Yakup Koray Budanaz

SoftHier Progress Report January 27

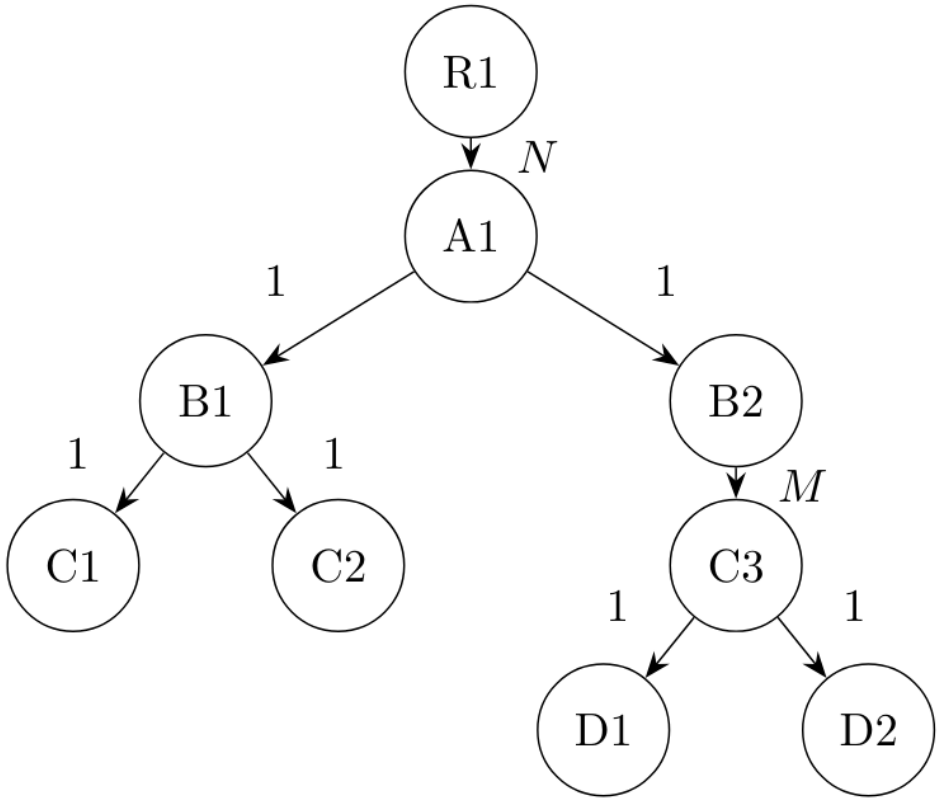
Overiew of the Topics:

- DaCe Features:
  - Struct-of-Arrays Flattening Transformation
  - Deferred Allocation
- Tiling Transformations
- DaCe + SoftHier
- Outlook For Next Weeks

- **Struct-of-Arrays Flattening Transformation**

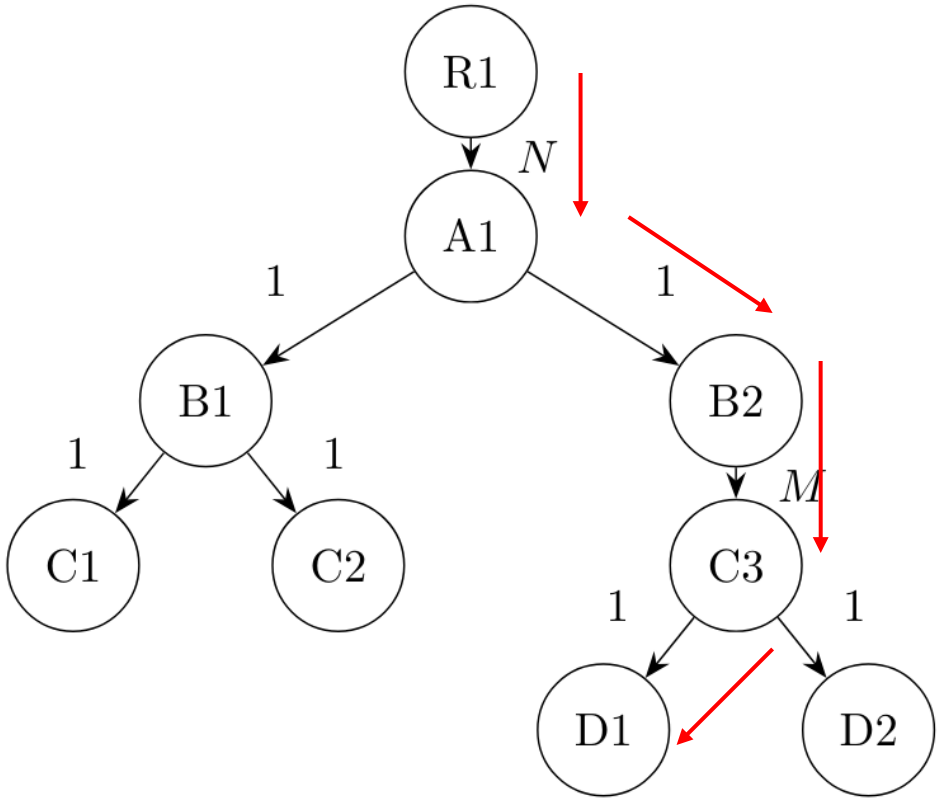
Structs in DaCe:

- DaCe supports structs. They can contain array of structs or arrays of native C types as members.
- Let's consider the following struct hierarchy:



Structs in DaCe:

- Struct-of-Arrays flattening iterates through all paths from the root to the leaves of the hierarchy
- This process determines the dimensions of the new arrays required



Structs in DaCe:

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

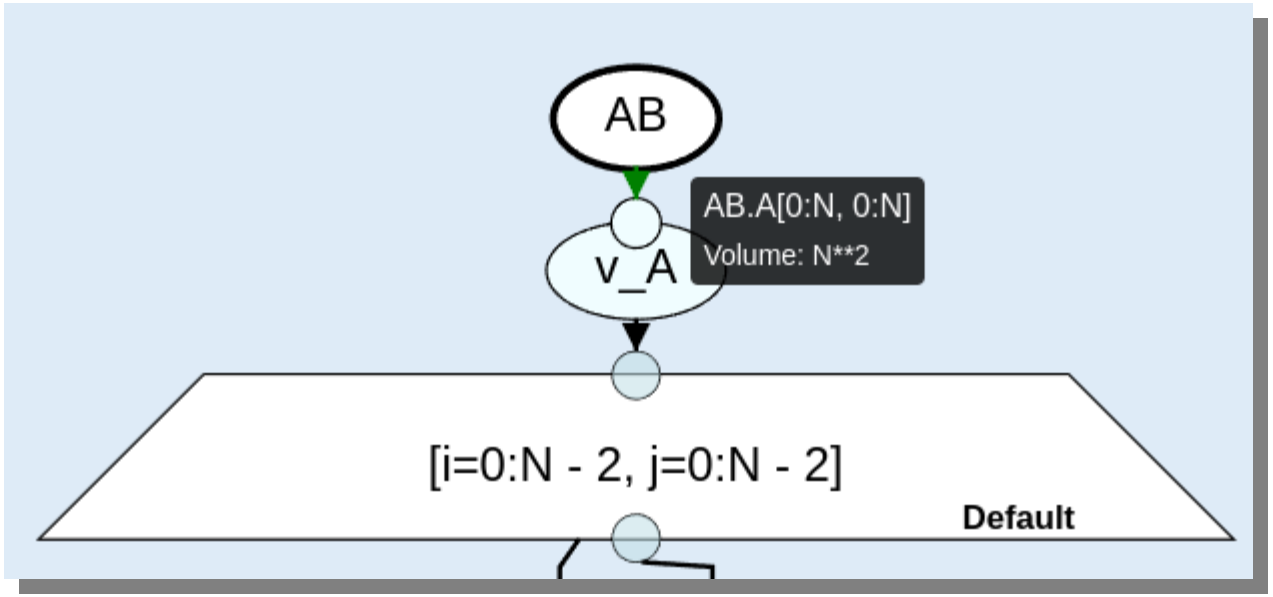
Container Name	Dimensions
__CG_R1__CA_A1_CG_B1__m_C1	N
__CG_R1__CA_A1__CG_B1__m_C2	N
__CG_R1__CA_A1__CG_B1__CA_C3__m_D1	N × M
__CG_R1__CA_A1__CG_B1__CA_C3__m_D2	N × M

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

Structs in DaCe:

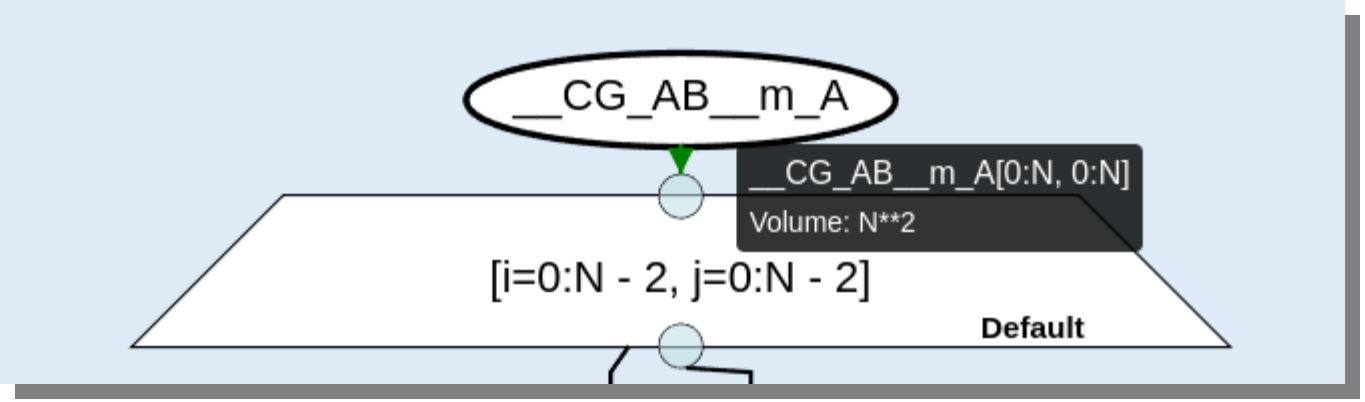
- Let's consider a simple SDFG that accesses a struct: A stencil kernel SDFG using a struct with two arrays for updates.
- Struct access in DaCe is facilitated through a *tower of views – one view per each member of struct array*.

```
1  struct AB {
2      float* A;
3      float* B;
4  };
5
```



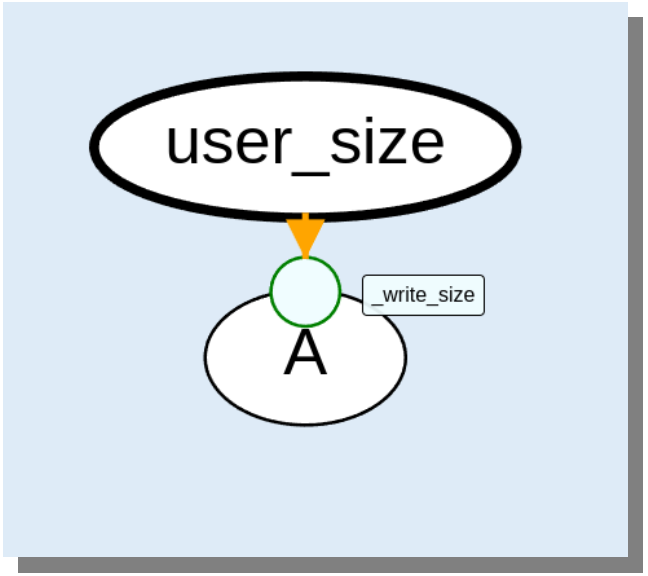
Structs in DaCe:

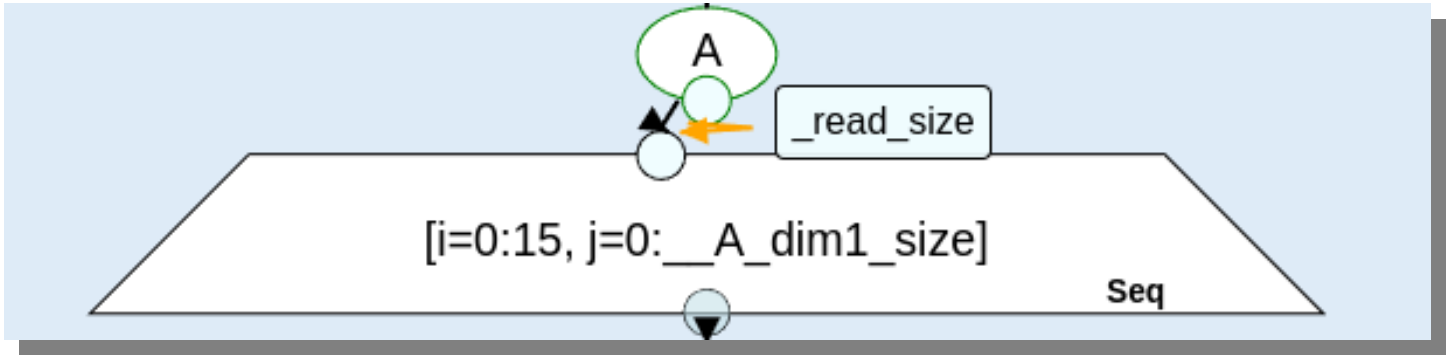
- The flattening pass creates an array of native C types for each member. (The length can be as small as one if it path from root to the leaf does not include any struct array)
- Replaces all struct accesses (using views) with accesses to arrays of C types (no views).
- Completely removes structs from the SDFG.



Deferred Allocation:

- This feature enables delayed (deferred) allocation of arrays during SDFG execution
- Allows allocation of arrays that are to be defined on a subset of the states of the SDFG. This is currently not possible due to scoping rules of DaCe.



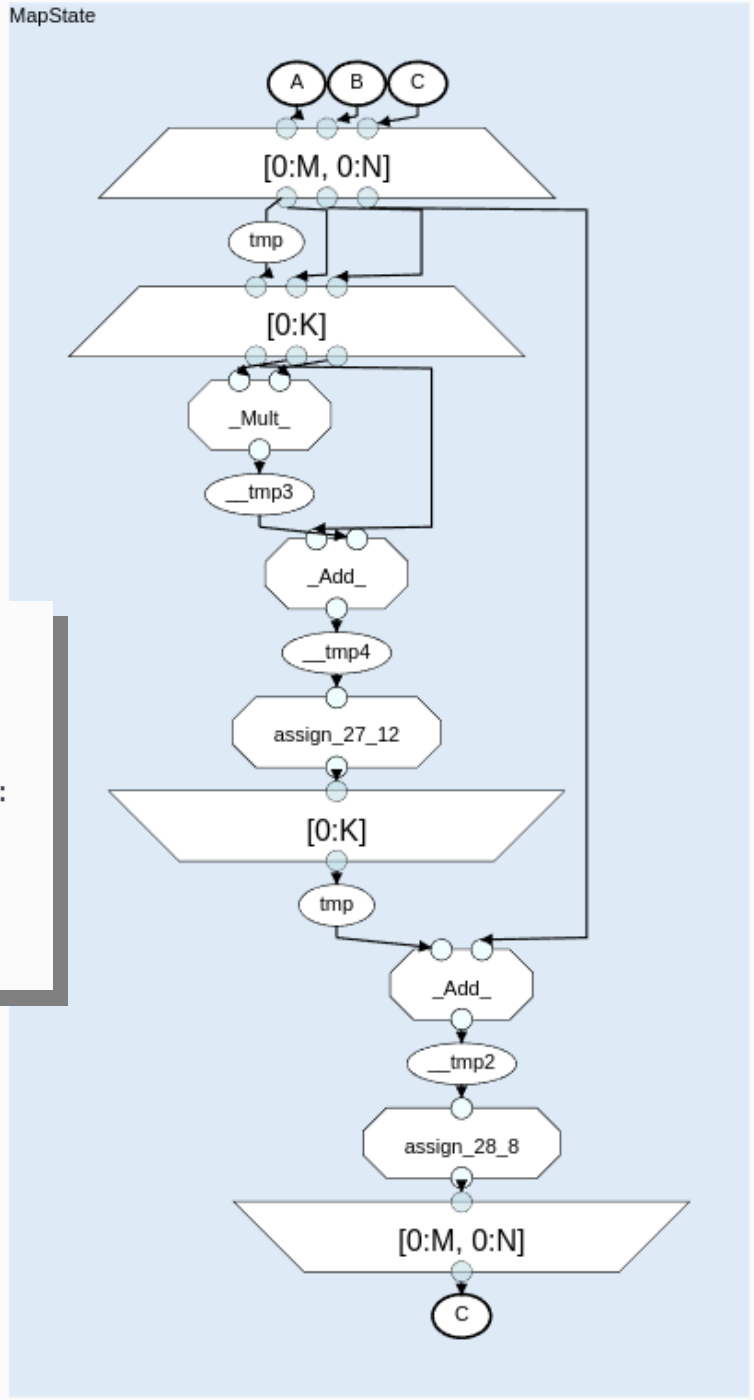


- **Tiling Transformations**

Current State of Tiling:

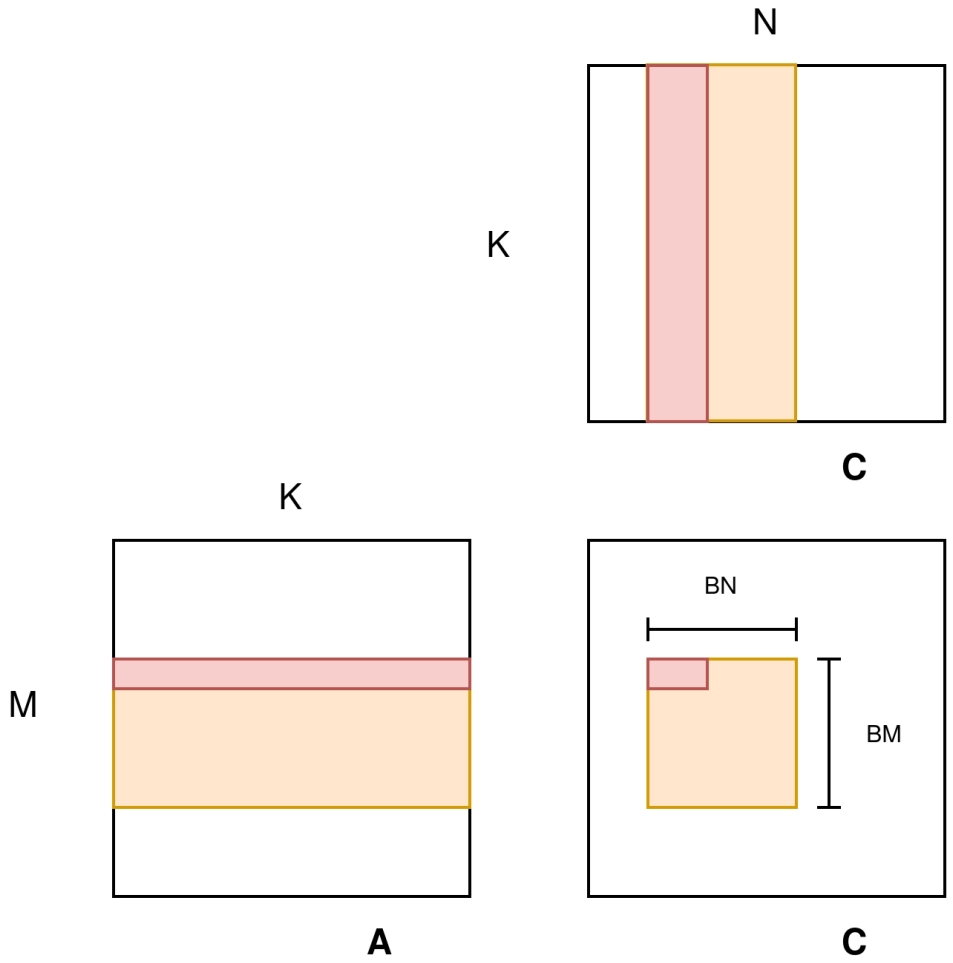
- The baseline SDFG of a GEMM derived from the following Python input:

```python
@dace.program
def gemm(A: dace.float32[M, K] @ dace.dtypes.StorageType.Ascend_Global,
         B: dace.float32[K, N] @ dace.dtypes.StorageType.Ascend_Global,
         C: dace.float32[M, N] @ dace.dtypes.StorageType.Ascend_Global):
    for i, j in dace.map[0:M, 0:N] @ dace.dtypes.ScheduleType.Ascend_Device:
        tmp = 0
        for k in dace.map[0:K] @ dace.dtypes.ScheduleType.Sequential:
            tmp = tmp + A[i, k] * B[k, j]
        C[i, j] = C[i, j] + tmp
```

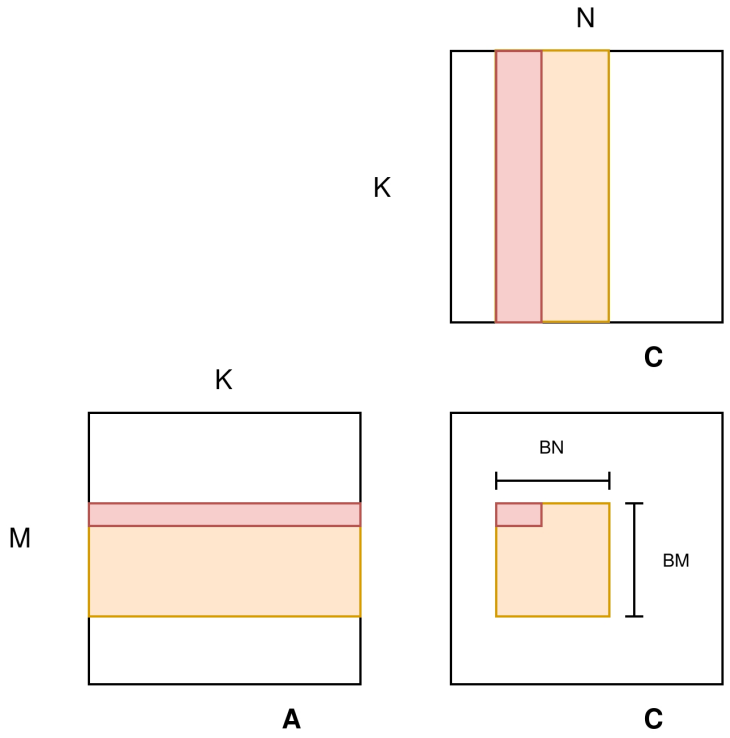


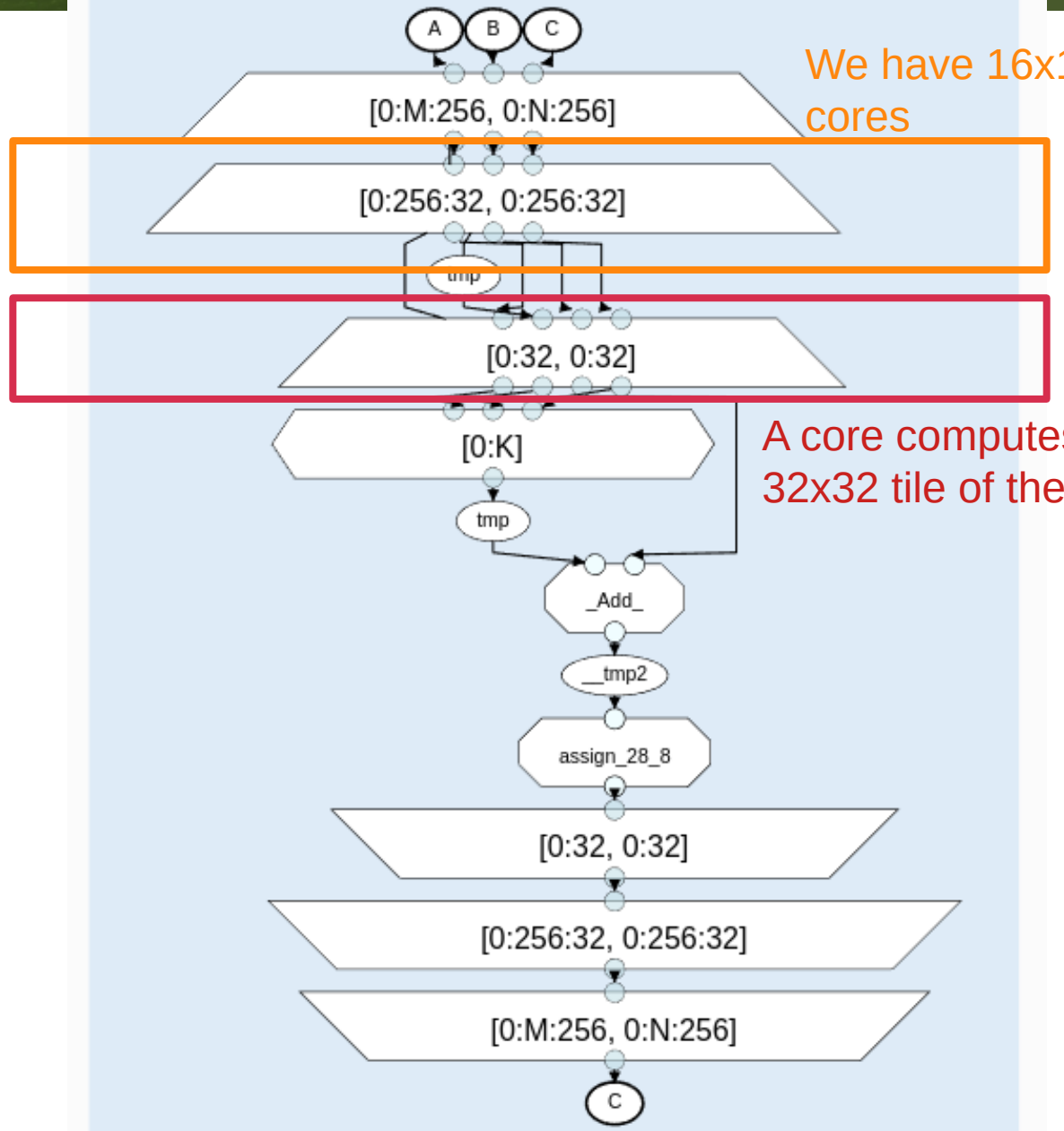
Current State of Tiling:

- The first transformation configures the layout of cores (e.g., 16x16 or 32x1 tiles, where each core computes a 1x1 cell of the output).
- The second transformation specifies the tiles (of the output, here matrix C) assigned to each core.



Current State of Tiling:



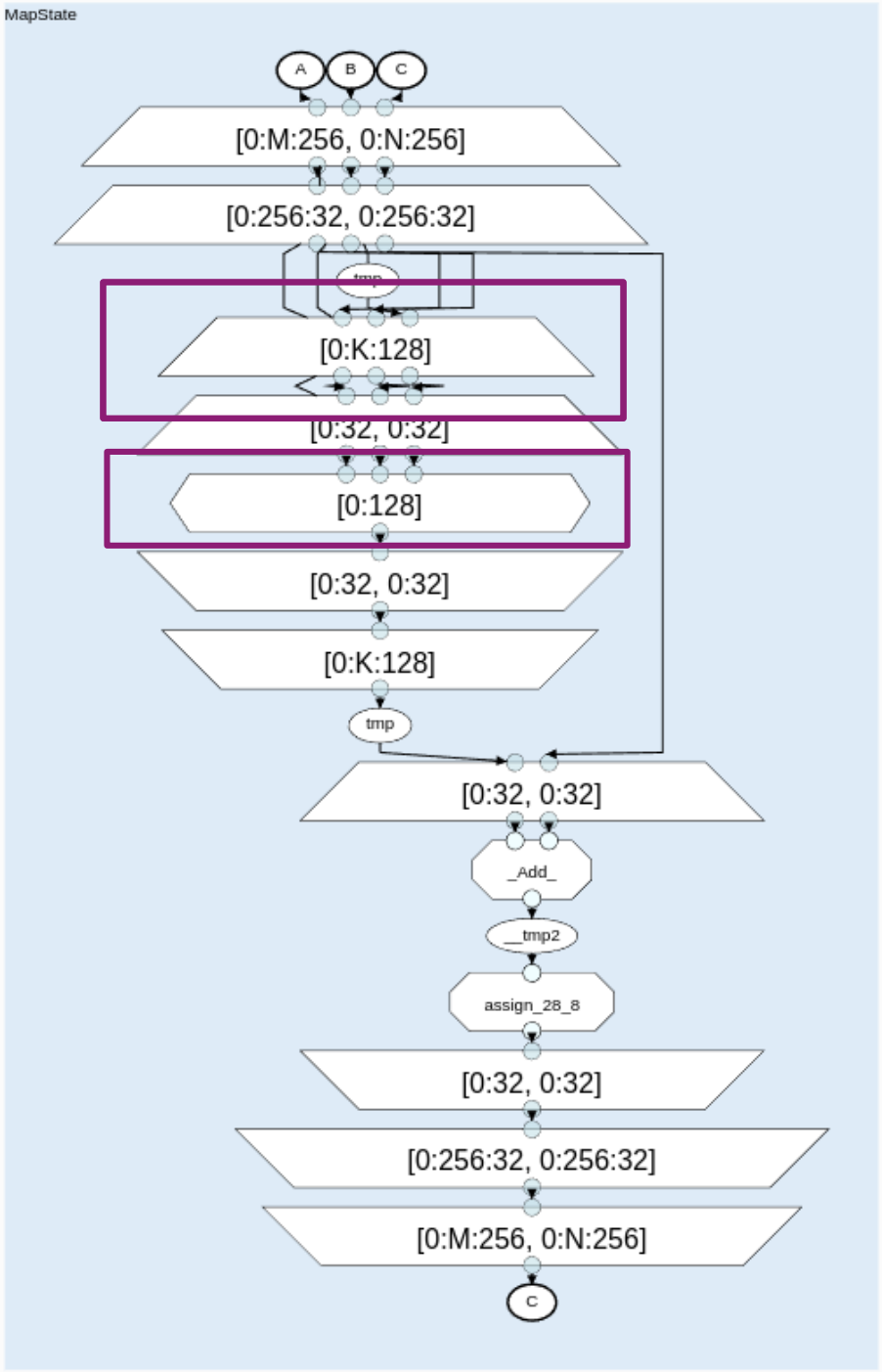


We have 16x16 cores

A core computes 32x32 tile of the output

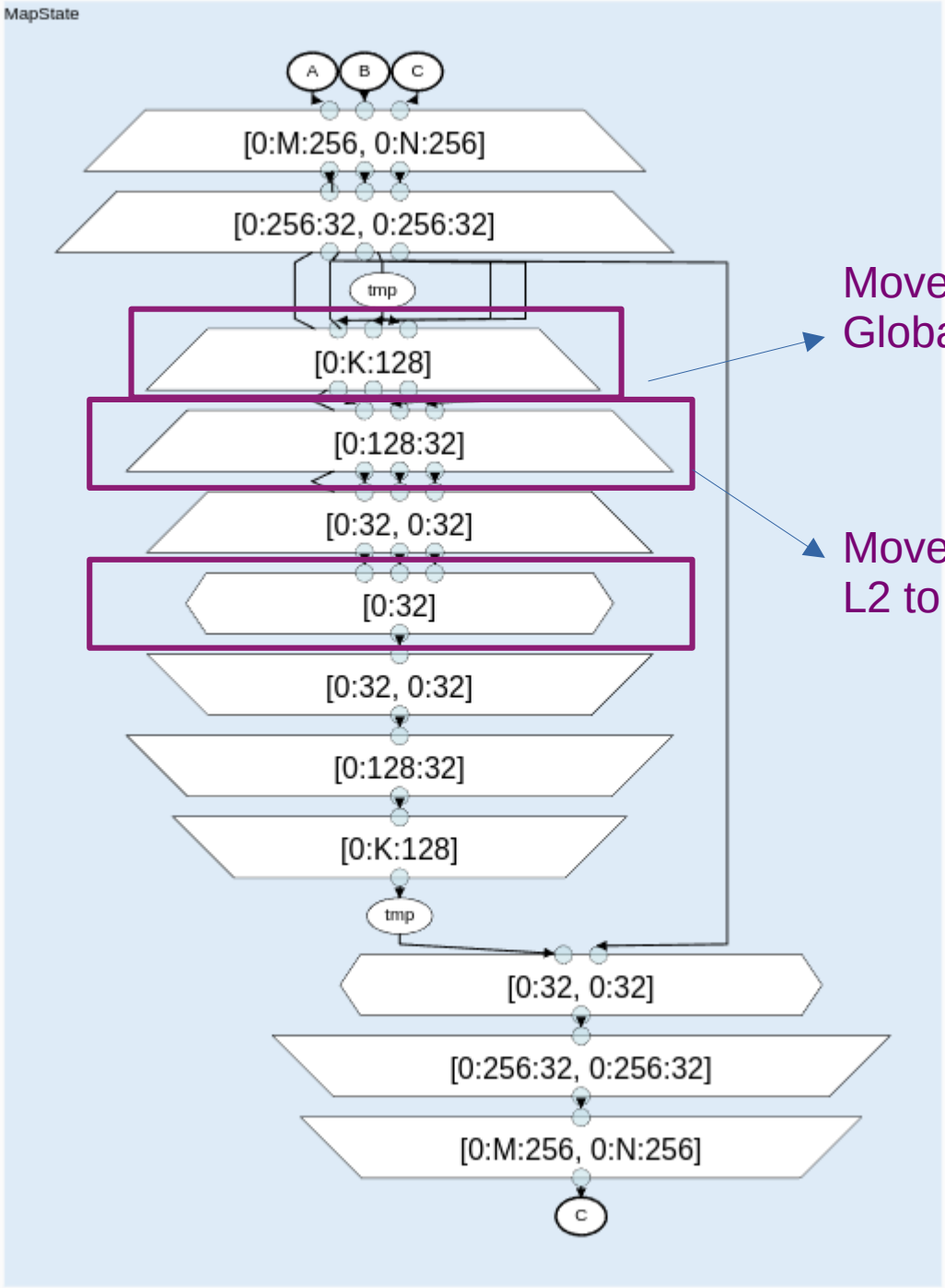
Current State of Tiling:

- Need to tile the work of a core to enable explicit memory movement. (Or better cache behavior)
- The work for a single core (reduction along the K-dimension in map *0:K:1*) is tiled into two maps ranging over *0:K:128* and *0:128:1*.



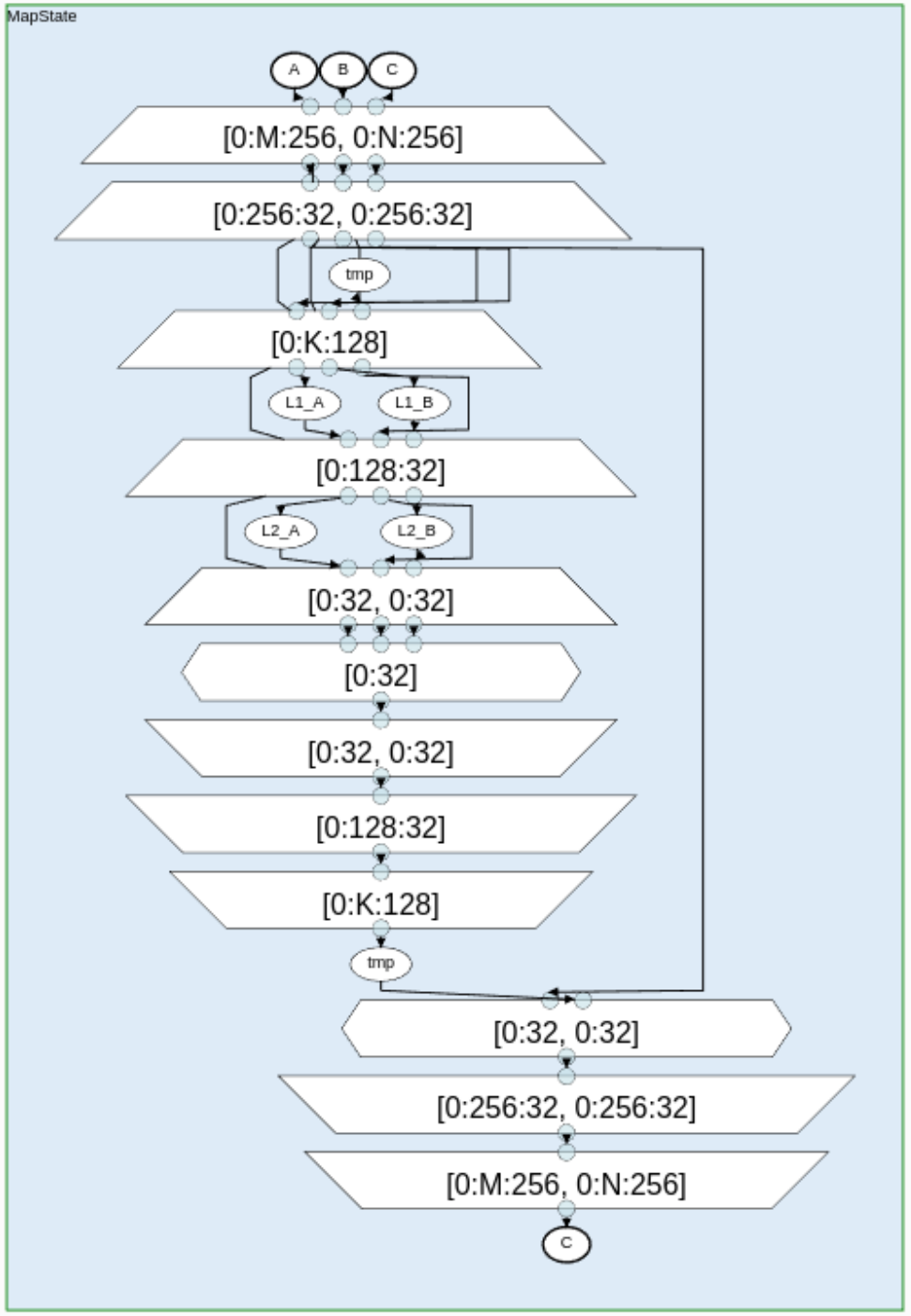
Current State of Tiling:

- Block tiling applied twice to support three levels of memory hierarchy; earlier issues have now been resolved since the last meeting.



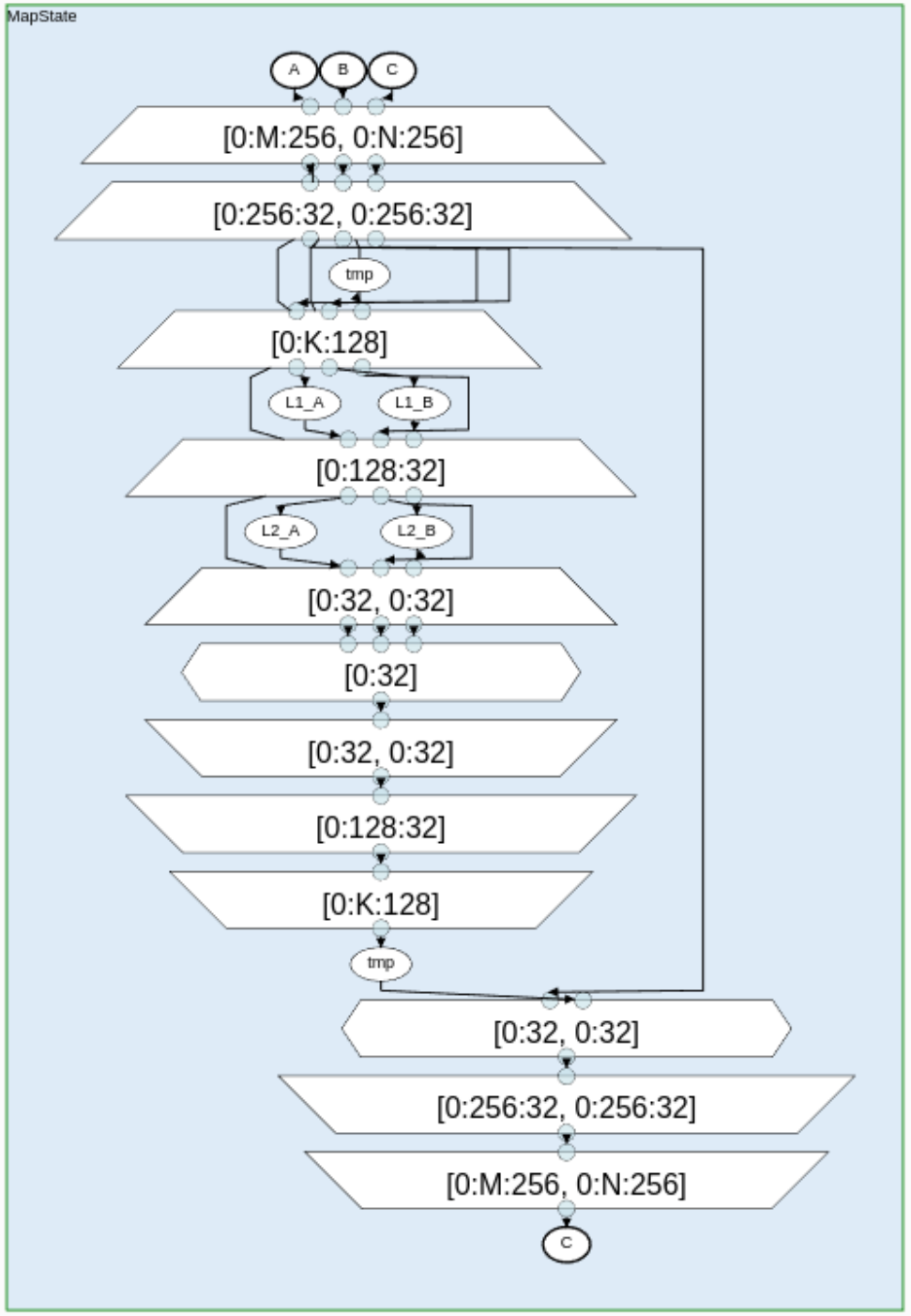
Current State of Tiling:

- Explicit memory movement transformations still face challenges.
  - Memory locations (e.g., A1, A2, B1, B2, CO1, CO2) need handling for input identification.
  - This may require extending the front-end with tag-based information or writing a pass that detects the purpose of GEMM inputsis from the SDFG.
- Needed features:
  - A pass to detect when memory needs to move between matrix and vector units.
  - Optionally, a pass to identify GEMM input purposes.



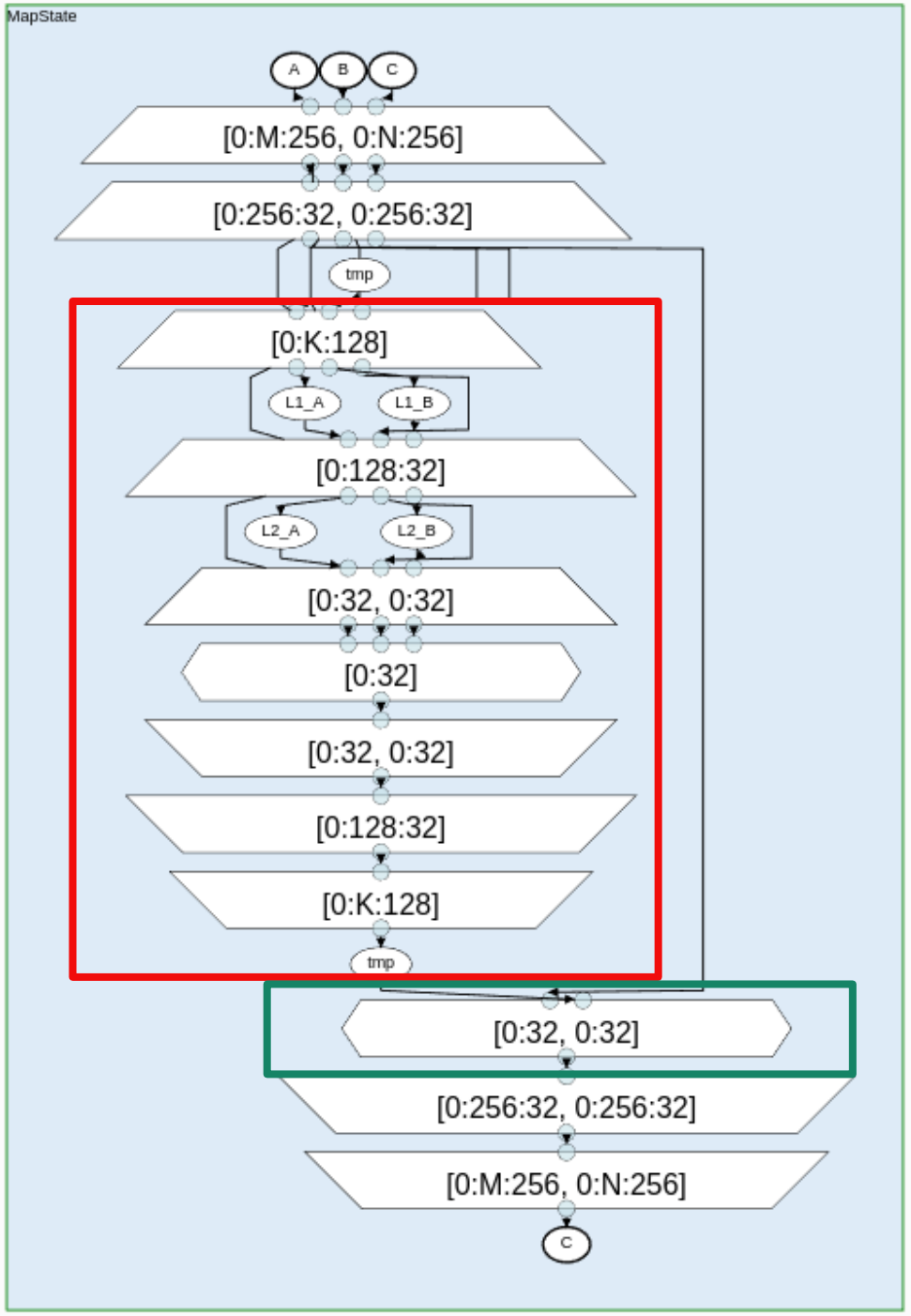
Current State of Tiling:

- Needed features:
  - A pass to detect when memory needs to move between matrix and vector units.
  - Optionally, a pass to identify GEMM input purposes.



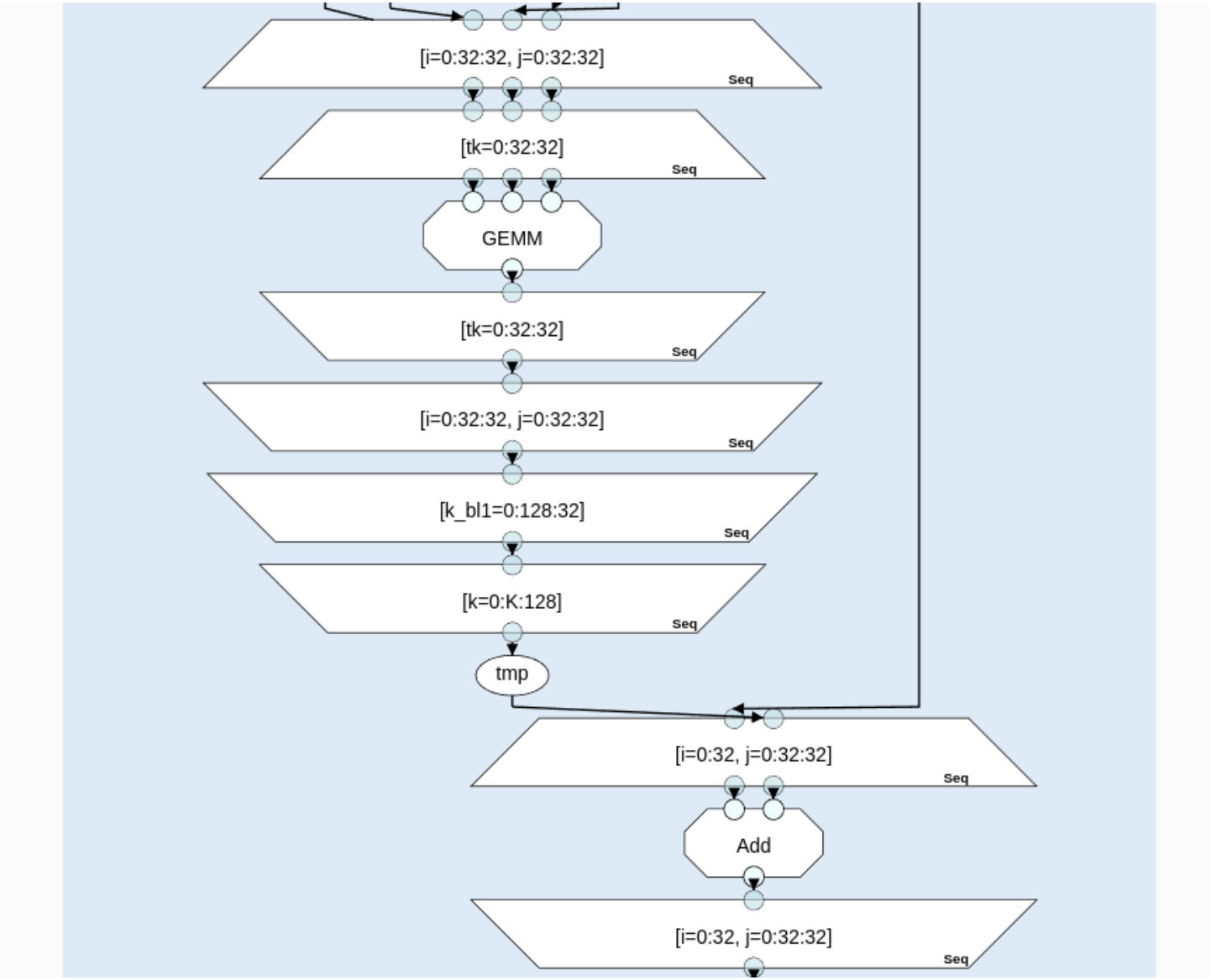
Current State of Tiling:

- In Red: Sub-SDFG that only contains tasklets for the vector unit
- In Green: Sub-SDFG that only contains tasklets for the vector unit
- Currently working on a pass that tags maps depending on which compute unit they use and adds the necessary memory movement nodes on the edges that connect two maps using different compute units.



Current State Of Tiling:

- Developed the pass, “ExplicitCubeUnitCall,” which detects inner-product GEMM or scalar patterns (Add, Sub, Div + Assign) and replaces them with cube unit or vector unit calls.
  - Requires users to define templates for how these units are invoked (e.g., useful for SoftHier support).
  - Pass an interval / set of possible sizes for the vector unit.
  - And which Python operators are support by the vector unit



Current State Of Tiling:

- Pass can be later used to detect which SDFGs can be mapped to an Ascend device.
- Currently only very primitive scalar patterns are supported. Will be extended as necessary
  - Supported patterns are:
    - C = A @ B for a tile
    - C = a + b (e.g., C = a + b + c + … + d is not supported)

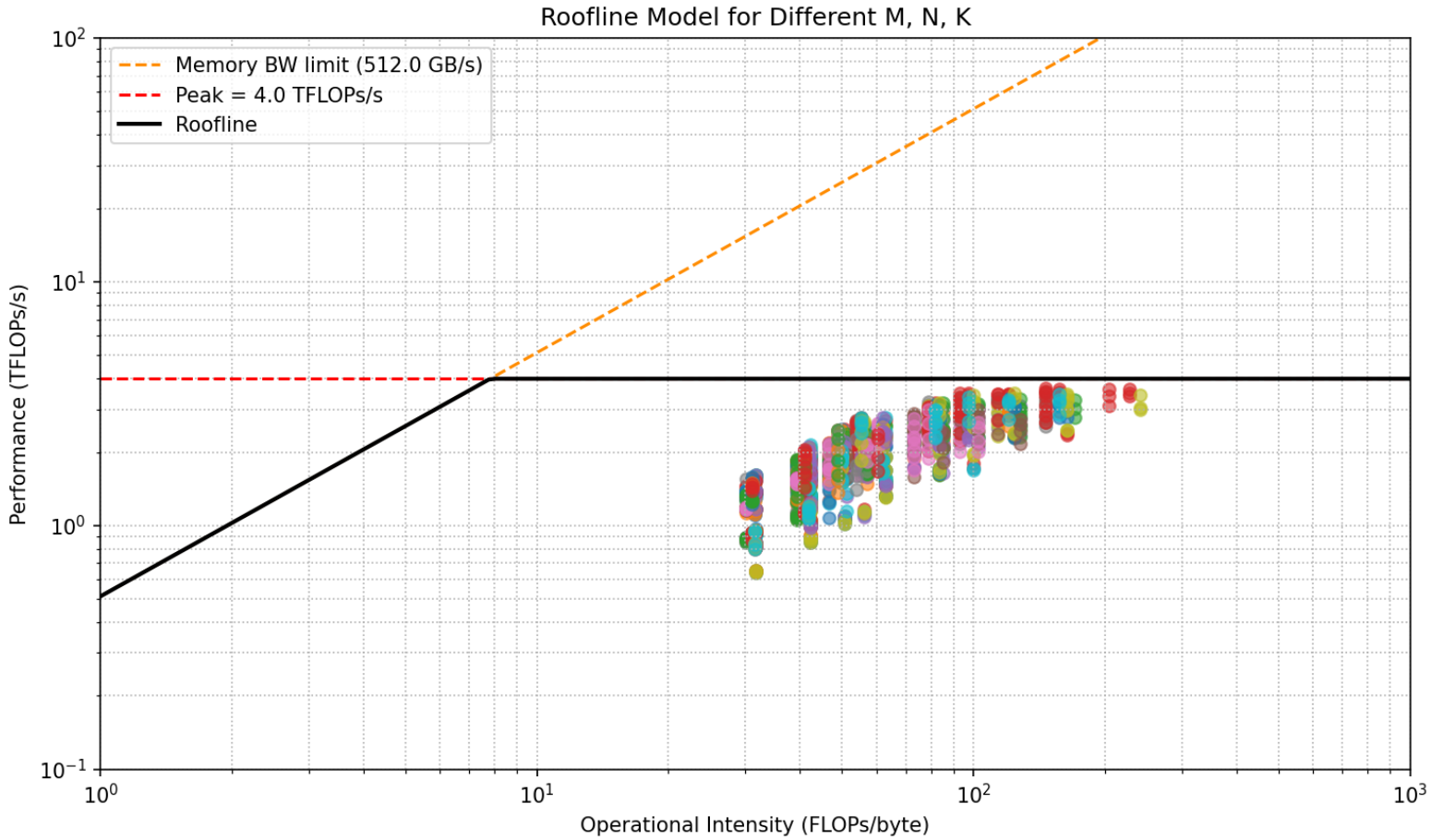
- **DaCe + SoftHier**

DaCe + SoftHier:

- Baseline GEMM SDFGs have been provided for SoftHier.
- To avoid delays in SoftHier development, a template has been created for a baseline GEMM SDFG. Aofeng started testing the SoftHier backend on this.
  - The template supports MMU and vector unit tasklets, the SDFG generation is based on a tiled-GEMM SDFG that assume scalar units only.
  - The baseline GEMM does not utilize on-chip interconnect

DaCe + SoftHier:

- SoftHier backend has generated code from the templated-SDFG:



- **Outlook For the Next Weeks**

Outlook / TODOs:

- Complete the ExplicitCubeUnitCall and ExplicitVectorUnitCall (for a subset of possible operators) passes.
- Develop a pass to detect the purpose of GEMM inputs (or receive it from the frontend).
- Merge the Flattening and Deferred Allocation features into DaCe's main branch.
- Work on passes that insert memory movement nodes between maps that use different compute units.
- Begin work on the backend support for the cube unit once transformations are finalized.