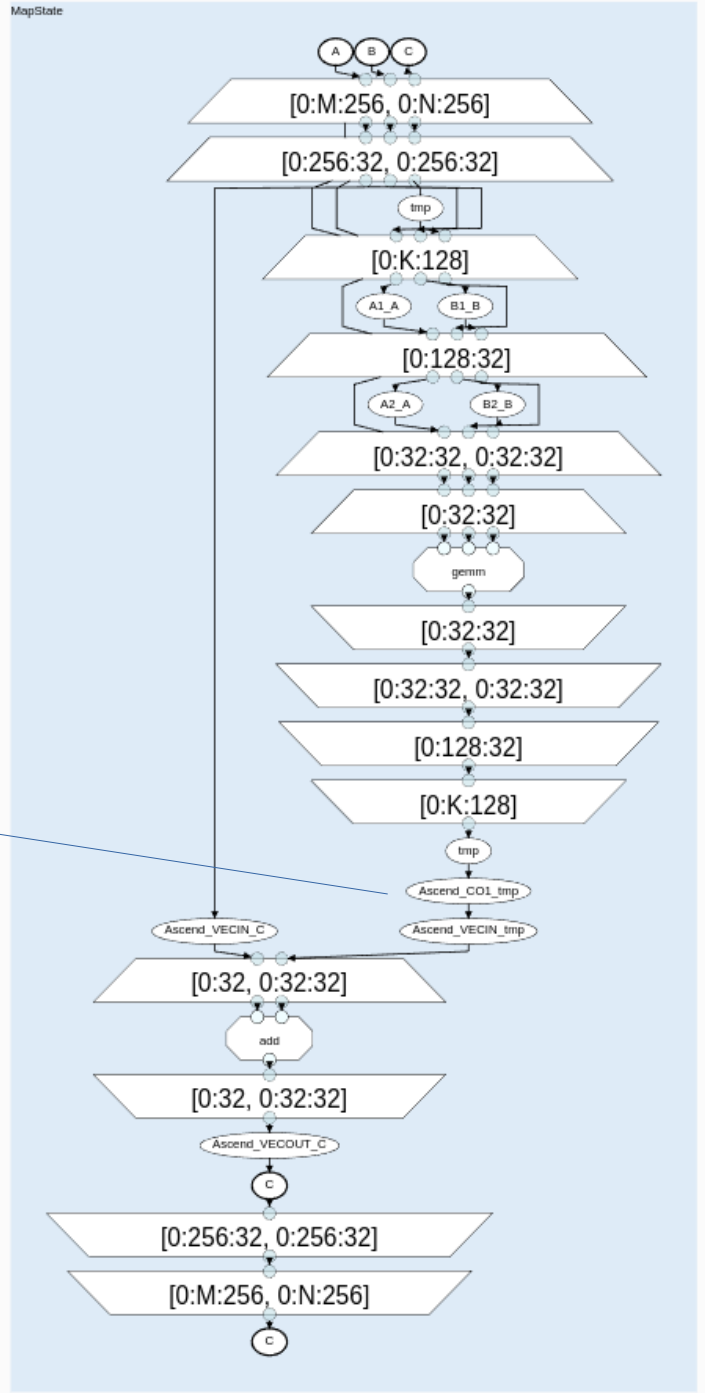
- It is a major inconvenience to search in the auto-translated documentation (website / PDF both).
- Public documentation seems to be only for 910A, is there documentation I can access for 910B?:



https://www.hiascend.com/document/detail/zh/canncommercial/81RC1/developmentguide/opdevg/Ascendcopdevg/atlas_ascendc_10_00006.html

Issues: 910B Documentation?

- I remember discussing that the C02 → C01 → VECIN transfer is not not necessary on 910B.
- Where can I find this information?
- Where can I find the exact differences between 910B and 910A?



Restarting Work on the AscendC Backend:

- I am working on AscendC backend to extend the explicit memory movement to generate code for A1 → A2, C02 → C01 memory copies etc.
- I will continue to using the Que abstraction with Enqueue and Dequeue abstractions
  - No major updates here, working on generating the code to perform code copies.

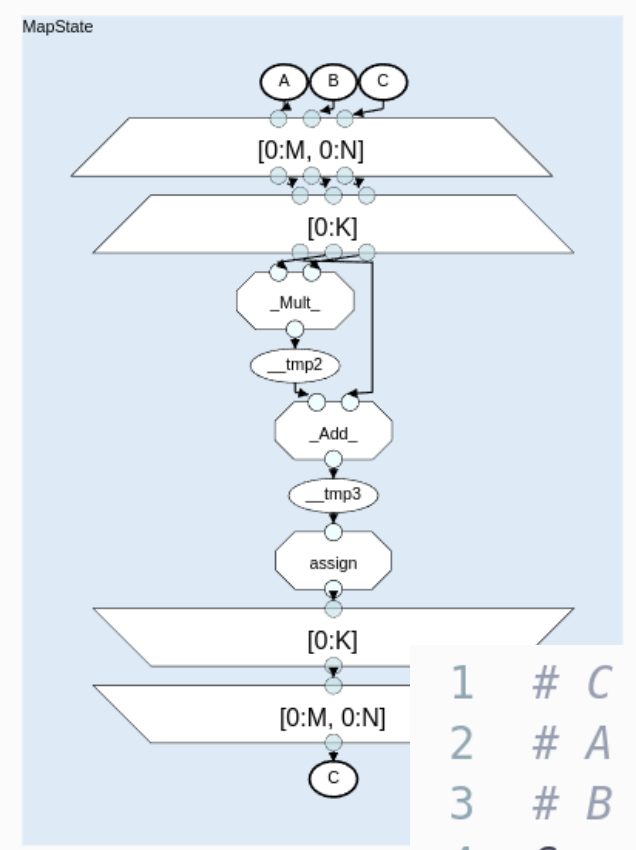
TODOs:

- Finish GEMM baseline (no double-buffering, using the Que abstraction, 2D-blocktiled tiling)
- Stencil microbenchmark running on vector processors – combined with an example of AoS-to-SoA flattening.

- **Cleaned-up / Completed AD of the Layout Paper**

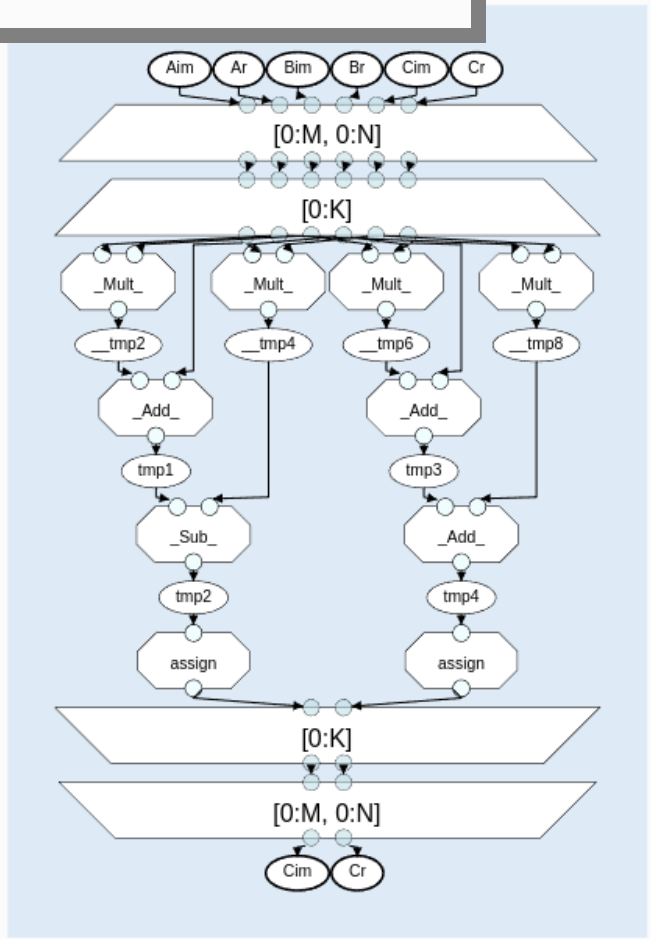
Layout Transformations: AoS-to-SoA Flattening

```
1  # C_real, C_imag : float32
2  # A_real, A_imag : float32
3  # B_real, B_imag : float32
4  C_real = A_real @ B_real - A_imag @ B_imag
5  C_imag = A_real @ B_imag + A_imag @ B_real
```



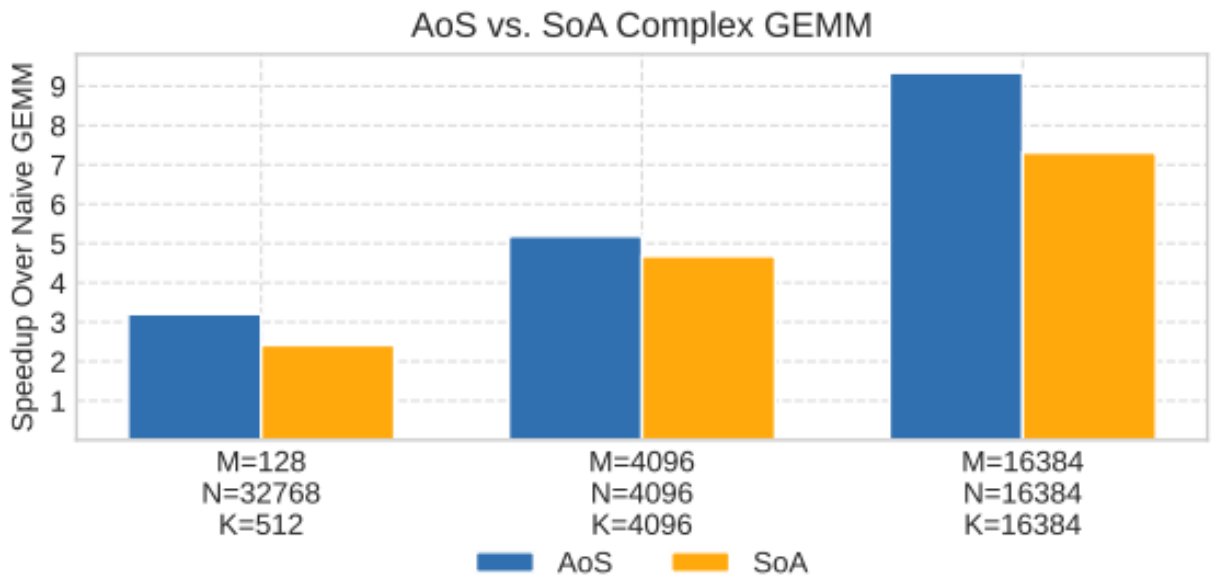

```
1  # C : complex64
2  # A : complex64
3  # B : complex64
4  C = A  @ B
```

.apply_pass(Flattening)



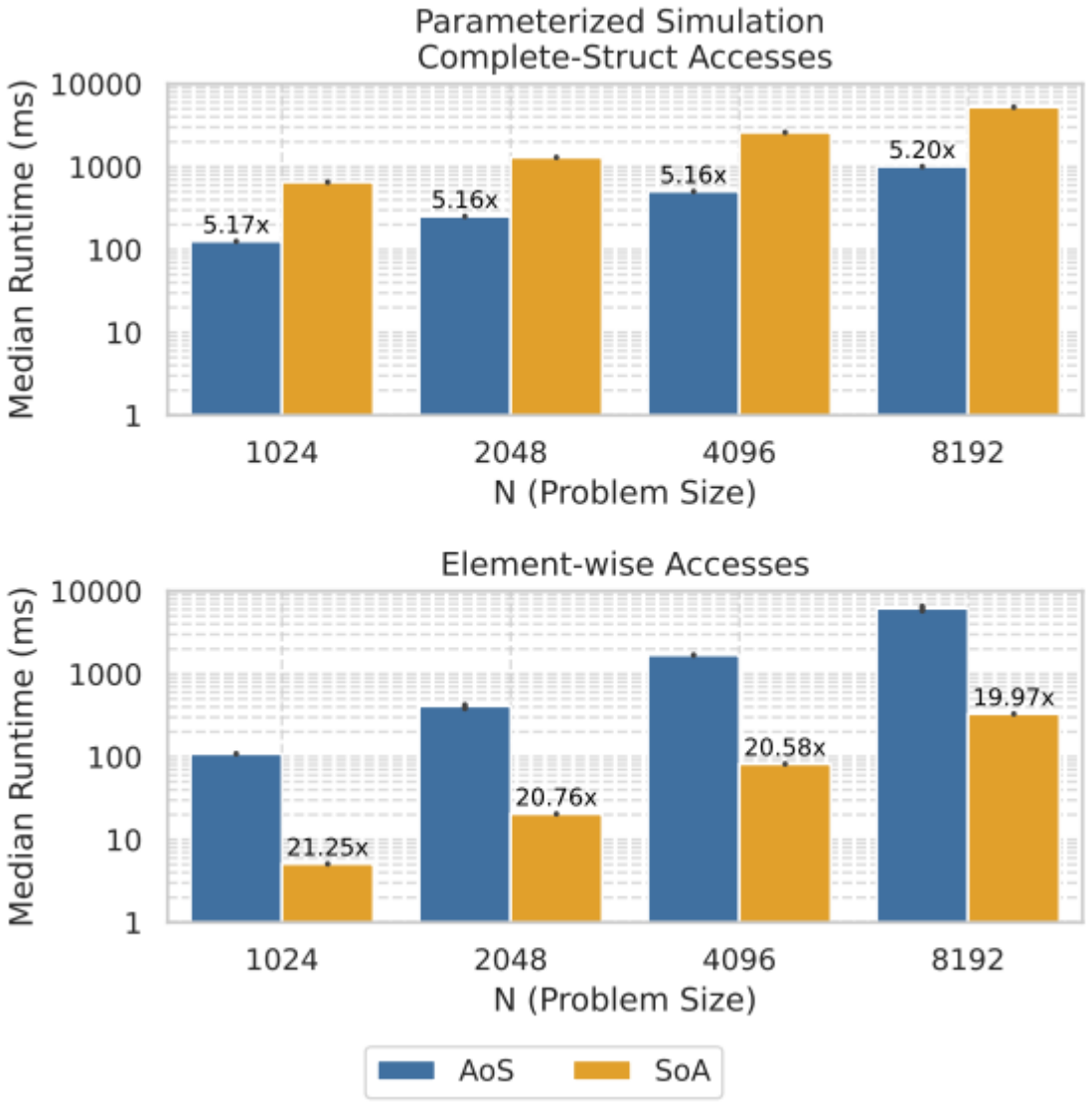
Layout Paper Results:

- AoS Layout is slightly faster on the AMD MI300A GPUs used for the benchmark (Beverin Cluster of CSCS)



Layout Paper Results:

- We then create synthetic benchmarks based on particle simulations – ensuring AoS / SoA is clearly faster than the other layout.
- Experiment ran on AMD EPYC 7502 CPUs

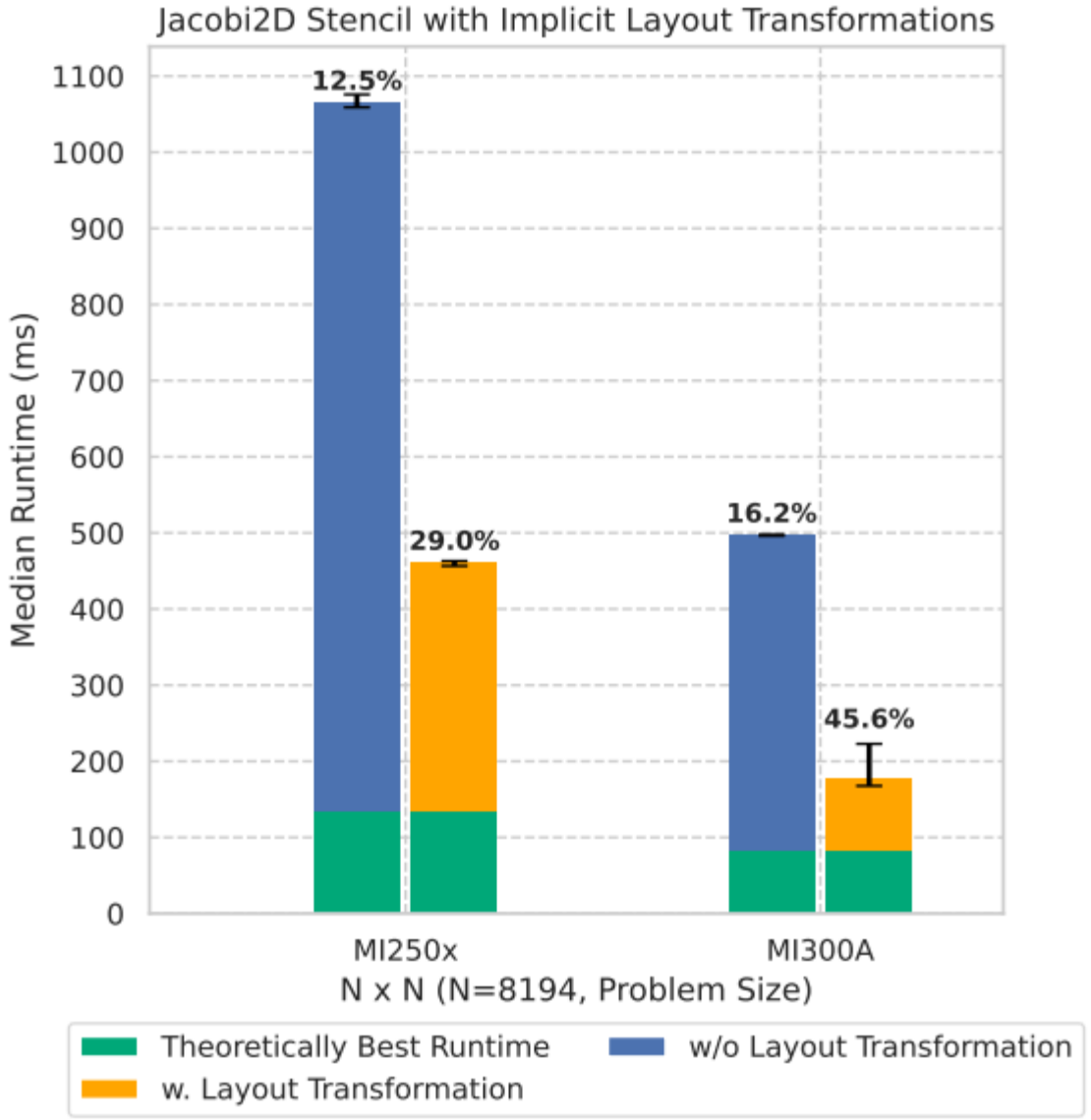


Takes For SoftHier Project:

- In some cases AoS (or hybrid) layouts are more performant than SoA.
  - But flattening enables these programs to run on Ascend devices (since it can only support SoA due to only having Vector/Matrix processors)

Layout Paper Results:

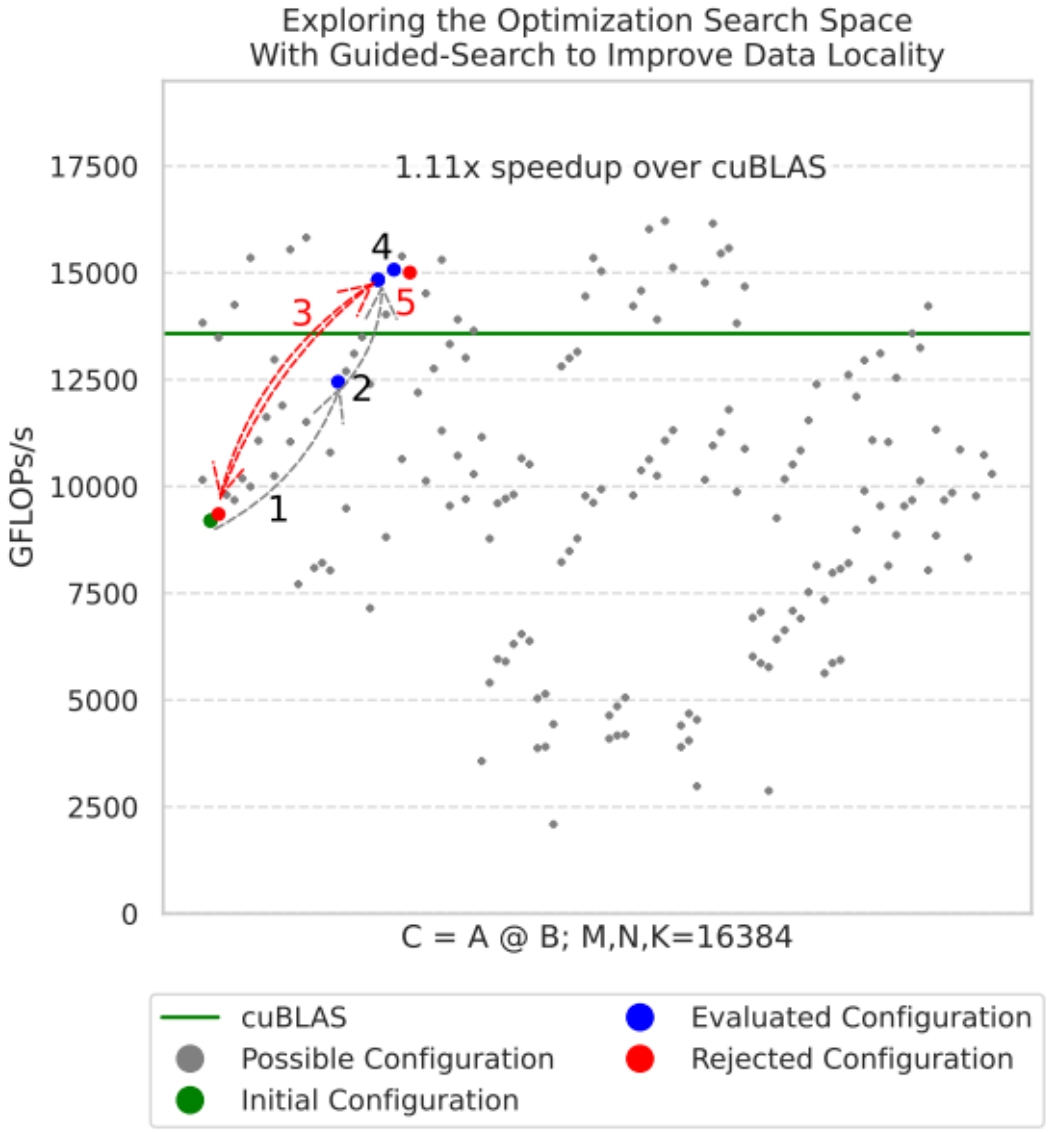
- The auto-layout pass we have applied on stencil microbenchmark (Jacobi2D as in previous slides) on AMD GPUs (with extensions/fixes to DaCe on GPU pipeline).
- HIP compilers out-of-the-box performance seem to be very bad.
- GPU-Stream on HIP with AMD MI300A gives ~60% of theo. bandwidth while CUDA on GH200 gives ~90%.




```
1  B[i, j] = 0.2 * (A[i, j] + A[i, j-1] + A[i, j+1] + A[i+1, j] + A[i-1, j])
```

Layout Paper Results:

- The guided-search function was also added to the publication. The algorithm has not changed much since the time I presented it couple of months ago.
- CPUs and GPUs require completely different heuristics on the search of tiling parameters.
- I guess AscendC will prefer similar heuristics to CPUs. (The guided-search function for Ascend will possibly need to be done after June)



Takes For SoftHier Project:

- Need to develop new heuristics for Ascend (as the better-working schedule/tiling configurations should be more similar to CPUs than GPUs)
  - CPU auto-tile/auto-schedule pipeline is partially done

Layout Paper Results:

- We also had a benchmark on index-permutations on a semi-structured microbenchmark:

```
for i in [1..N-1]:
  for j in [1..N-1]:
    for k in [1..N-1]:
      B[i, j, k] = c * (
        A[i, j, k] +
        A[i, j - 1, k] + A[i, j + 1, k] +
        A[idxA[i, j, 0], j, idxA[i, j, 4]] +
        A[idxA[i, j, 1], j, idxA[i, j, 5]] +
        A[idxA[i, j, 3], j, idxA[i, j, 6]] +
        A[idxA[i, j, 4], j, idxA[i, j, 7]]
      )
```

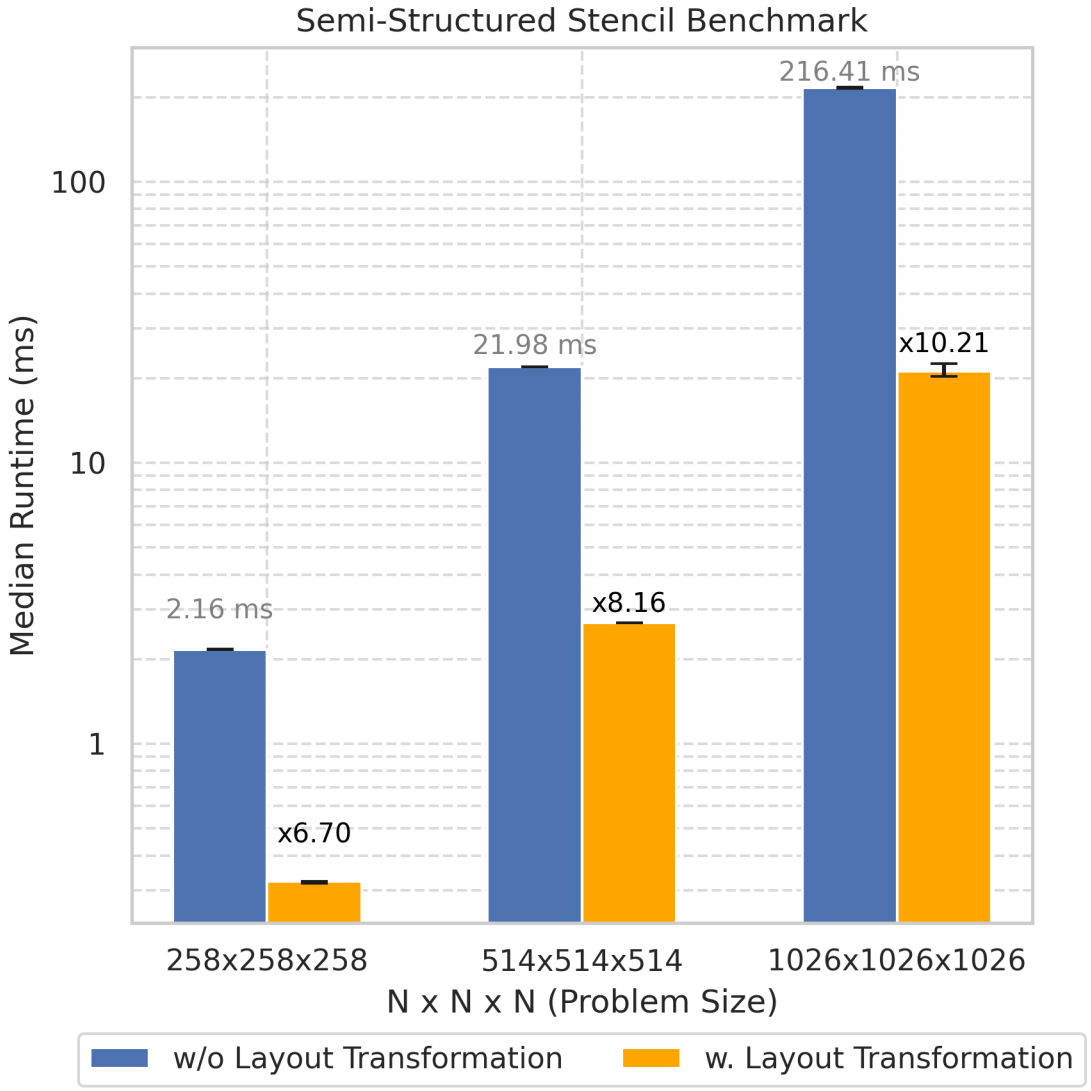
Layout Paper Results:

- Applied as transformation, which looks as follows:

```
if (__LOOP_EXCHANGE):
  # permutes nproma and nlev
  program.permute_indices(array_id, [1, 0, 2])
  program.permute_schedule(loop_id, [1, 0, 2])
```

Layout Paper Results:

- Ensuring structured access to fall on the contiguously stored dimension ensures a speed-up of up to ~10x speed-up on stencil based microbenchmarks.

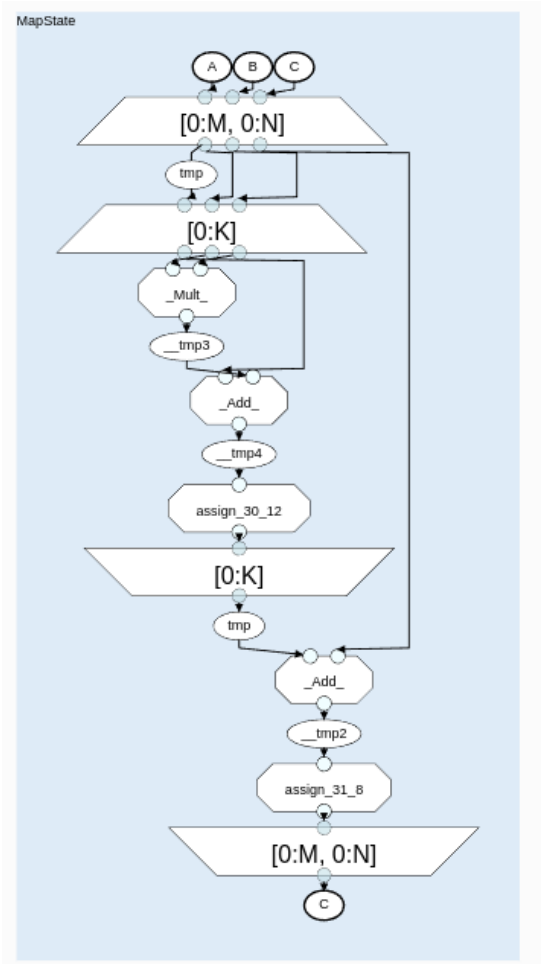


Takes For SoftHier Project:

- We might be able to run some programs that would be otherwise hard to run on Ascend with dedicated layout transformation.
  - (Completely unstructured accesses are possibly not feasible to target on Ascend)

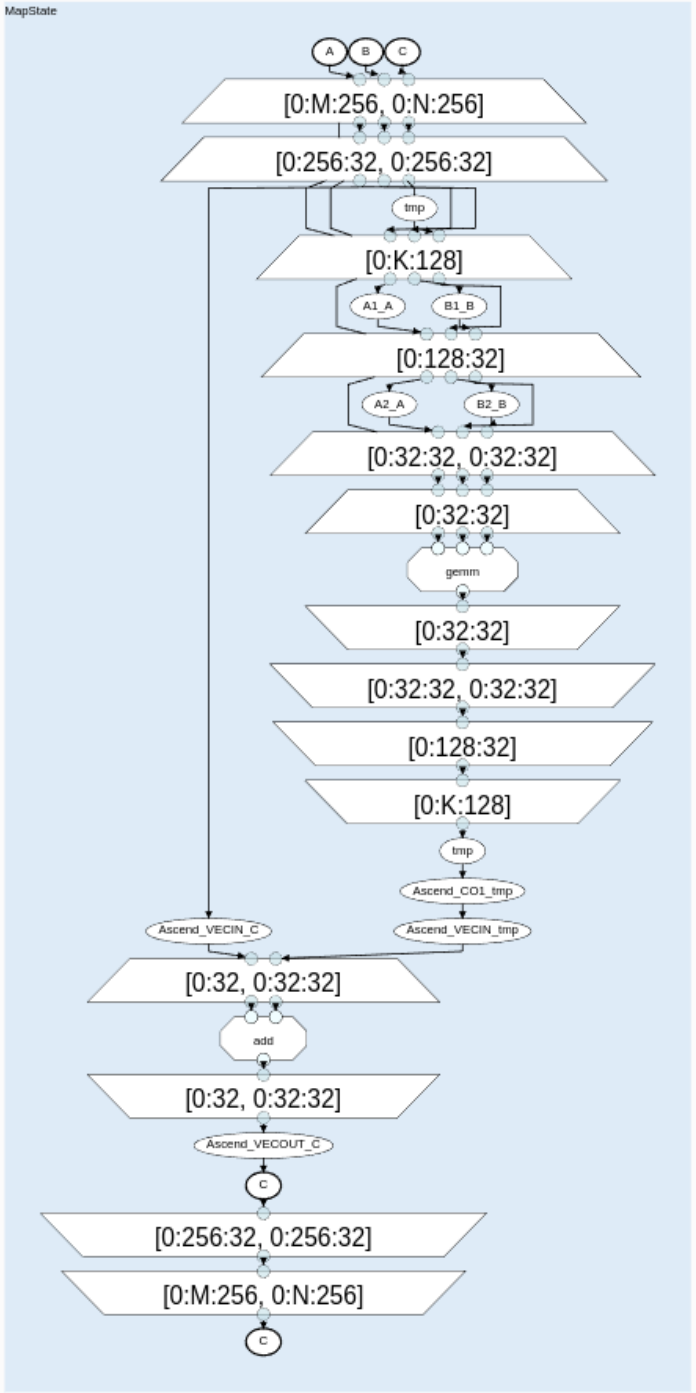
- **Extra Slides (On Layout Transformations)**

Tiling Transformations:

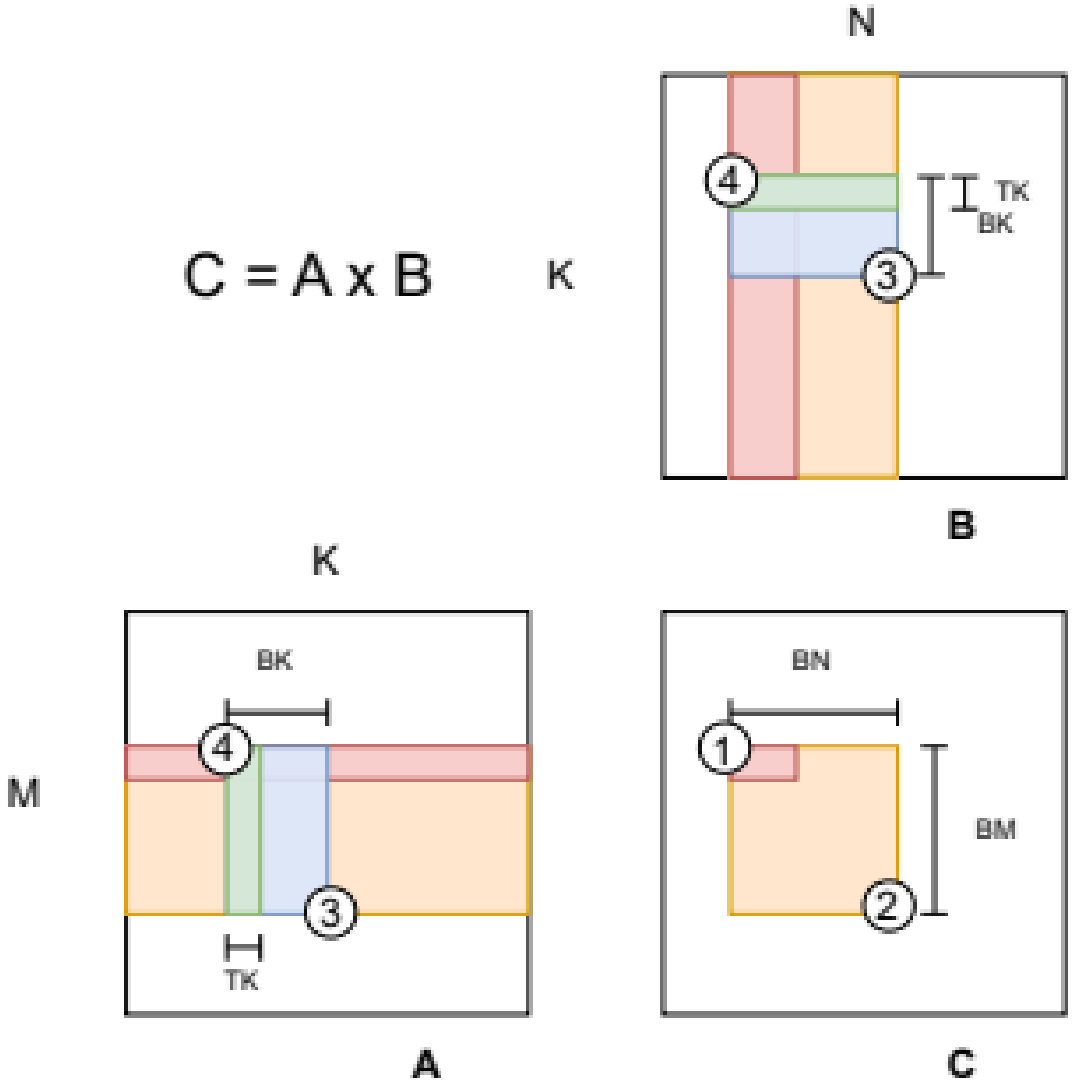


Implement fully in the tiling pass.

`.tile(<hardware_info>)`



Tiling Transformations:

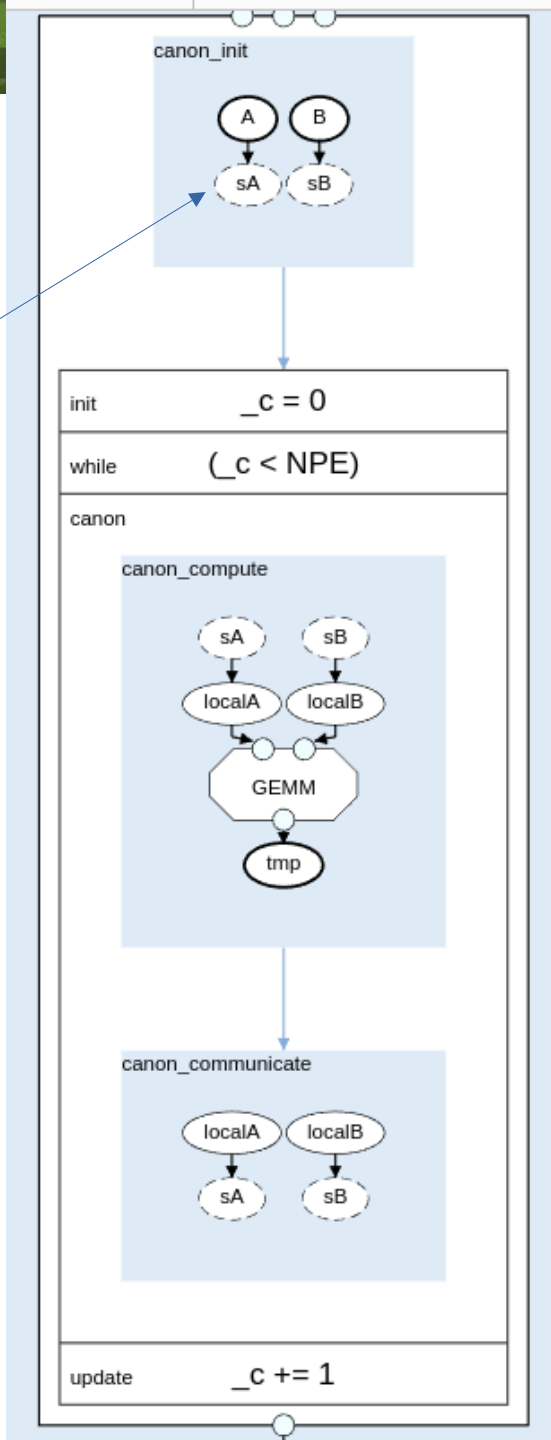


- Transformation (1) defines the per-thread computation domain (*TM* × T*N*)
- Transformation (2) establishes the computation of the core-group (e.g., 32 for 910A, 25 for 910B) domain (*BM* × *BN*).
- The first tiling transformation (3) enables explicit data movement from global memory to A1 and B1 memory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



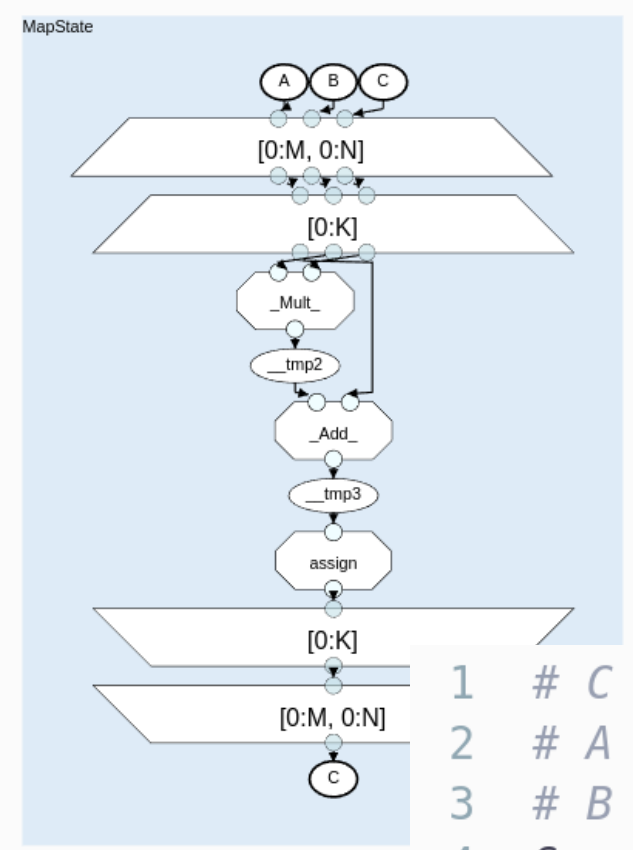
Advanced Tiling Transformations: Double-Buffering

- 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.

- **Restarted Work on AscendC Backend**

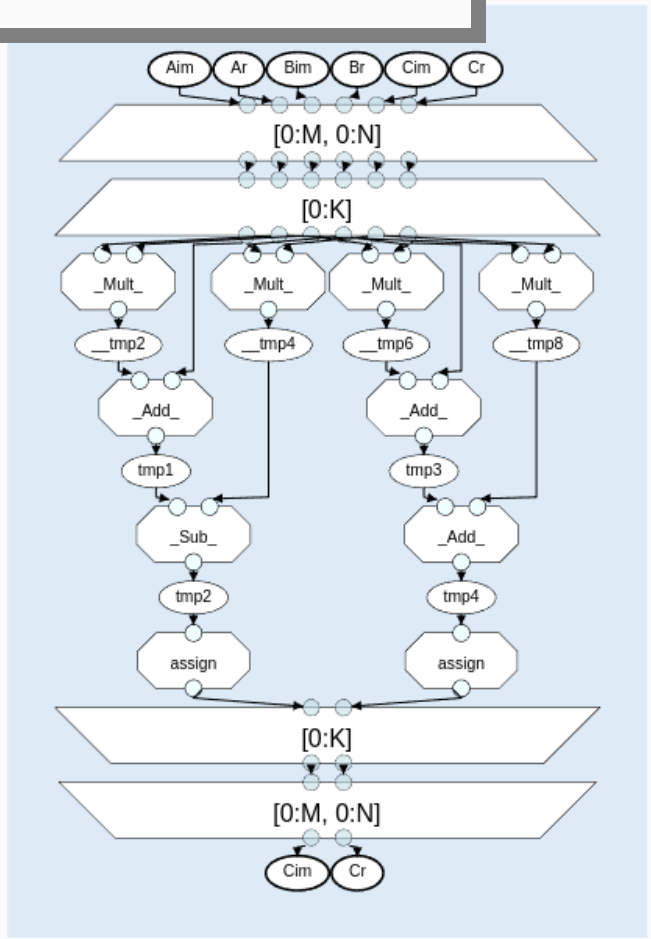
Layout Transformations: AoS-to-SoA Flattening

```
# 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
```

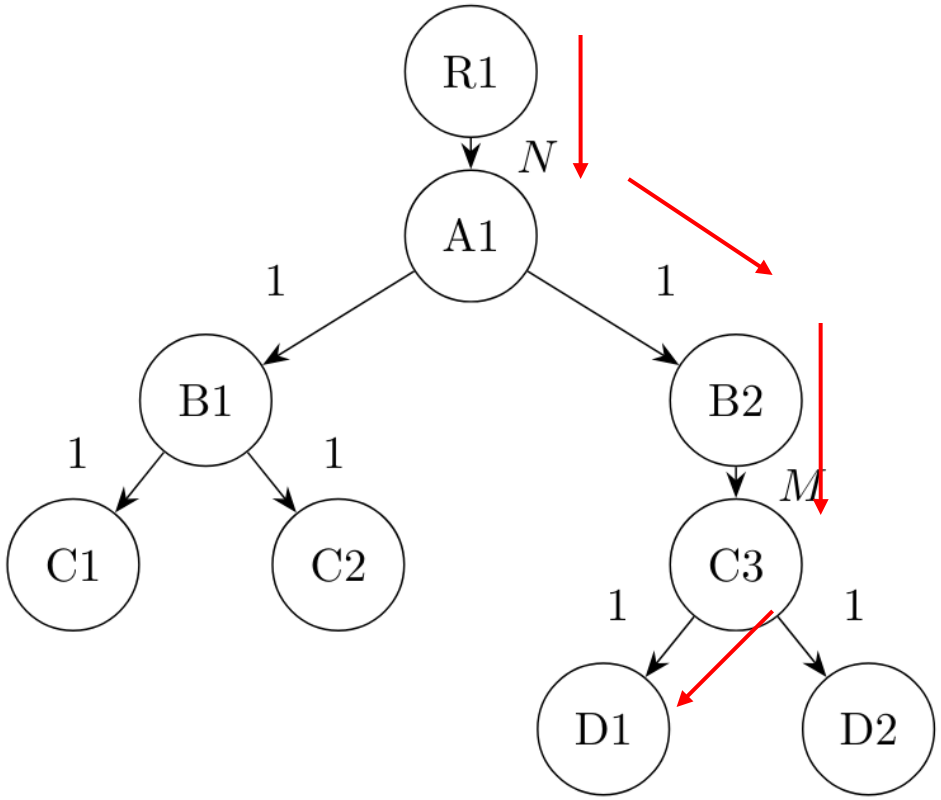
`.apply_pass(Flattening)` →



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



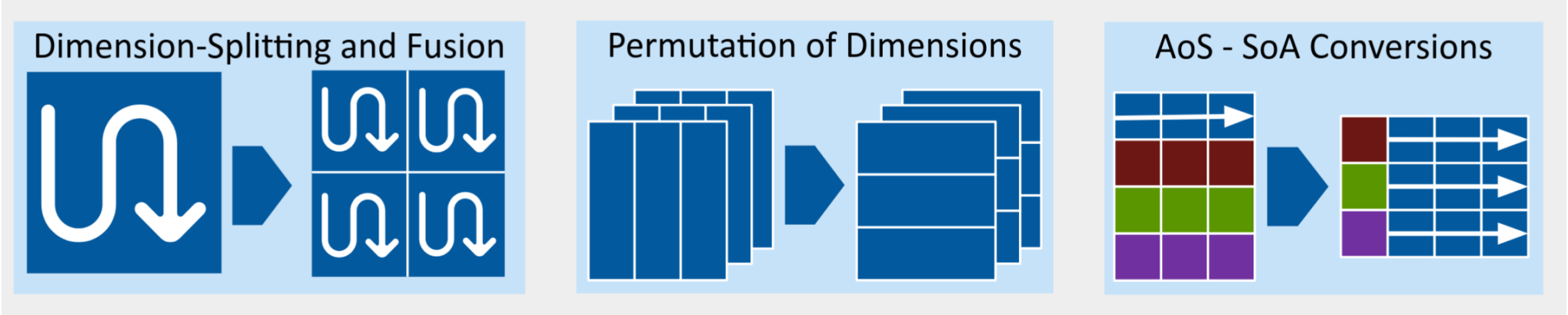
Layout Transformations: AoS-to-SoA Flattening

- 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]**

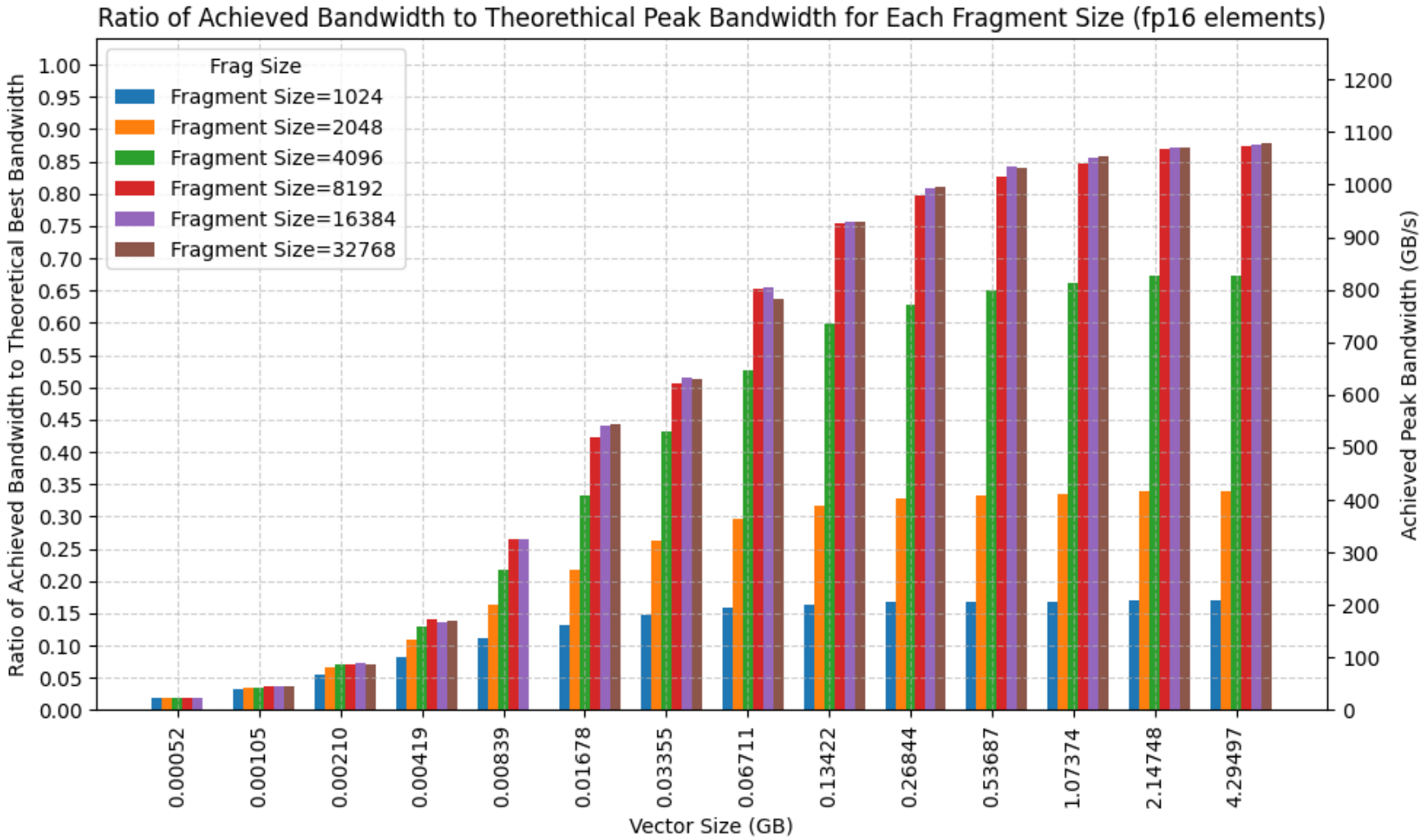
- Flattening is useful to support the programming language abstract without impacting performance:
- **C[M, N] : complex64 → C_real[M, N] : float32, C_imag[M, N] : float32**

Code-Generation and Results

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

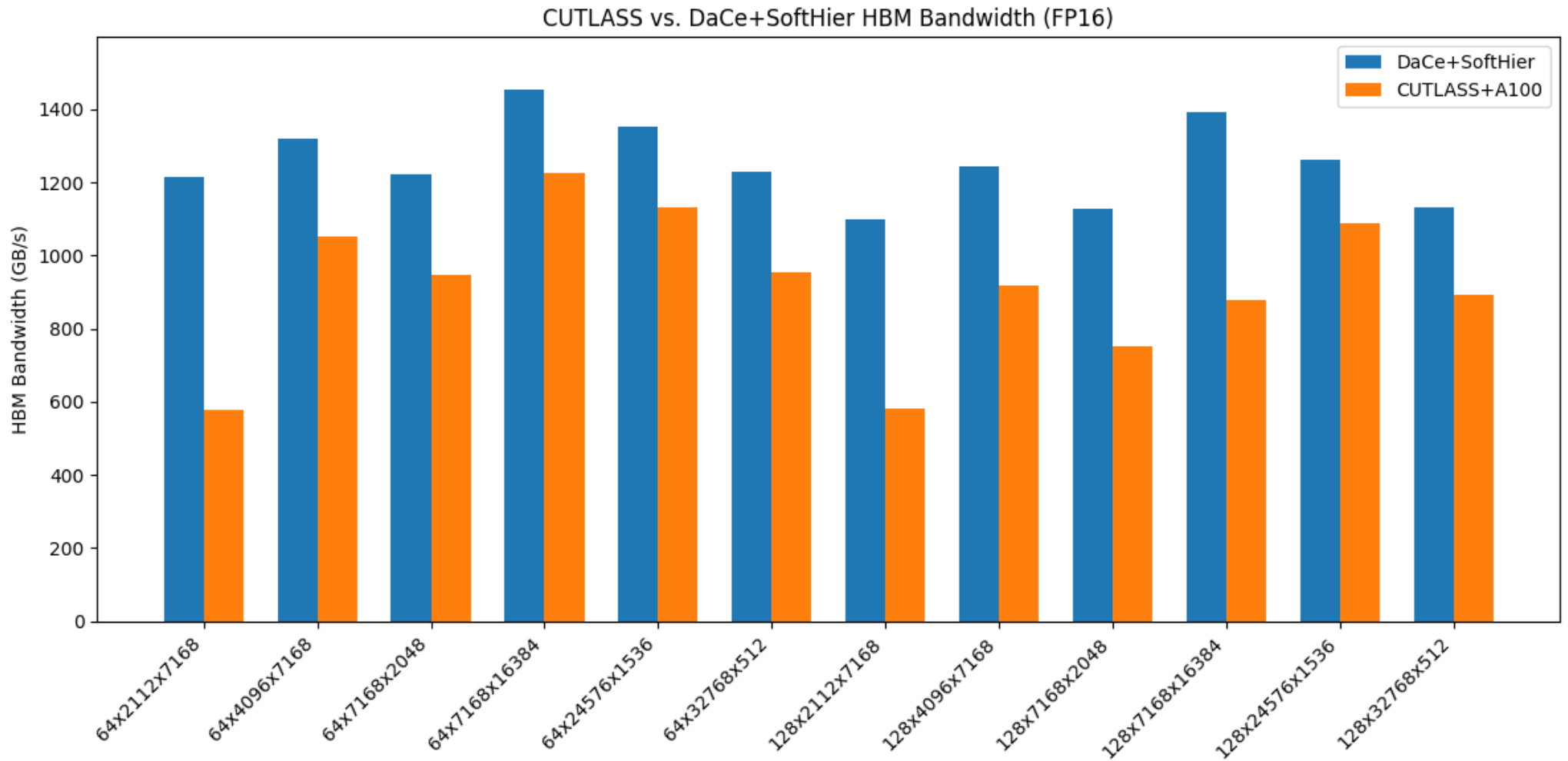
Code-Generation and Results

- 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.