Yakup Koray Budanaz

SoftHier Progress Report March 17

Overiew of the Topics:

- Fixes to DaCe
- Transformation Updates

- **Fixes to DaCe**

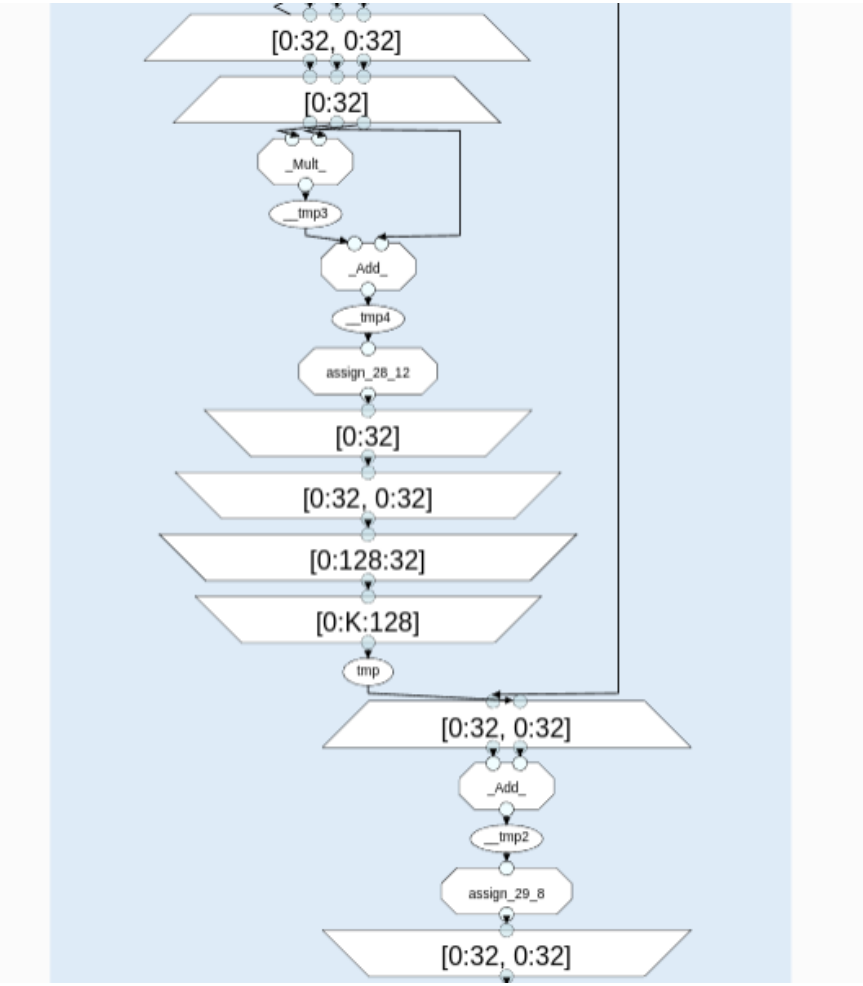
Fixes to DaCe

- I fixed a couple of unhandled-cases in the flattening transformation (3 issues)
- Fixed an issue in ArrayElimination pass (there are still more issues) (1 issue)
- DaCe requires cupy to be installed when running on GPUs, cupy does not work with ROCm, I have added PyTorch support to target AMD GPUs as a fallback mechanism. (1 feature)
- I have fixed multiple bugs in ExplicitMemoryMovement transformation (2 issues)
- I have fixed one issue in RemainderLoop transformation (1 issue)

- **Transformation Updates**

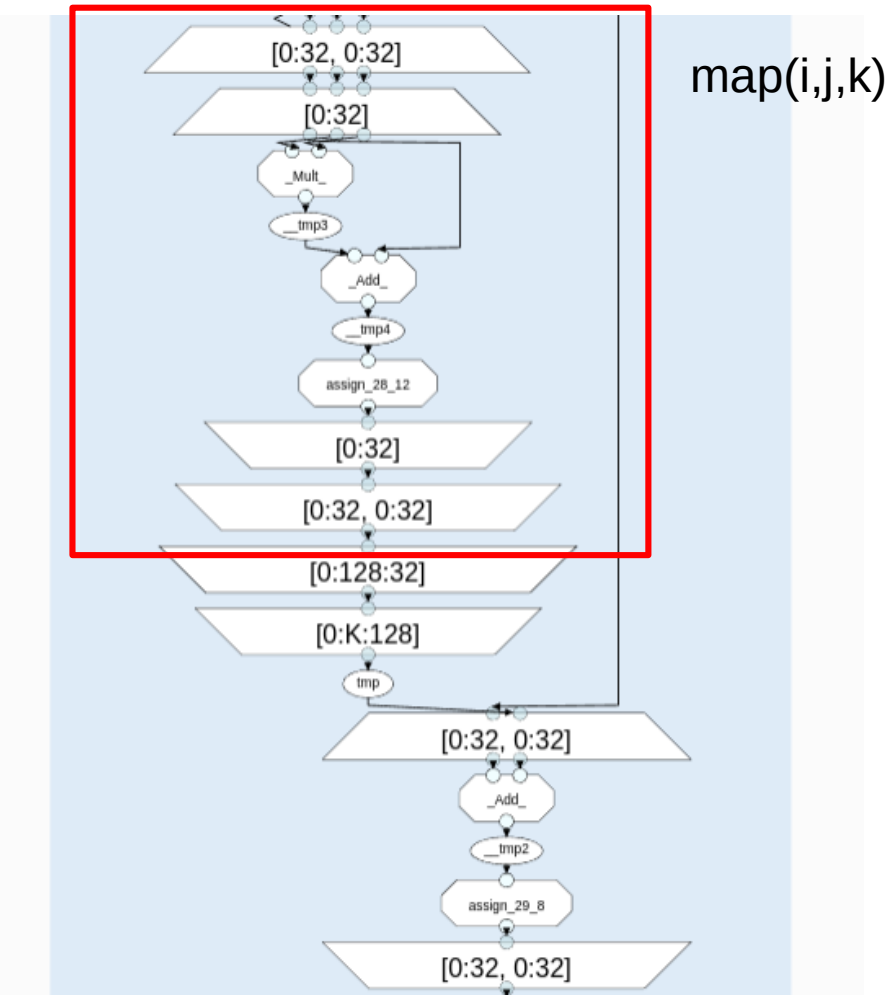
Transformation Updates

- Recap: pattern-matcher for tasklets for patterns arising in GEMM



Transformation Updates

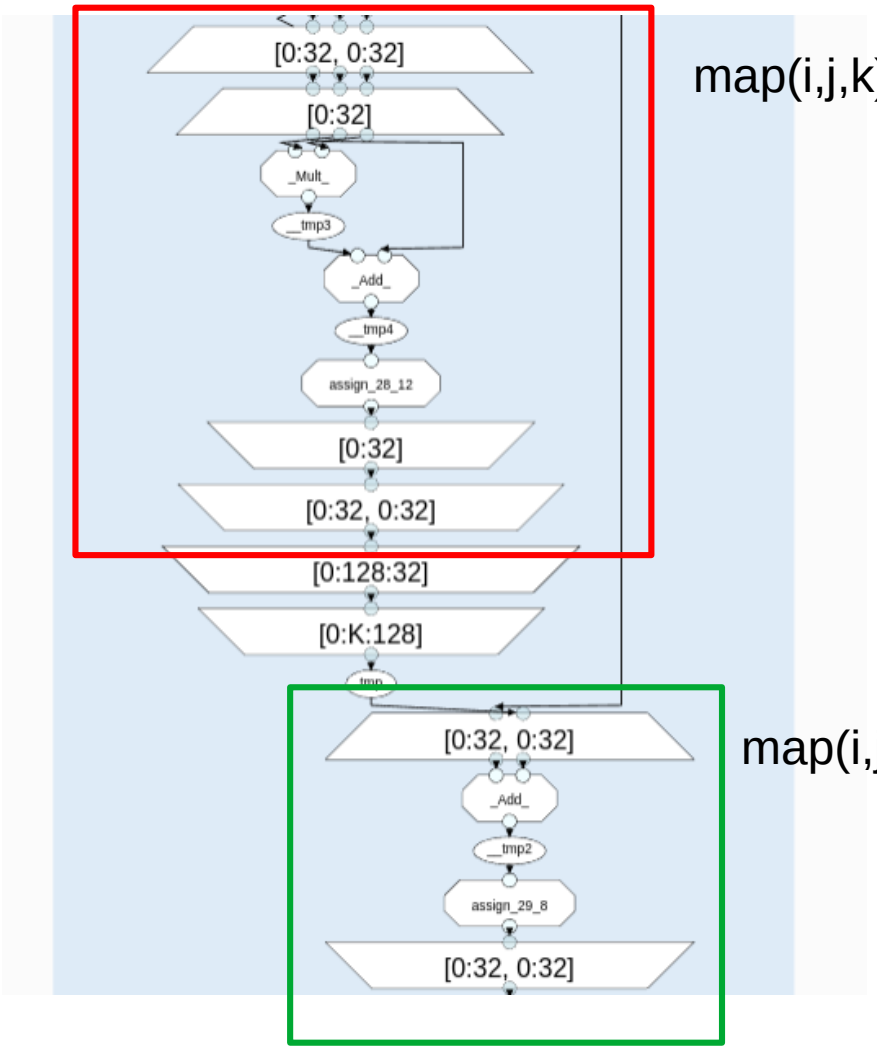
- Recap: pattern-matcher for tasklets for patterns arising in GEMM



map(i,j,k): c = c + mul(a[i,k],b[k,j]) → GEMM pattern

Transformation Updates

- Recap: pattern-matcher for tasklets for patterns arising in GEMM

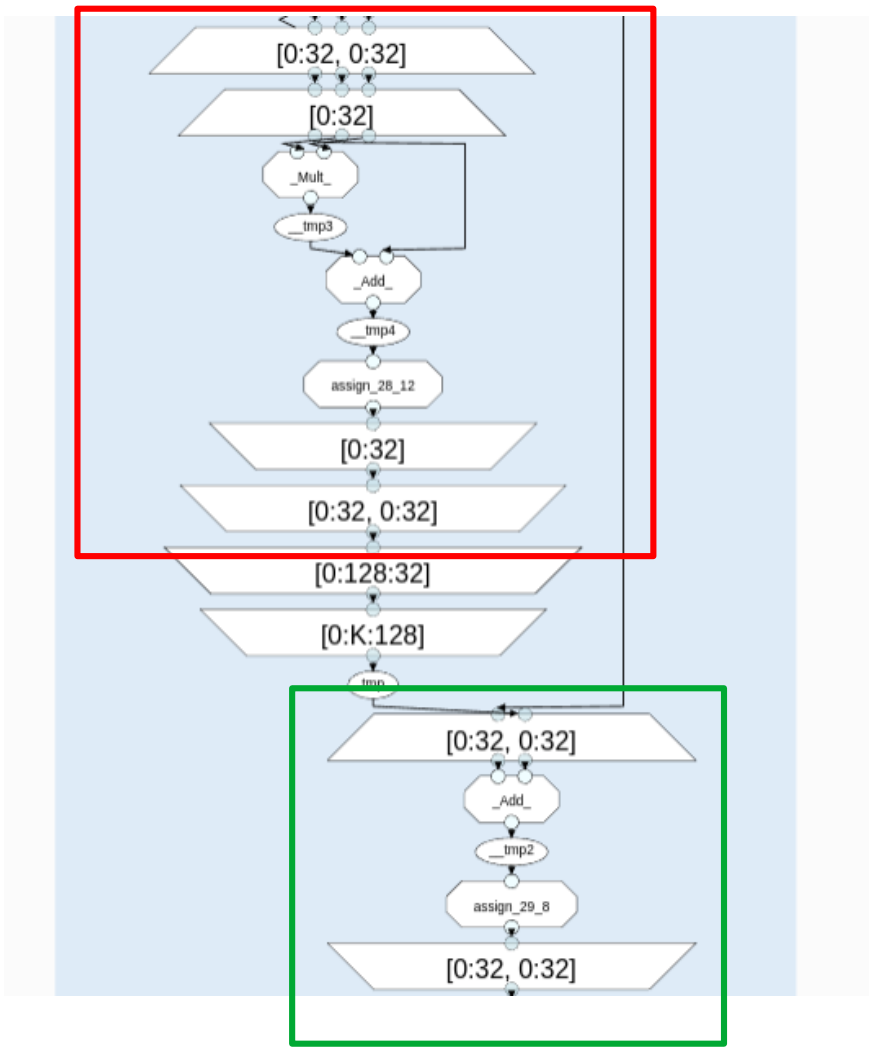


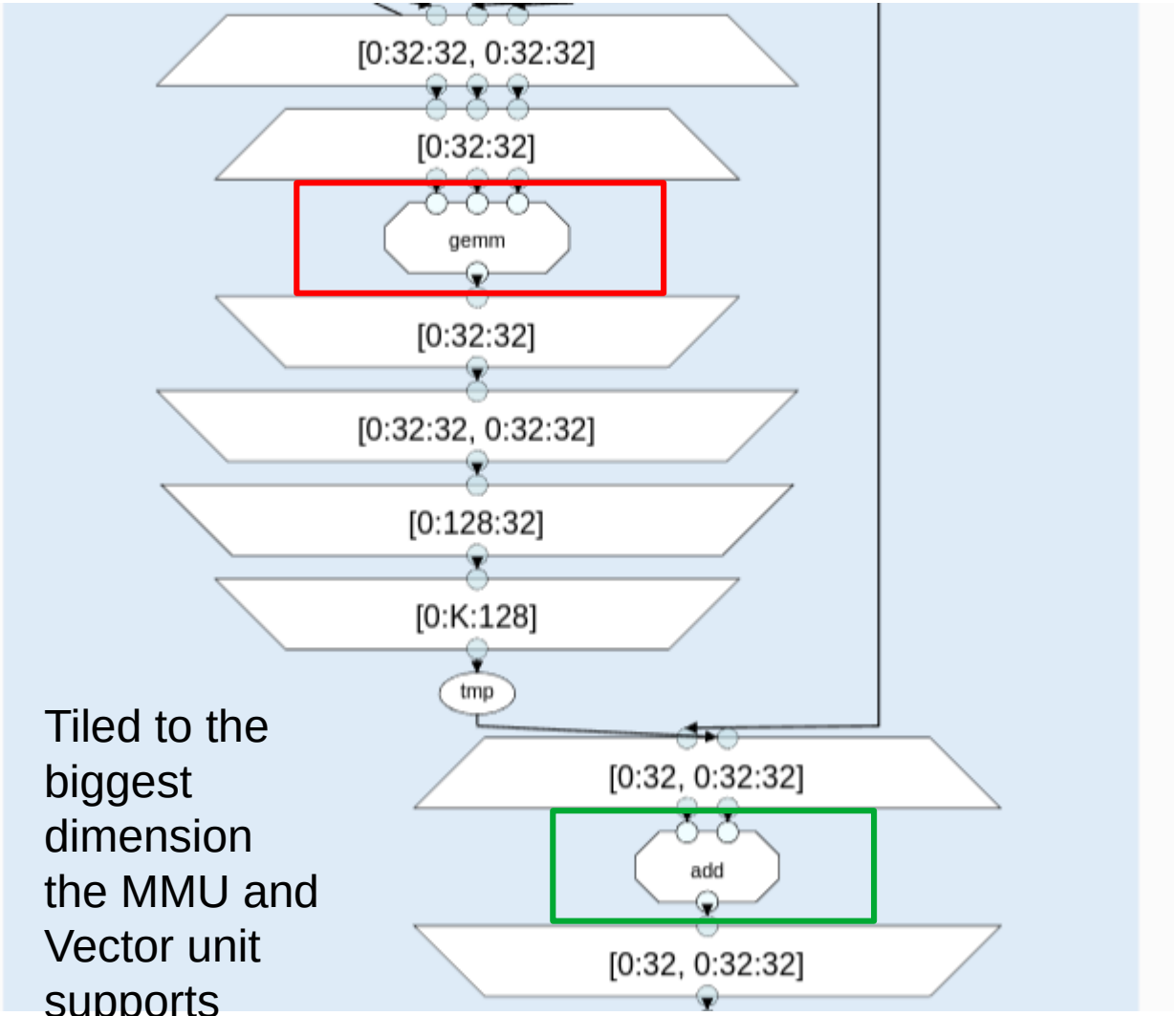
map(i,j,k): c = c + mul(a[i,k],b[k,j]) → GEMM pattern

map(i,j): c = c + mul(a[i,j],b[i,j]) → VECTOR pattern ADD

Transformation Updates

- Recap: pattern-matcher for tasklets for patterns arising in GEMM





Tiled to the biggest dimension the MMU and Vector unit supports

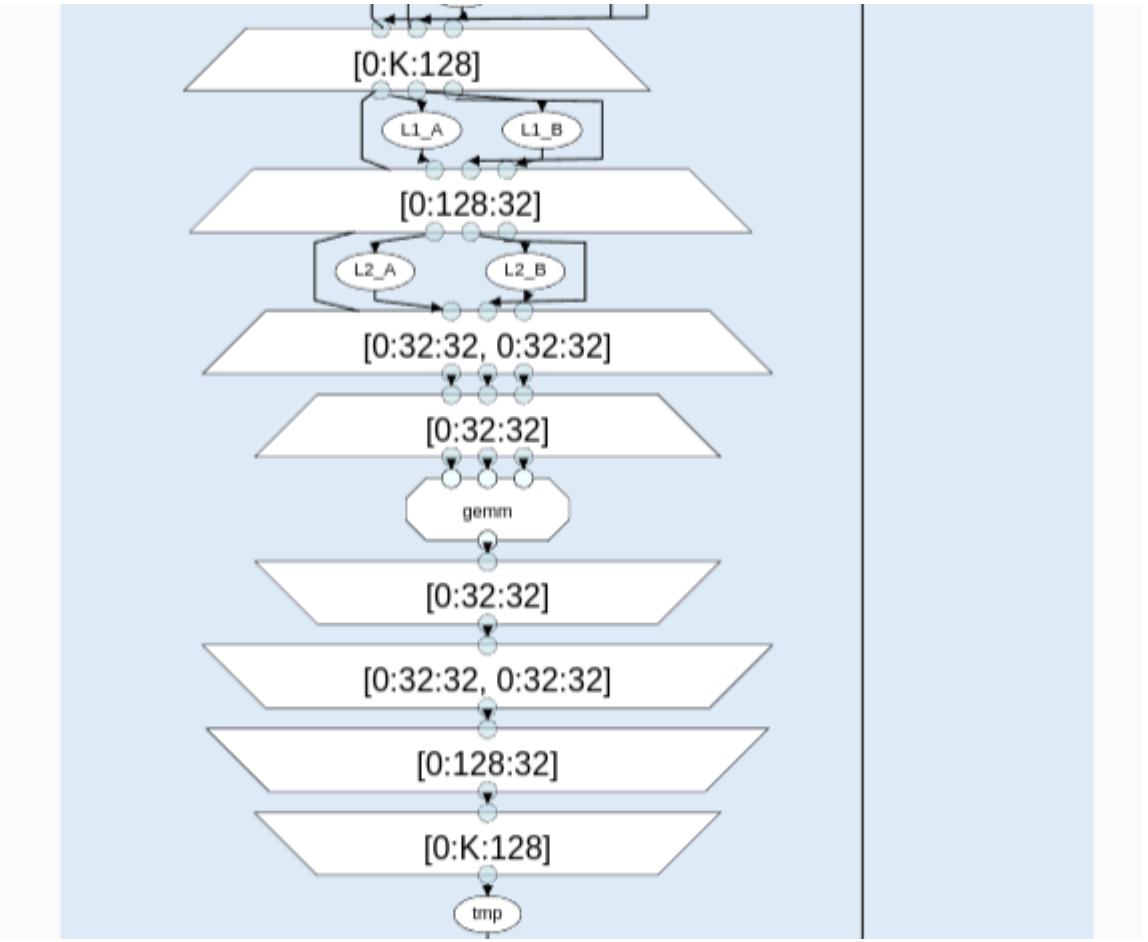
Transformation Updates

- Recap: Purpose detection. For GEMM the purposes are A, B, acc. Used to map generic location to specialized location. For example L2 to A2.
- Update: The format of purposes are concretized

- Purposes can be the set:
  - *{"acc", "A", "B"}*
- Purpose dict is saved as an attribute of the device map.

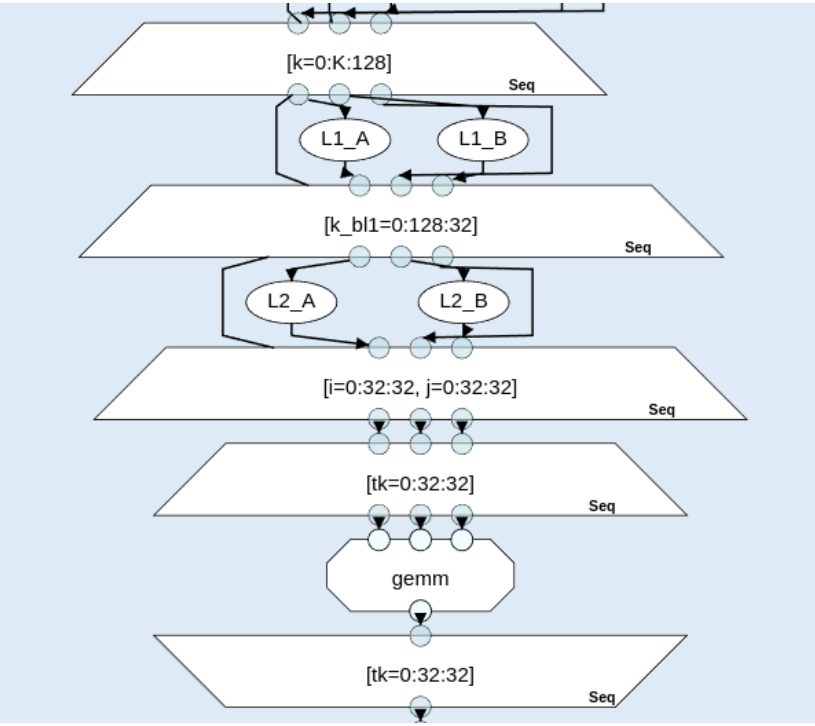
- Example purpose_dict
  - *{"tmp": "acc", "A": "A", "B": "B"}*

- I assume all data names to be the format:
  - *<LocationPrefix>_<ArrayName>*
  - *L2_A will map to purpose A.*

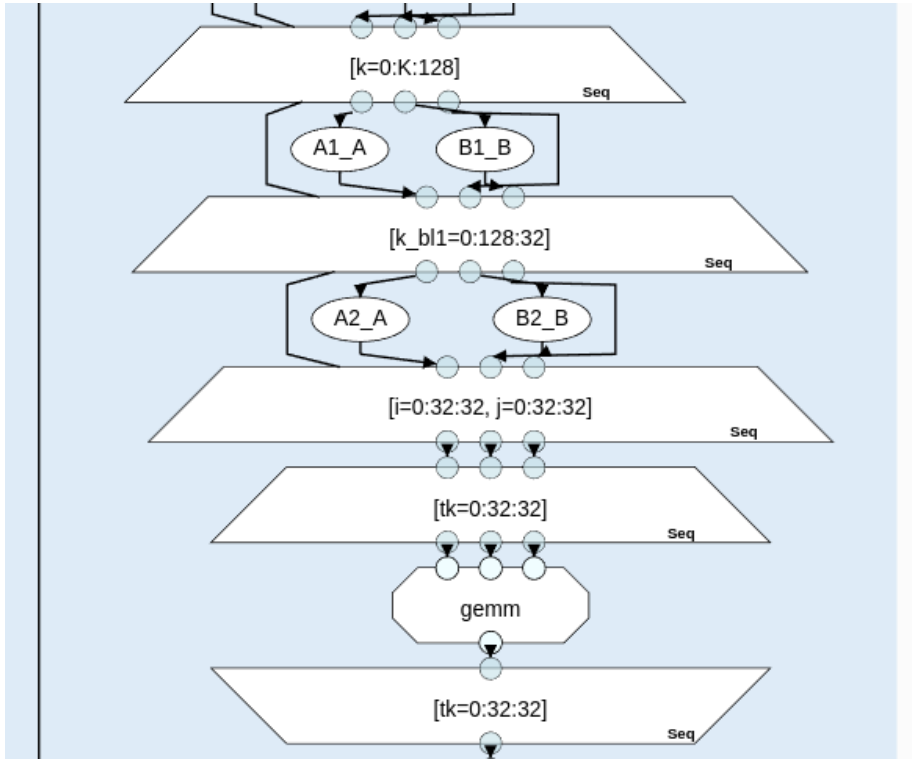


Transformation Updates

- Input Specialization Pass:



Purpose: *{"A": "A", "B": "B"}*
L1_A maps to level 1 and is specialized as *A1_A*.



Transformation Updates

- Insert Transfers Pass, Inputs:
- 1. All storage locations.
- 2. Graph of possible memory transfers with nodes of type *<unspecialized_storage>@<specialize_storage>* as all possible memory locations.

```
glb = str(dace.dtypes.StorageType.Ascend_Global)
a2 = str(dace.dtypes.StorageType.Ascend_L2) + "@" + str(dace.dtypes.StorageType.Ascend_A2)
b2 = str(dace.dtypes.StorageType.Ascend_L2) + "@" +  str(dace.dtypes.StorageType.Ascend_B2)
a1 = str(dace.dtypes.StorageType.Ascend_L1) + "@" +  str(dace.dtypes.StorageType.Ascend_A1)
b1 = str(dace.dtypes.StorageType.Ascend_L1) + "@" +  str(dace.dtypes.StorageType.Ascend_B1)
co2 = str(dace.dtypes.StorageType.Ascend_L2) + "@" +  str(dace.dtypes.StorageType.Ascend_CO2)
co1 = str(dace.dtypes.StorageType.Ascend_L1) + "@" +  str(dace.dtypes.StorageType.Ascend_CO1)
vecin = str(dace.dtypes.StorageType.Ascend_VECIN)
vecout = str(dace.dtypes.StorageType.Ascend_VECOUT)
nodes = [glb, a2, b2, a1, b1, co2, co1]
finf = float('inf')
graph = {
    glb: { glb: 0, a2: 1, b2: 1, a1 : finf, b1: finf, co2 : finf, co1: finf, vecin: 1, vecout: finf },
    a2: { glb: finf, a2: 0, b2: finf, a1 : 1, b1: finf, co2 : finf, co1: finf, vecin: finf, vecout: finf },
    b2: { glb: finf, a2: finf, b2: 0, a1 : finf, b1: 1, co2 : finf, co1: finf, vecin: finf, vecout: finf },
    a1: { glb: finf, a2: finf, b2: finf, a1 : 0, b1: finf, co2 : 1, co1: finf, vecin: finf, vecout: finf },
    b1: { glb: finf, a2: finf, b2: finf, a1 : finf, b1: 0, co2 : 1, co1: finf, vecin: finf, vecout: finf} ,
    co2: { glb: finf, a2: finf, b2: finf, a1 : finf, b1: finf, co2 : 0, co1: 1, vecin: finf, vecout: finf },
    co1: { glb: 1, a2: finf, b2: finf, a1 : finf, b1: finf, co2 : finf, co1: 0, vecin: 1, vecout: finf },
    vecin: { glb: 1, a2: finf, b2: finf, a1 : finf, b1: finf, co2 : finf, co1: finf, vecin: finf, vecout: 1 },
    vecout: { glb: 1, a2: finf, b2: finf, a1 : finf, b1: finf, co2 : finf, co1: finf, vecin: finf, vecout: finf }
}

```

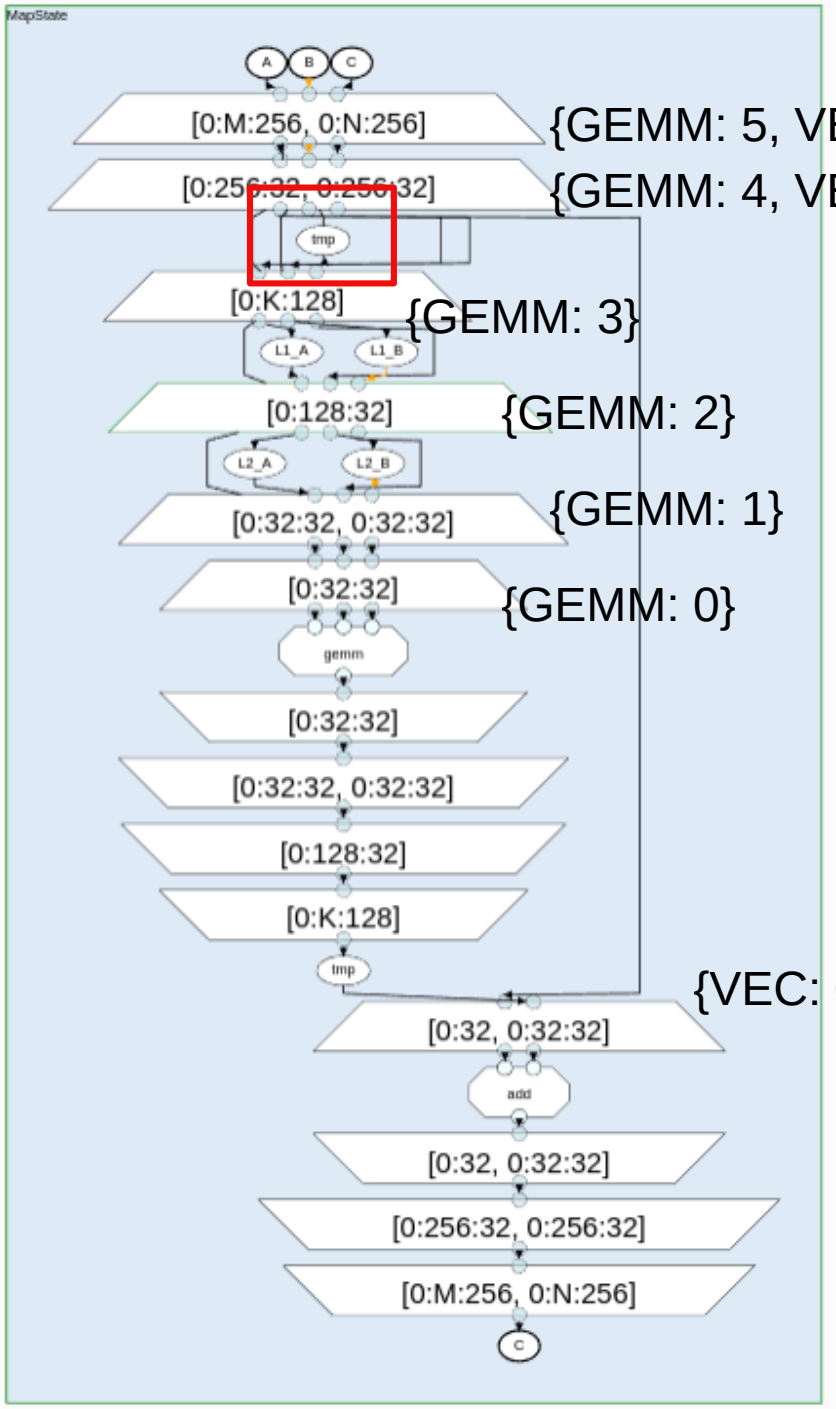
Transformation Updates

- Insert Transfers Pass, Inputs:
- 3. *EntryLocationRequirements* for each compute unit (Where the data needs to be for the compute unit)
- 4. *ExitLocationRequirements* (where the tasklet puts the memory out)
- 5. *in_out_types* is the same thing for tasklets (MMU takes *A2* and *B2* as inputs and outputs to CO2, VECTOR takes *VECIN* and outputs *VECOUT*)
- 6. *ComputationalUnitsRegisterLocations* is where one needs to map the *Register* storage depending on the compute unit. (*Register* is storage the scalar unit can use)
- 7. *UnspecializedLocations* are used to derive specialized locations from level names.

```
entry_location_requirements = {
    "MMU": [a1, b1,],
    "VECTOR": [vecin,],
}

exit_location_requirements = {
    "MMU": [co2,],
    "VECTOR": [vecout,],
}

computational_unit_register_locations =  {
    "MMU": co2,
    "VECTOR": vecin,
}

in_out_types = {
    vecin: vecout,
    a2 + "_AND_" + b2: co2
}

l1 = str(dace.dtypes.StorageType.Ascend_L1)
l2 = str(dace.dtypes.StorageType.Ascend_L2)
unspecialized_locations = [l1, l2]
```

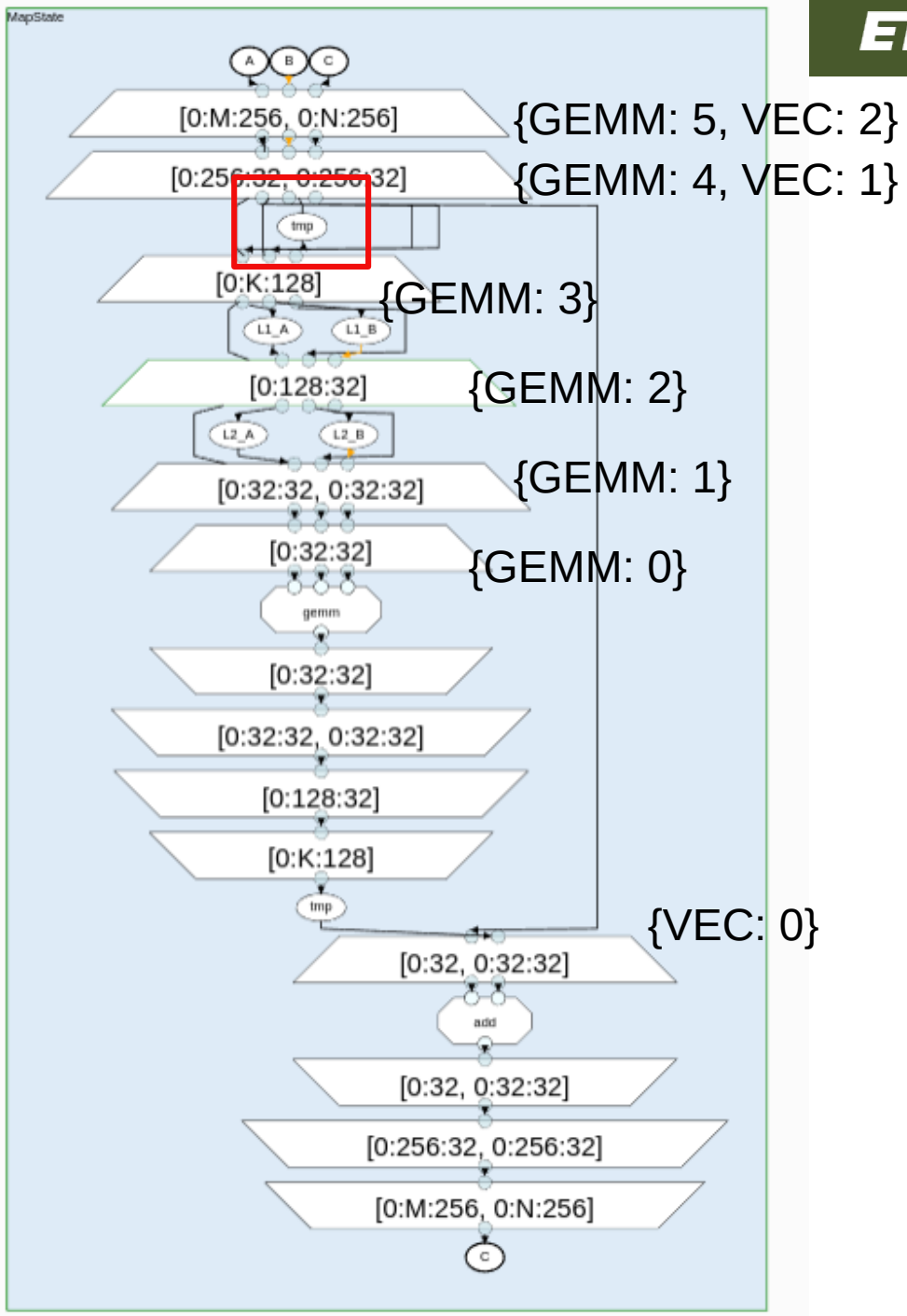
Transformation Updates

- Insert Transfers Pass - Step1:
- Map all existing arrays with *Register* storage to the storage needed by the first tasklet they have been used in.
  - For MMU/GEMM this is *CO2*, for Vector this is *VECIN*.
- tmp is used in GEMM first, the first output from a tmp access node directs to *{GEMM:3}* second output from a tmp access node goes to *{VEC:0}* therefore it is mapped to *CO2*.



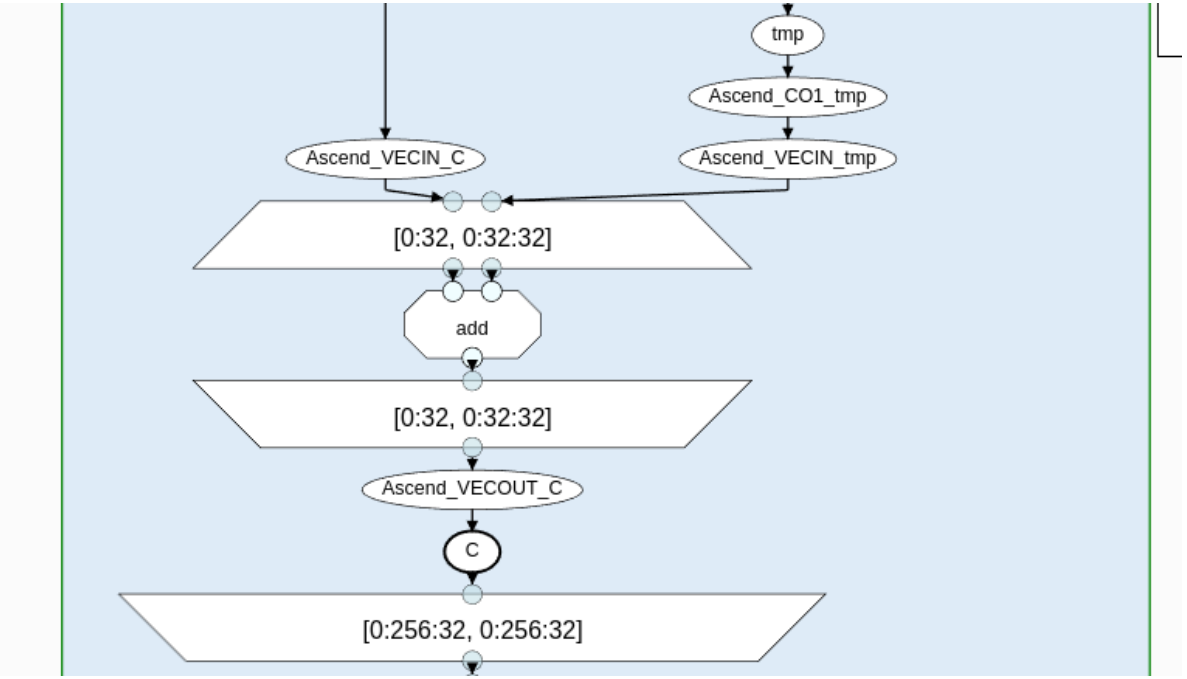
Transformation Updates

- Insert Transfers Pass - Step2:
- Create shortest paths between each node of the *MemoryMovementGraph*.
- For all input edges to MapEntry nodes,
  - If the input data's distance to computational unit's desired storage type is higher than the distance of the map, insert transfers as long as the distance is same.
- This approach only works on maps that have one type of computational unit used. (e.g. no transfers in *{GEMM:4 , VEC:1}*)



Transformation Updates

- Example insertion: the pass then inserts the movements to addition map:
  - For C: *Global* → *VECIN*
  - For tmp: *CO2* → *CO1* → *VECIN*
- Step 3:
  - For each map where an input was inserted insert the corresponding output.
    - Add a VECOUT to map exit. (As many movements we need to reach the storage type of C)



Transformation Updates

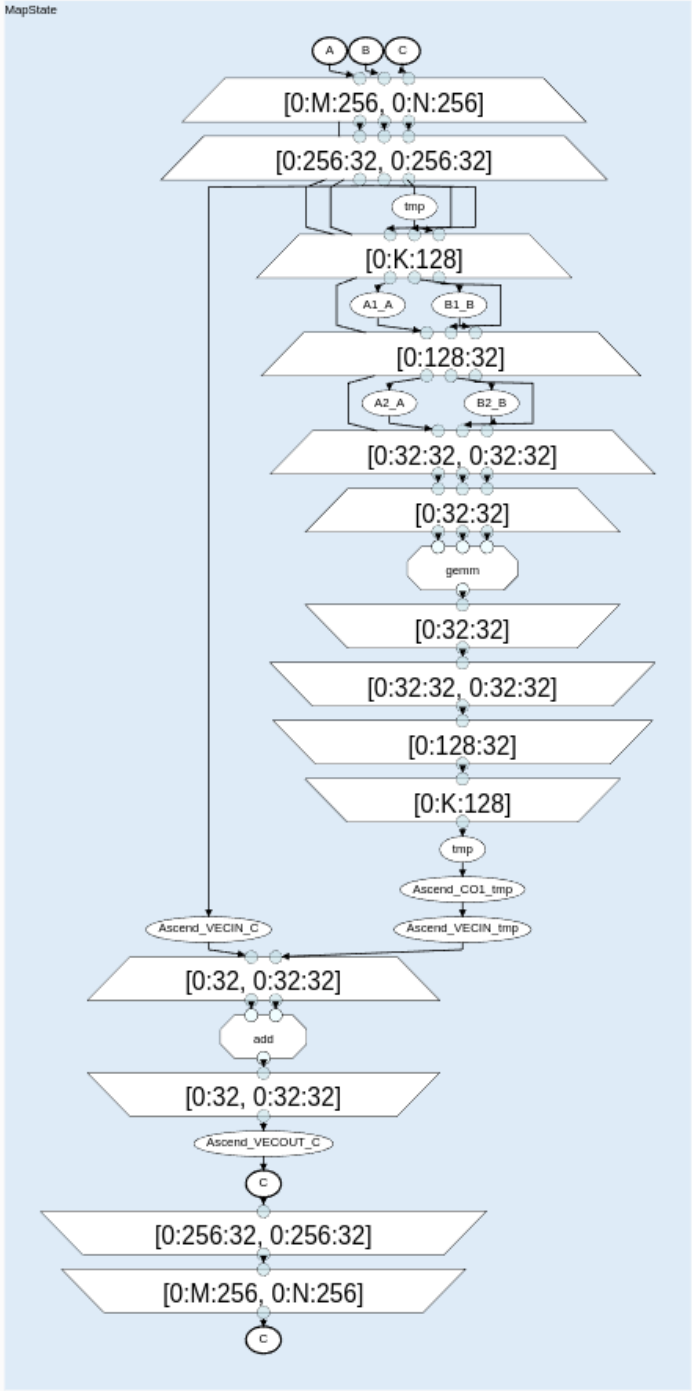
Output looks like this :)

1. I need to clean-up the pass.

2. I need to integrate the *ExplicitMemoryMove* transformation into the *InsertTransfer* pass

3. I need to update *RemainderLoop* transformation to support the graph seen on the right.

4. I need to extend the pass to not have implicit assumptions but explicitly check them.



Transformation Updates

- The pass comes with many assumptions:
- A map can contain one tasklet.
  - For a second tasklet a new map scope is necessary.
- It assumes one can directly write from a tasklet output to global with enough transfers and implements that, might not be the most efficient.
- Currently the cases for 910A are: VECOUT → Global, or CO2 → CO1 → Global
- I also need to generalize the *ScalarToMMU* and *ScalarToVector* to pass to support different tasklet strings.
  - I plan to get and end-to-end transformation pipeline that takes a device-agnostic SDFG and returns a tiled SDFG for Ascend devices next.

- **Outlook for the Next Weeks**

Outlook for the Next Weeks

- Due to the SC deadline I have shifted my focus to other projects.
  - I will continue my work on the transformation and but might need to allocate more time to other projects.
- I will continue improving the transformation and fixing bugs and issues in DaCe.
- I want to test the quality and portability of the transformations by applying them on CUTLASS backend.
  - The memory locations needed in CUTLASS are the similar to Ascend.
  - It is a good chance to test layout transformation as data is moved (e.g., CUTLASS permuted layout)
- Get and end-to-end transformation pipeline that takes a device-agnostic SDFG and returns a tiled SDFG for Ascend devices.