Yakup Koray Budanaz

SoftHier May 19 Status Update

Overiew of the Topics:

- Fixes for AscendC Build Pipeline And Vector Unit
- New Strategy For Generating Copies
- Current Status of CubeUnit Codegen and Current Issues

Work on the AscendC Backend:

- For performance numbers I refer to: https://cset.georgetown.edu/publication/pushing-the-limits-huaweis-ai-chip-tests-u-s-export-controls/
- I see HBM capacity is 32 GB on the server there I assume, the server 910B has 800 GB/s HBM bandwidth.

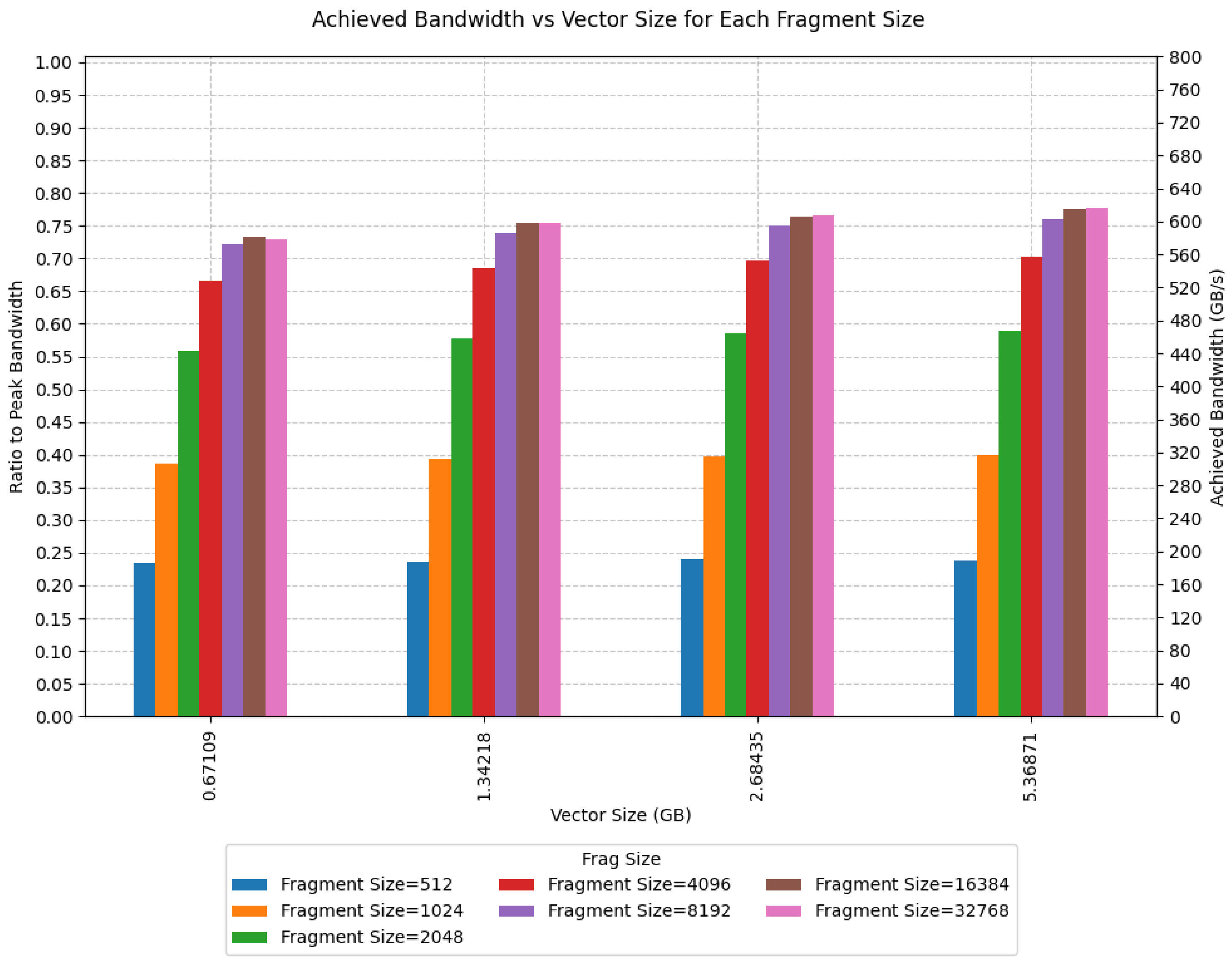
	Ascend 910 series (first-generation)	Ascend 910B series (second-generation)
Launch Date	Mid 2019	Late 2022
Performance (FP16 TFLOPS)	220 - 320	280 - 400
Total Number of AI Cores	32	?
Number of Active AI Cores	30 - 32	20 - 25
Clock Speed	0.9 - 1.15 GHz	1.65 - 1.85 GHz
Maximum On-chip Memory	76 MB	211 MB
HBM Type	HBM2	HBM2e
HBM Bandwidth	1228 GB/s	800/1600 GB/s
HBM Capacity	32 GB	32/64 GB
Fabrication	TSMC 7nm (N7+)	SMIC 7nm (N+2)

CSET CENTER for SECURITY and EMERGING TECHNOLOGY

Work on the AscendC Backend:

- Updated NPU-Info Parsing Script to work on 910B server.
- Updated CMake Build Files, Compile Flags etc.
- Fixed couple of compile errors that appeared after migrating to 8.1RC1 related to typecasts. (From 8.0)

Vector Unit Also Functional On Ascend 910B:



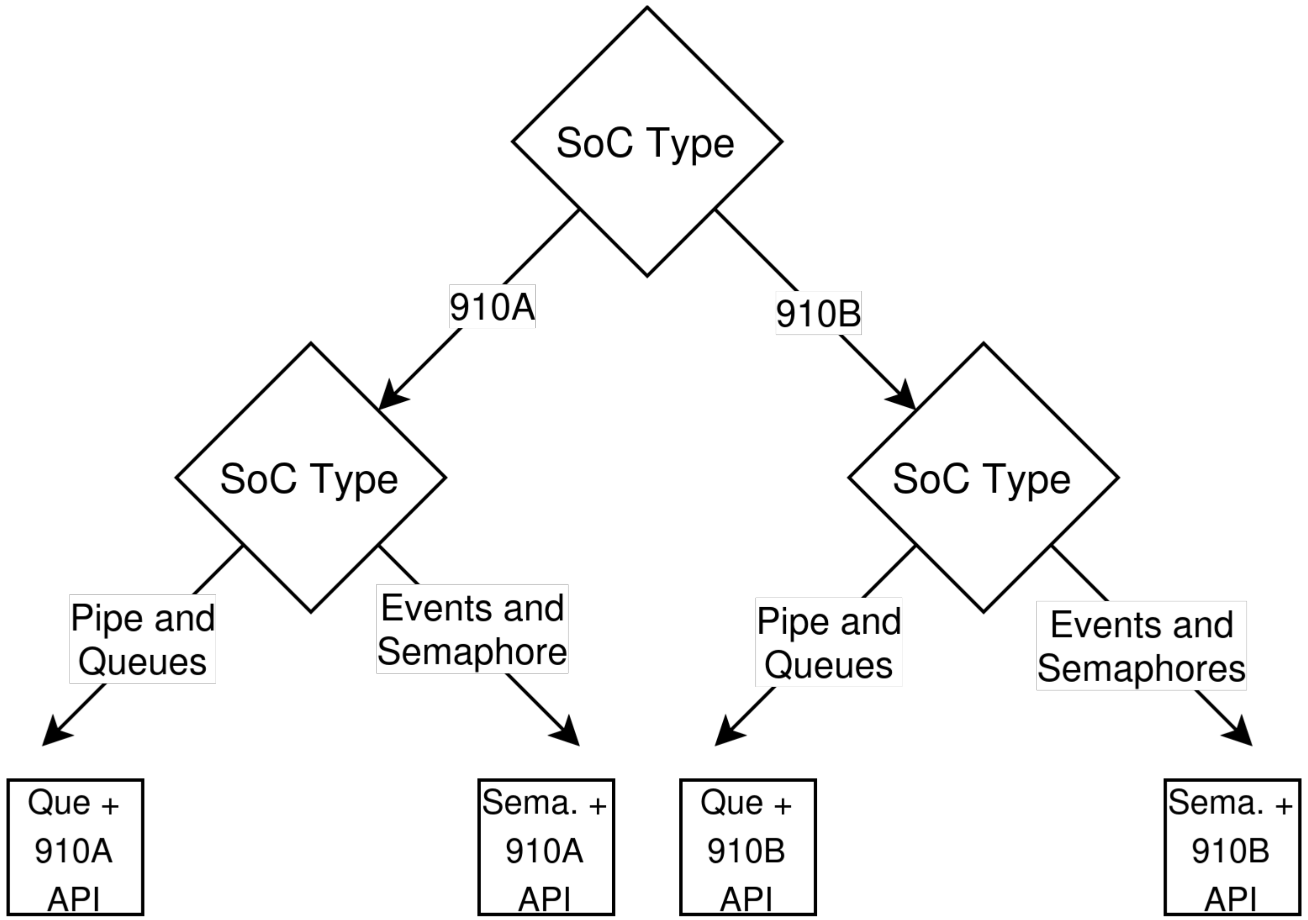
Vector Copy / Addition Benchmark

- Ran N=25+5 benchmarks, Outliers (Z-score > 3) removed, 100 bins, launched 2 kernels per program, measured the second kernel. Discarded the first 5 programs launches.
- The runtime distribution is not normal distribution. The median runtime of the N'=25 runs. The distribution looks binomial and the difference between minimum and maximum runtime is not significant.
- Vector Copy kernel computing **B = A** timeouts using AscendC::Que's, the kernel computes **C = A + B**
- Every AiCore adds **frag_size** elements at a time, vectors added **32** * **frag_siz**e at a time. Pseudocode for the copy benchmark:

```
for (int64_t i = 0; i < vector_size; i += 32 * frag_size)
{
    int ii = (frag_size * AscendC::GetBlockIdx());
    {
        ...
        Add(OUT_frag_C, IN_frag_A, IN_frag_B, frag_size);
        ...
    }
}
```

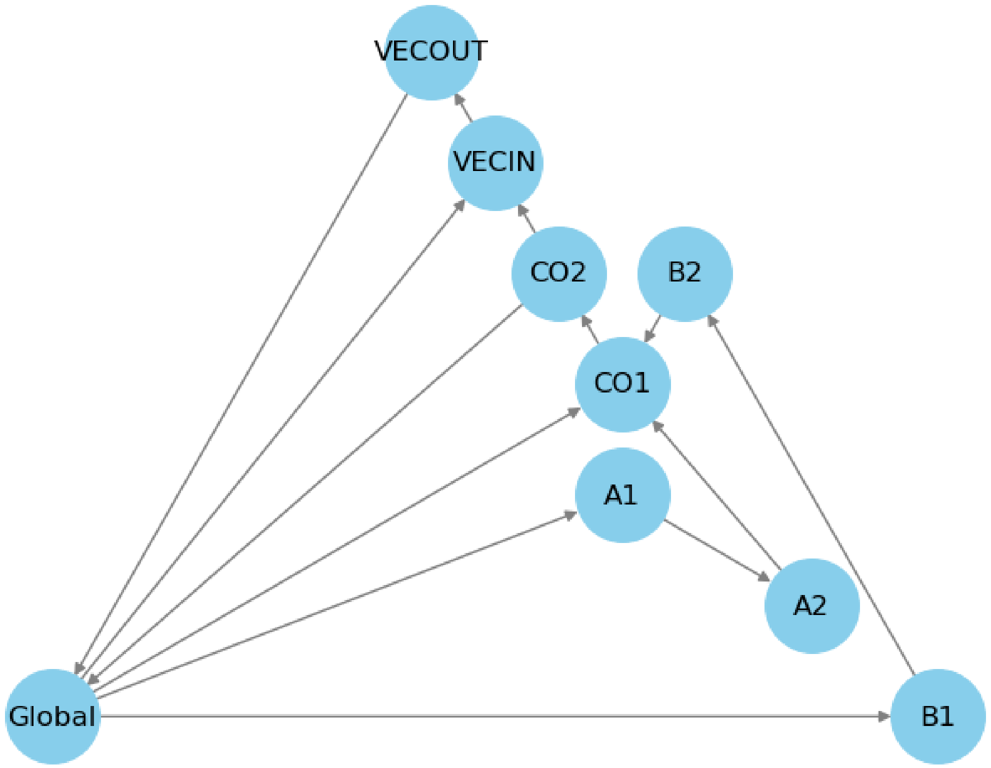
New Strategy To Generate Copies:

- Strategy pattern with a pool of copy functions.
- One the copies are chosen depending on the configuration and the hardware model.

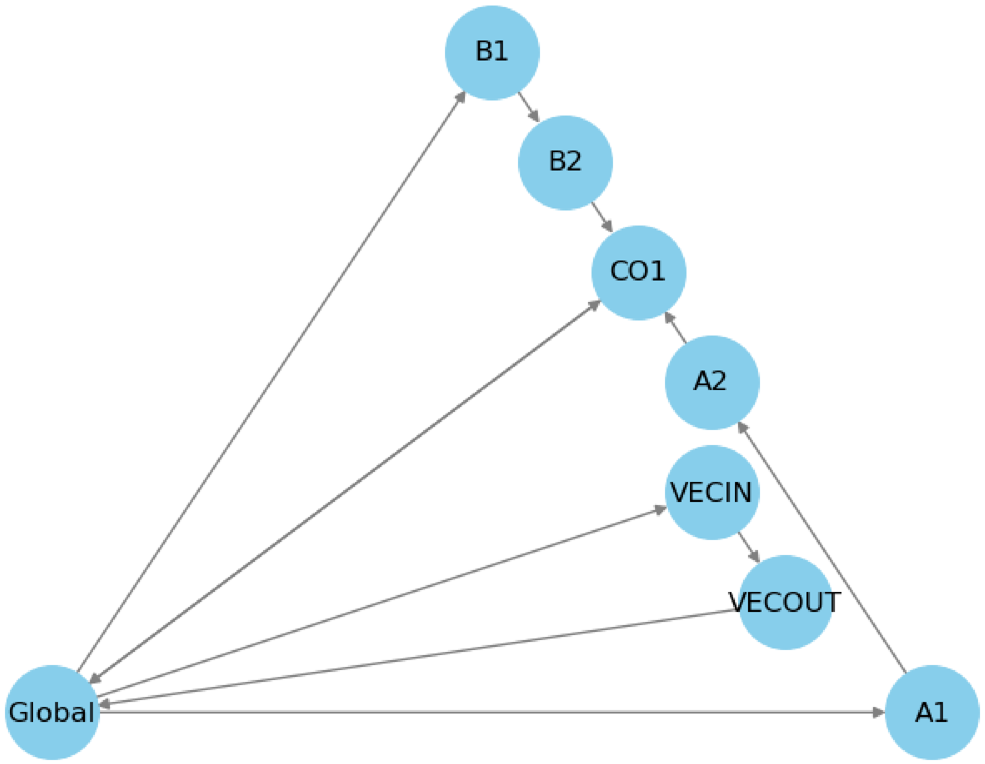


New Strategy To Generate Copies:

- Why a pool of copy functions?



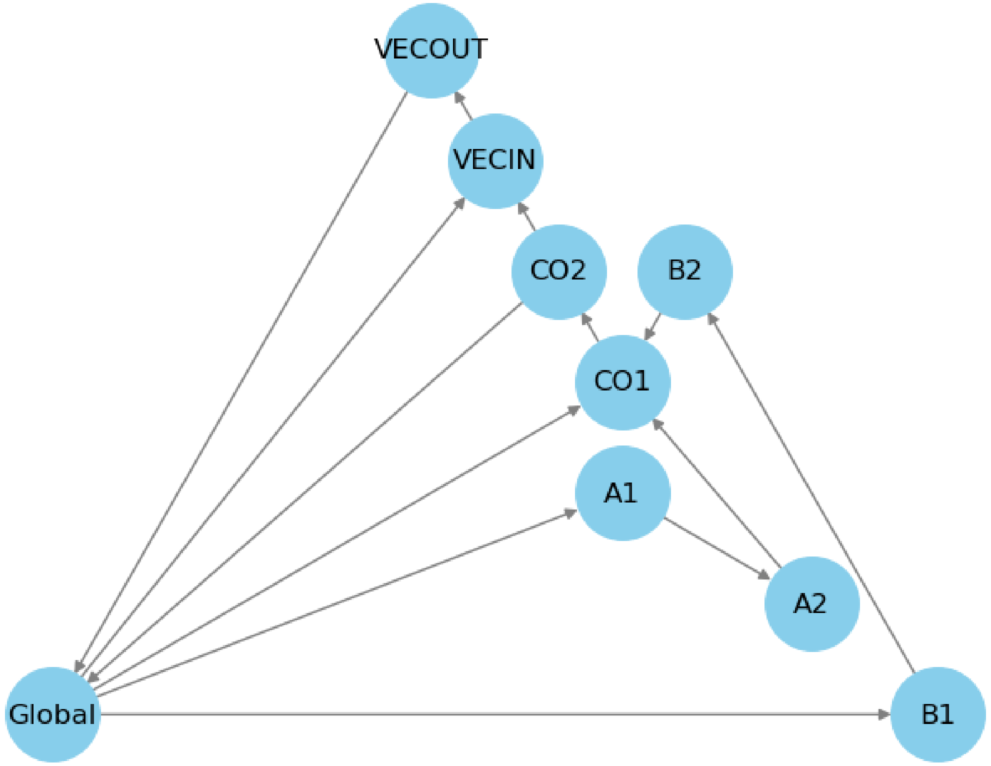
Possible memory copies for 910A



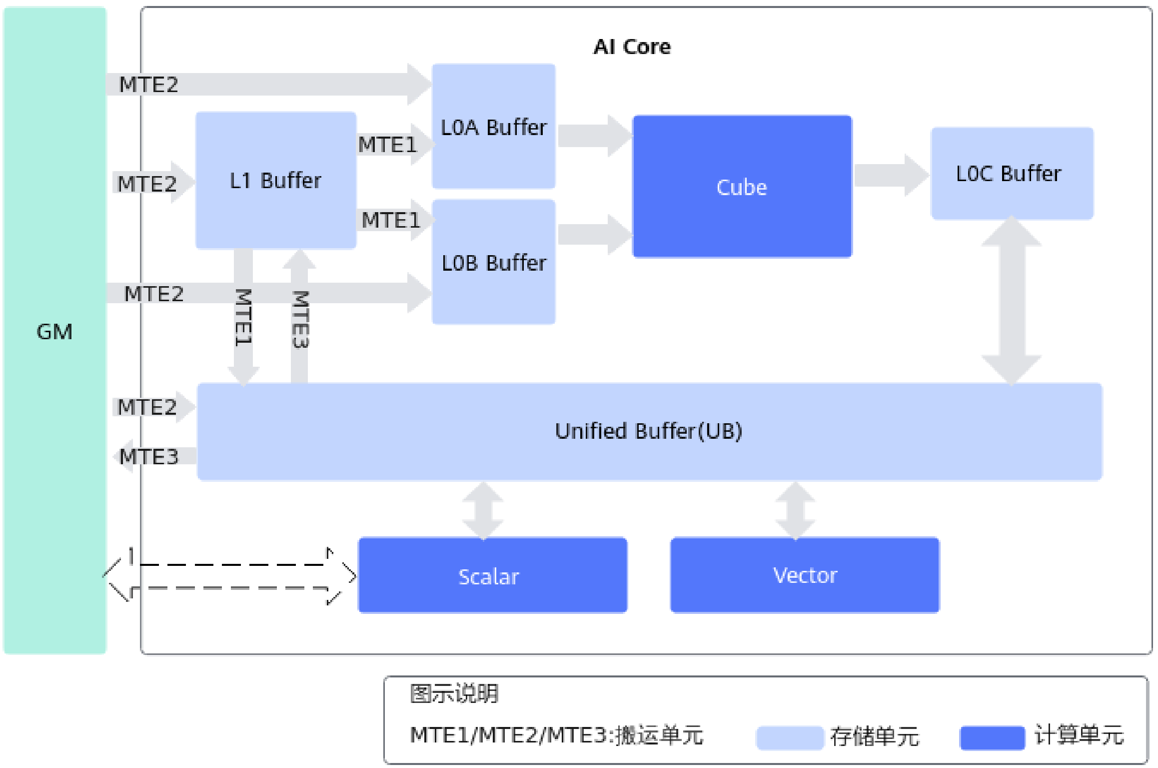
Possible memory copies for 910B

New Strategy To Generate Copies:

- Why a pool of copy functions?



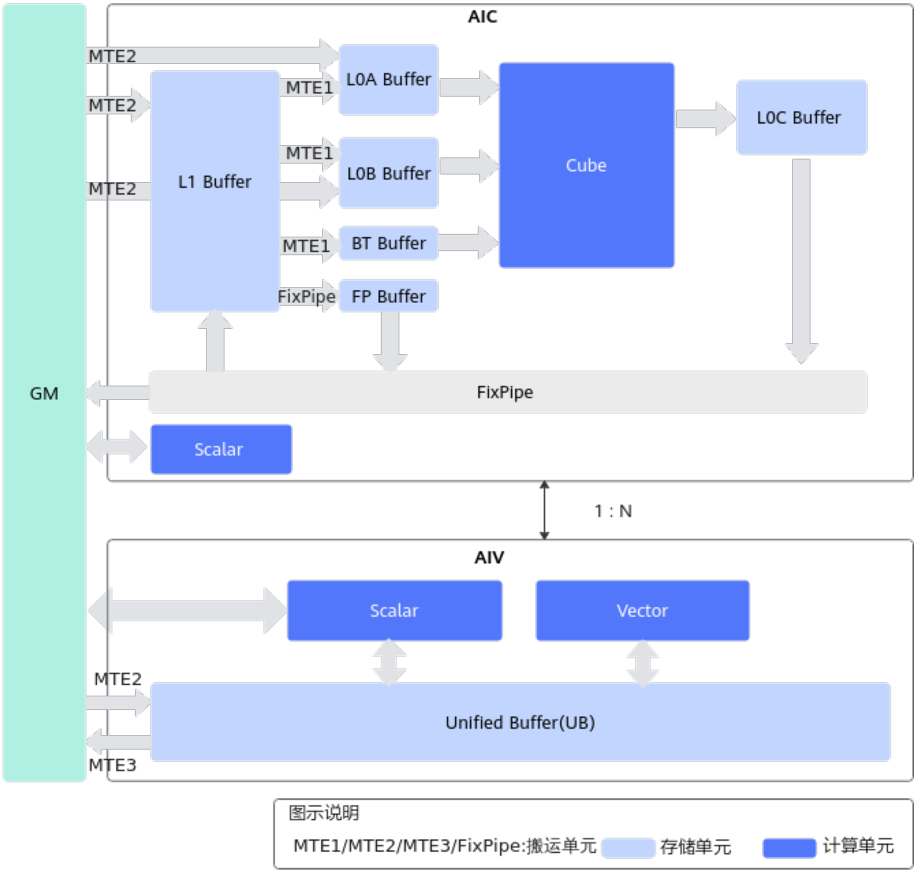
Possible memory copies for 910A



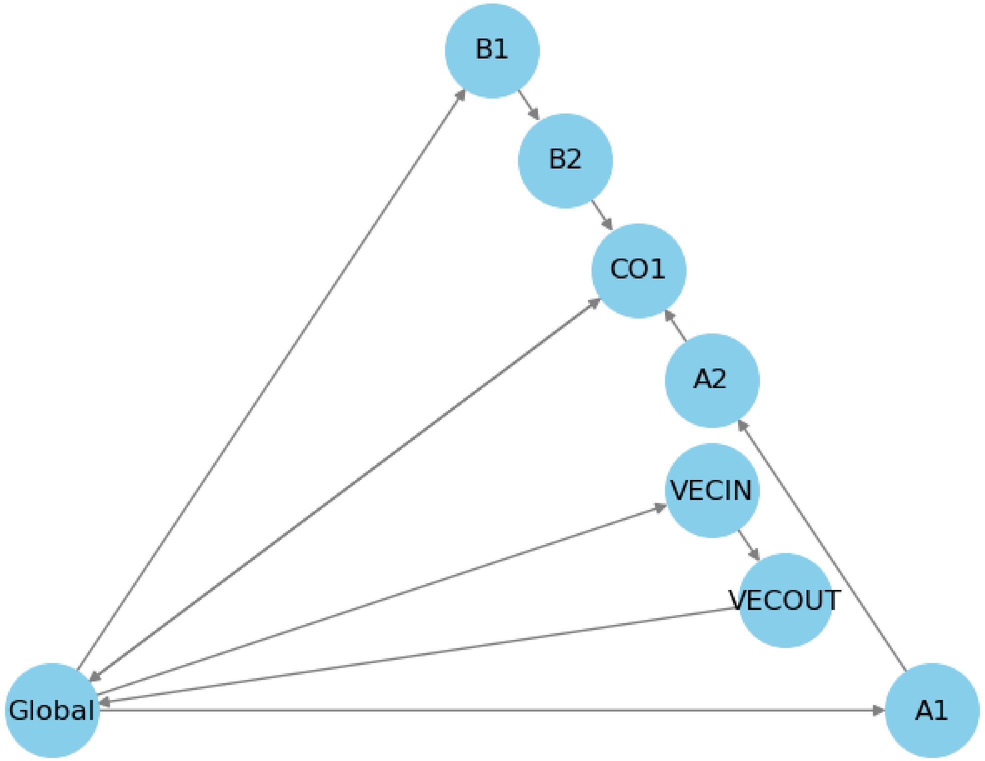
https://www.hiascend.com/document/detail/zh/canncommercial/81RC1/developmentguide/opdevg/Ascendcopdevg/atlas_ascendc_10_0008.html

New Strategy To Generate Copies:

- Why a pool of copy functions?



https://www.hiascend.com/document/detail/zh/canncommercial/81RC1/developmentguide/opdevg/Ascendcopdevg/atlas_ascendc_10_0008.html

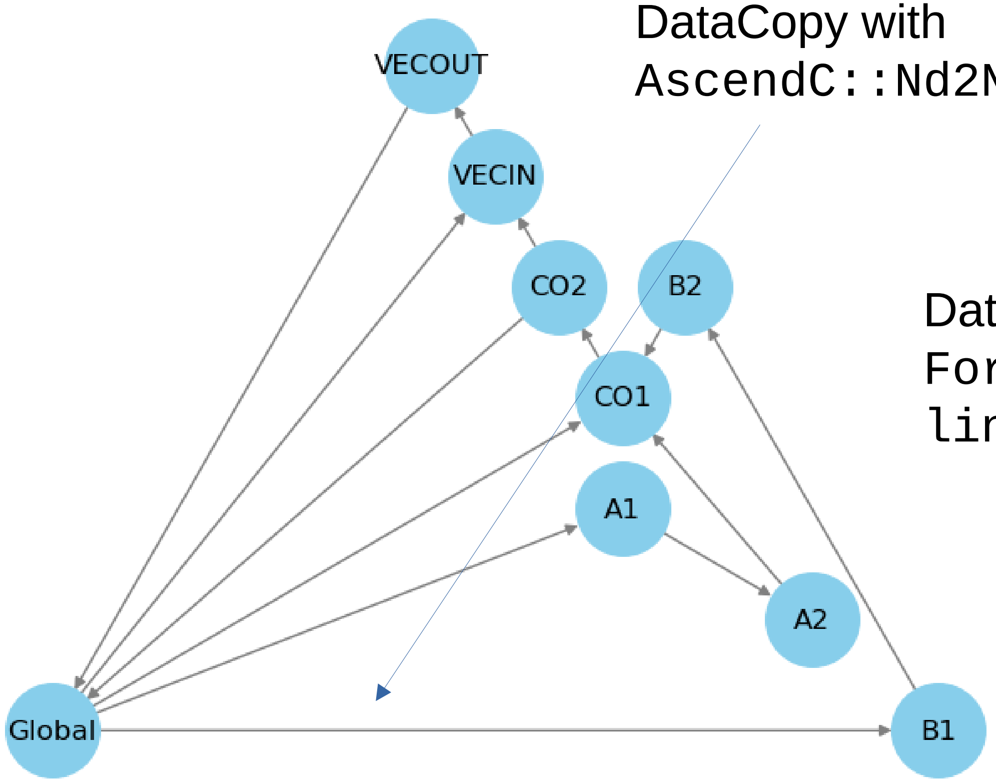


Possible memory copies for 910B

New Strategy To Generate Copies:

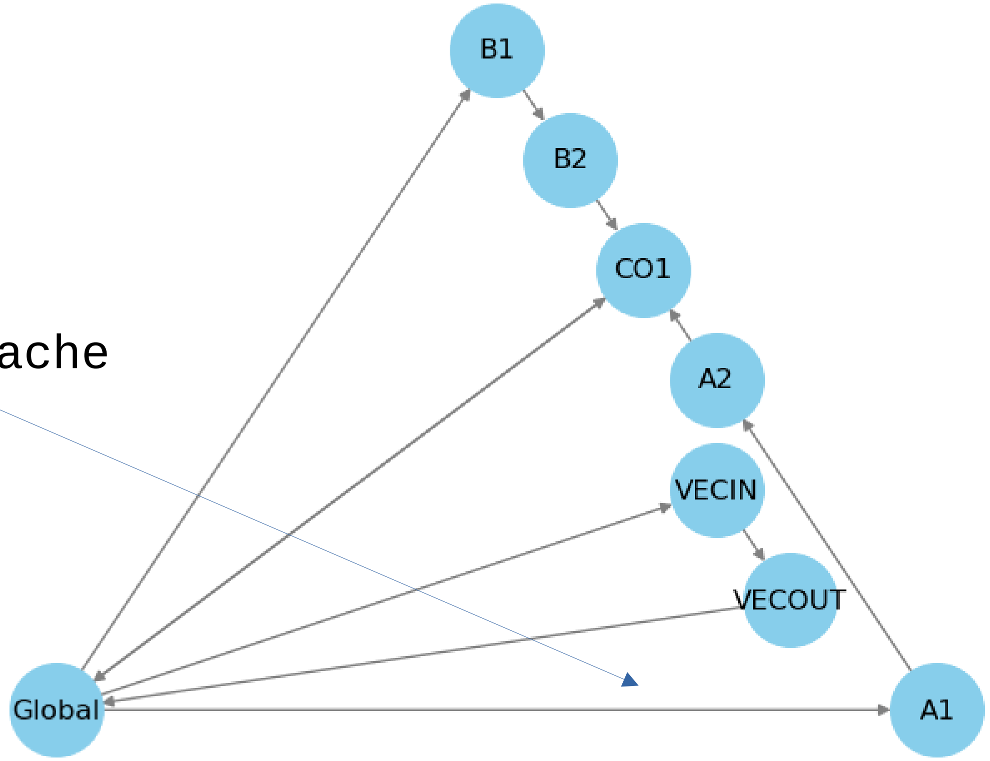
- Why a pool of copy functions?

DataCopy with
`AscendC::Nd2NzParams`



Possible memory copies for 910A

DataCopy with
`For-loop + cache lines width`

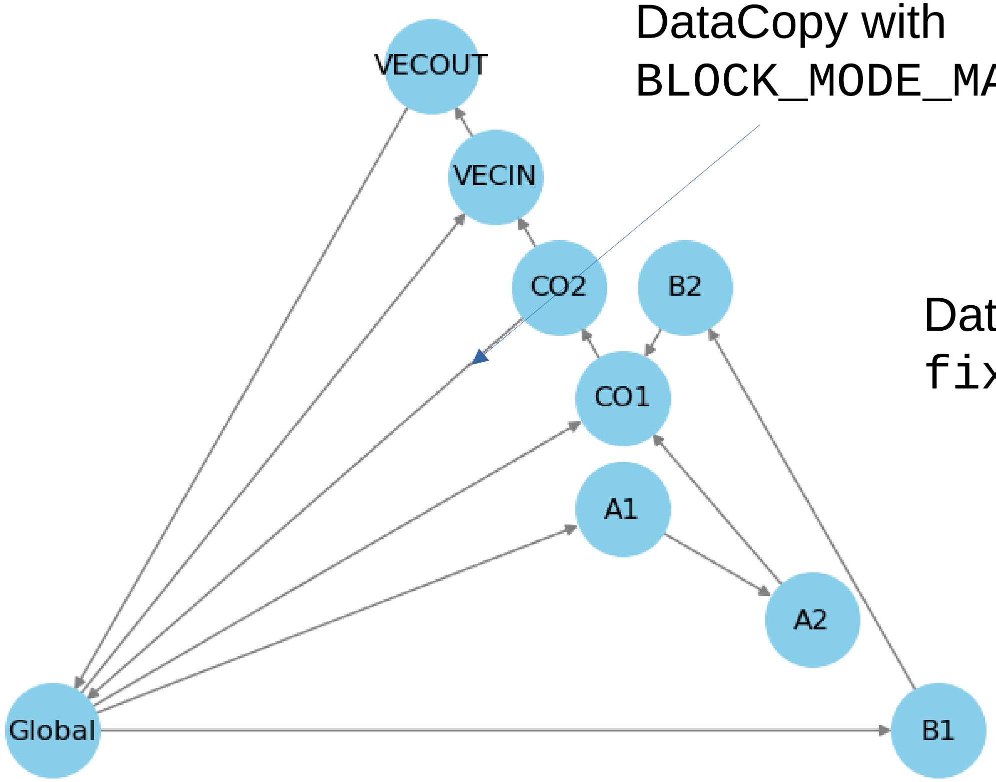


Possible memory copies for 910B

New Strategy To Generate Copies:

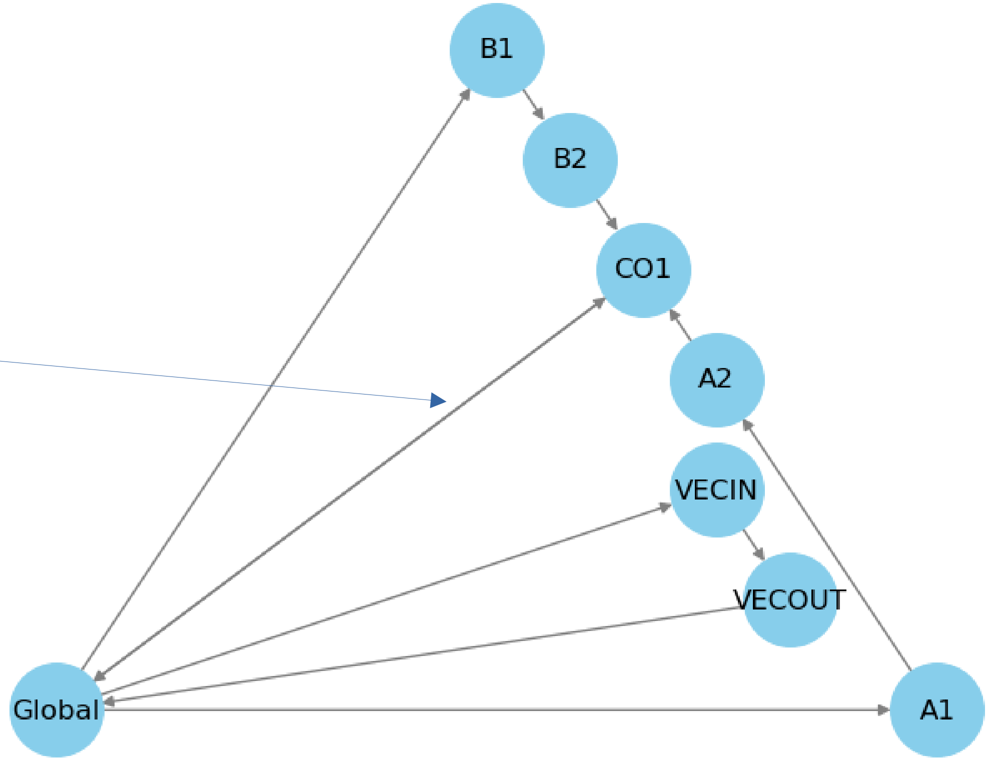
- Why a pool of copy functions?

DataCopy with `BLOCK_MODE_MATRIX`



Possible memory copies for 910A

DataCopy with `fixpipe`



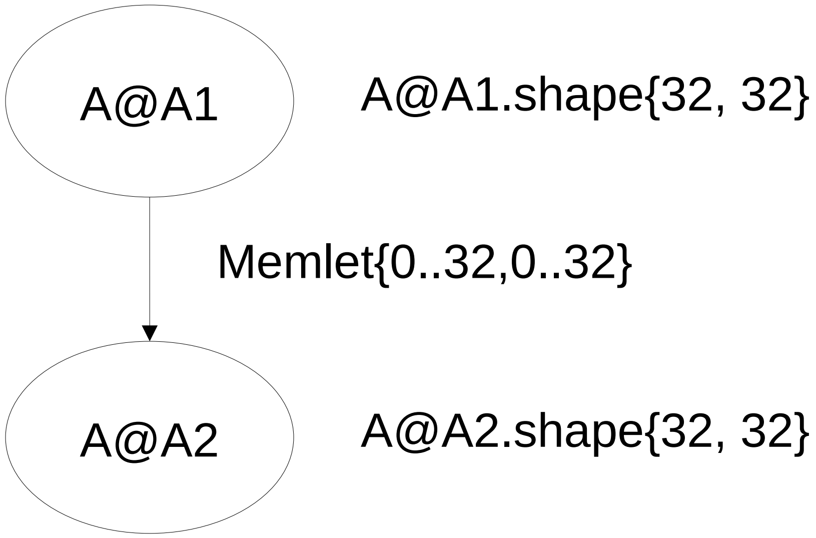
Possible memory copies for 910B

Motivation In the Backend Implementation:

- Expose the dataflow of data
  - While hiding the implementation specific layouts requirements from the user

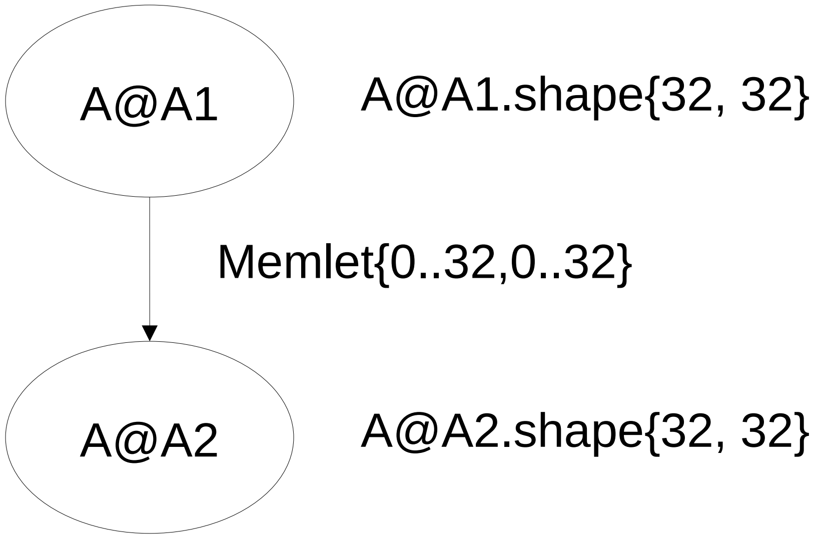
Anatomy of a Copy in SDFG:

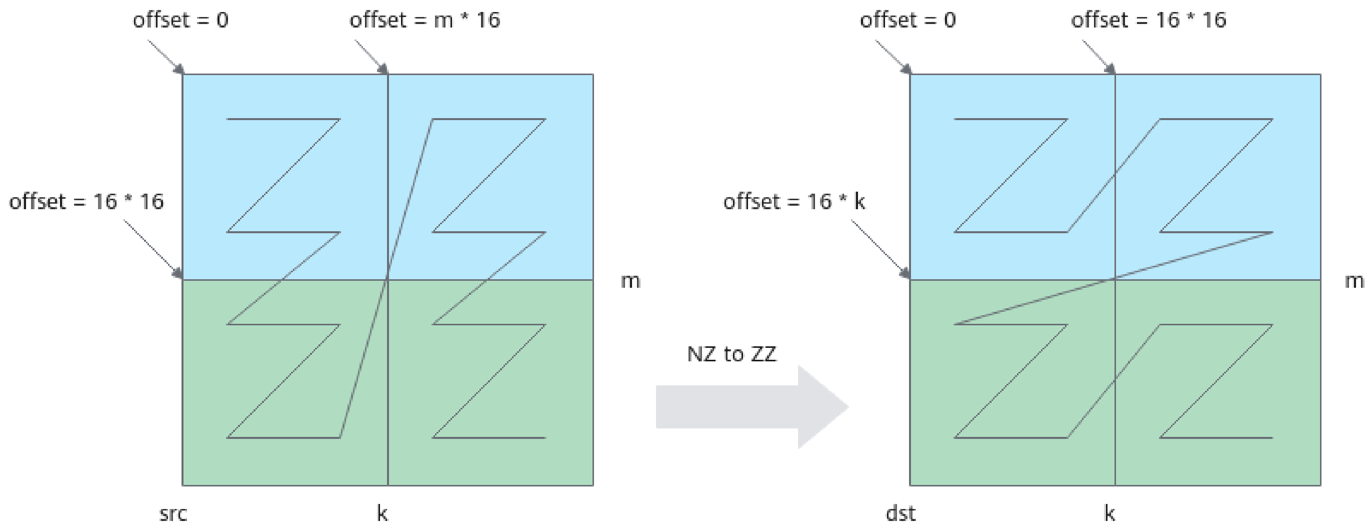
The Goal is the show this to the user:



Anatomy of a Copy in SDFG:

The Goal is the show this to the user:





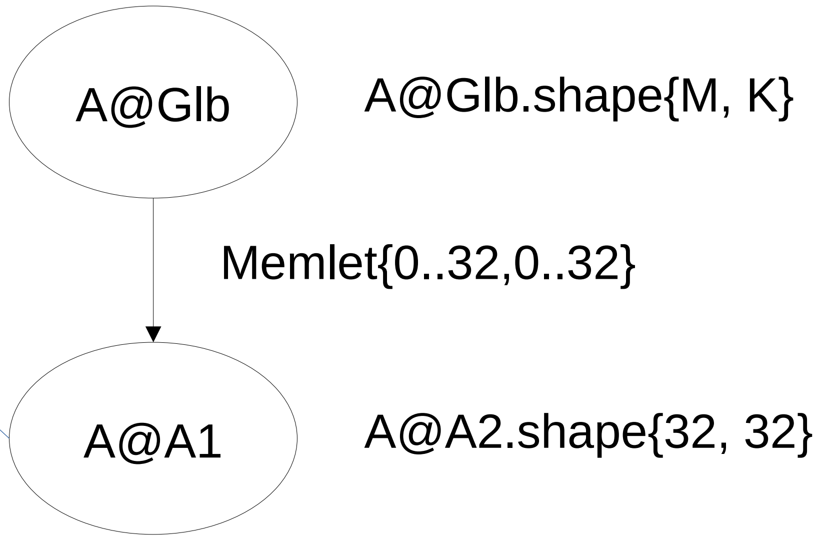
Such that the user does not need to worry about the which storage format is necessary for which A2 vs A1.

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

Anatomy of a Copy in SDFG:

1. Alloc local tensor

```
LocalTensor<>(A_A1)


setGlobalBuffer(A, 32*32)

DataCopyParams Params;
Params.blockCount = 32;
Params.blockLen = 32 / 16;
Params.srcStride =
static_cast<uint16_t>((N / 16) - 1);
Params.dstStride = 0;
```




```
DataCopy(…)
QueueA_A1.Enque(A_A1)
```

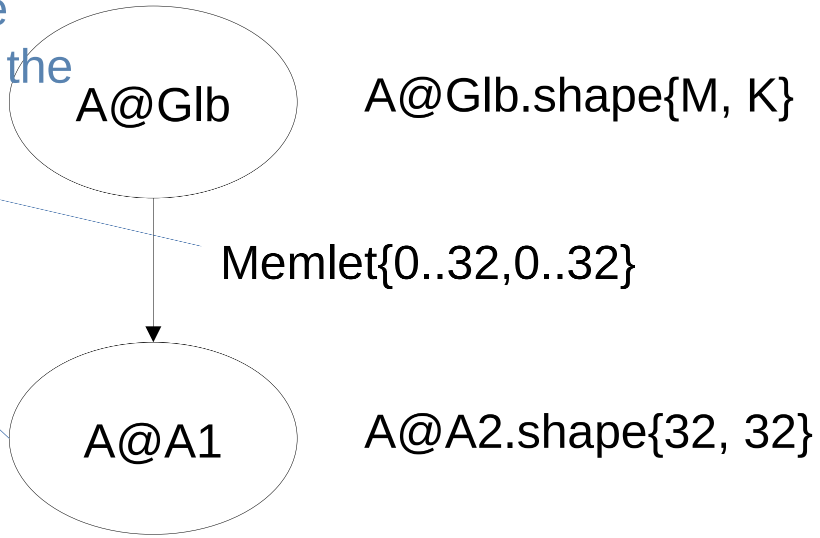
Anatomy of a Copy in SDFG:

1. Alloc local tensor

```
LocalTensor<>(A_A1)
```

2. Read the buffer size read from the memlet

```
setGlobalBuffer(A, 32*32)

DataCopyParams Params;
Params.blockCount = 32;
Params.blockLen = 32 / 16;
Params.srcStride =
static_cast<uint16_t>((N / 16) - 1);
Params.dstStride = 0;
```




```
DataCopy(…)
QueueA_A1.Enque(A_A1)
```

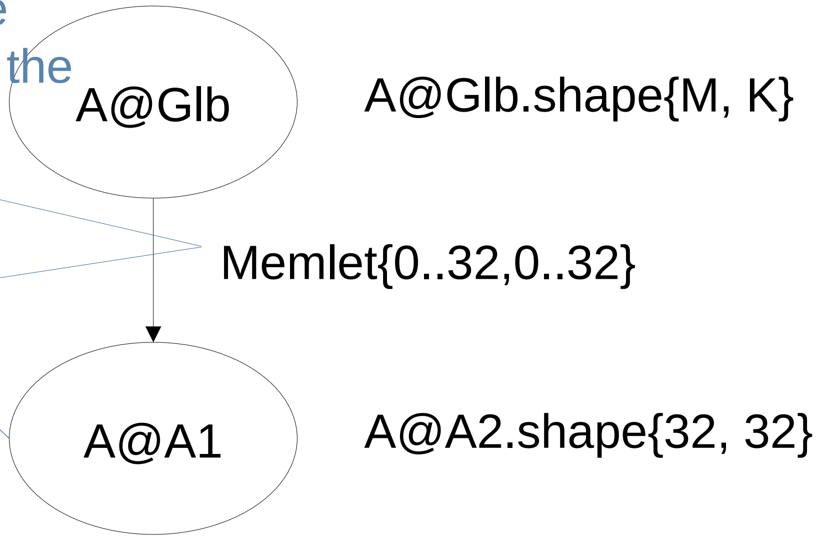
Anatomy of a Copy in SDFG:

1. Alloc local tensor

```
LocalTensor<>(A_A1)
```

2. Read the buffer size read from the memlet

```
setGlobalBuffer(A, 32*32)

DataCopyParams Params;
Params.blockCount = 32;
Params.blockLen = 32 / 16;
Params.srcStride =
static_cast<uint16_t>((N / 16) - 1);
Params.dstStride = 0;
```



3. Read the copy parameters (currently the input matrices need to be row-major)

```
DataCopy(…)
QueueA_A1.Enque(A_A1)
```

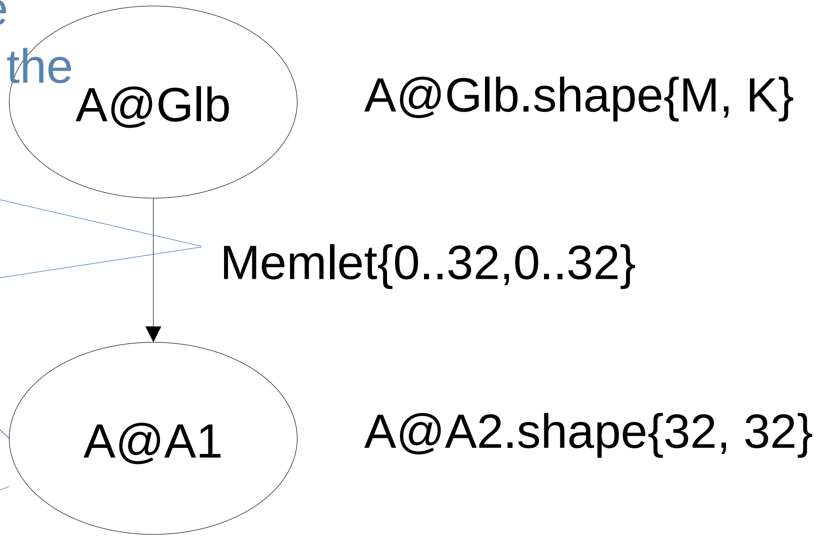
Anatomy of a Copy in SDFG:

1. Alloc local tensor

```
LocalTensor<>(A_A1)
```

2. Read the buffer size read from the memlet

```
setGlobalBuffer(A, 32*32)

DataCopyParams Params;
Params.blockCount = 32;
Params.blockLen = 32 / 16;
Params.srcStride =
static_cast<uint16_t>((N / 16) - 1);
Params.dstStride = 0;
```



3. Read the copy parameters (currently the input matrices need to be row-major)

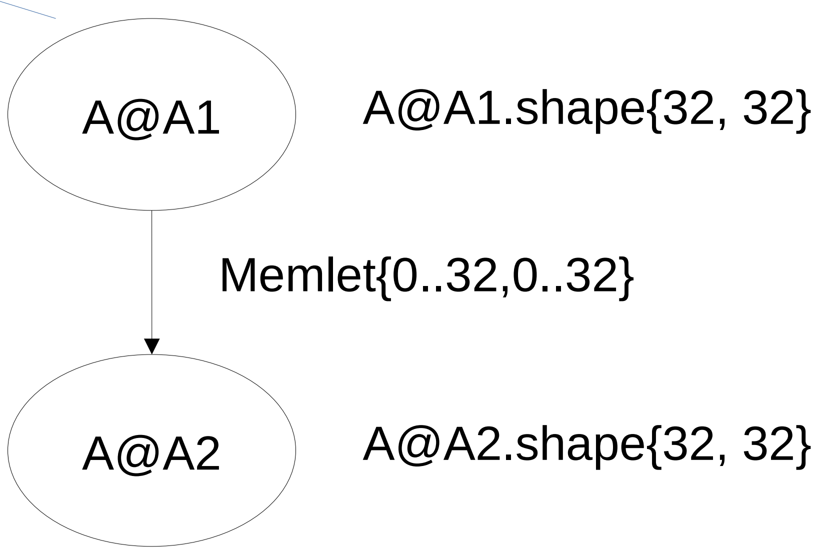
```
DataCopy(…)
QueueA_A1.Enque(A_A1)
```

4. Register copy and enque

Anatomy of a Copy in SDFG:

```
A_A1 = queue_A_A1.DeQue<>()

For (...) {
int srcOffset = …, int dstOffset = …



LoadDataParams;
LoadDataParams.repeatTimes = 32 / 16;
LoadDataParams.srcStride = 32 / 16;
LoadDataParams.ifTranspose = true;

LoadData(A_A2[dstOffset], A_A1[srcOffset],
LoadDataParams);

}

queue_A_A2.EnQue(A_A2);
queue_A_A1.FreeTensor(A_A1);
```

1. Deque src

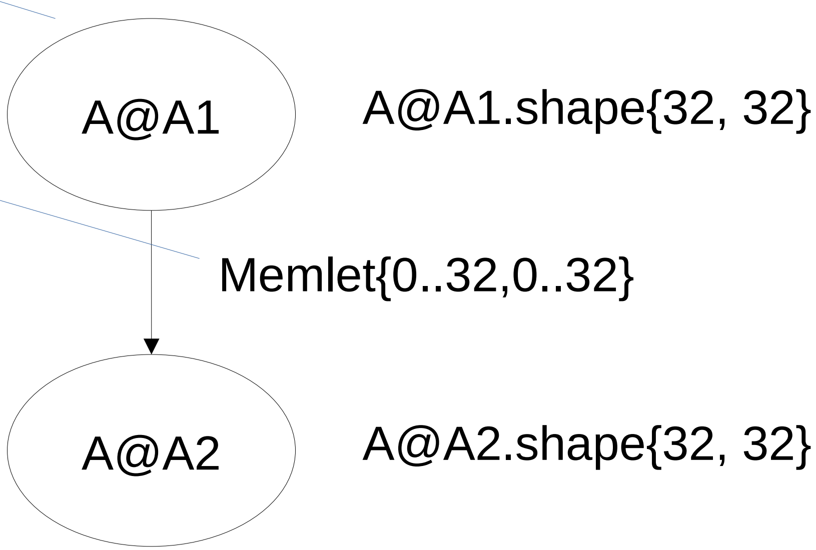


Anatomy of a Copy in SDFG:

```
A_A1 = queue_A_A1.DeQue<>()

For (...) {
int srcOffset = …, int dstOffset = …



LoadDataParams;
LoadDataParams.repeatTimes = 32 / 16;
LoadDataParams.srcStride = 32 / 16;
LoadDataParams.ifTranspose = true;

LoadData(A_A2[dstOffset], A_A1[srcOffset],
LoadDataParams);

}

queue_A_A2.EnQue(A_A2);
queue_A_A1.FreeTensor(A_A1);
```

1. Deque src

2. generate the for loop to copy the memlet shape as a unit of 16x16 blocks



Anatomy of a Copy in SDFG:

```
A_A1 = queue_A_A1.DeQue<>()

For (...) {
int srcOffset = …, int dstOffset = …
```

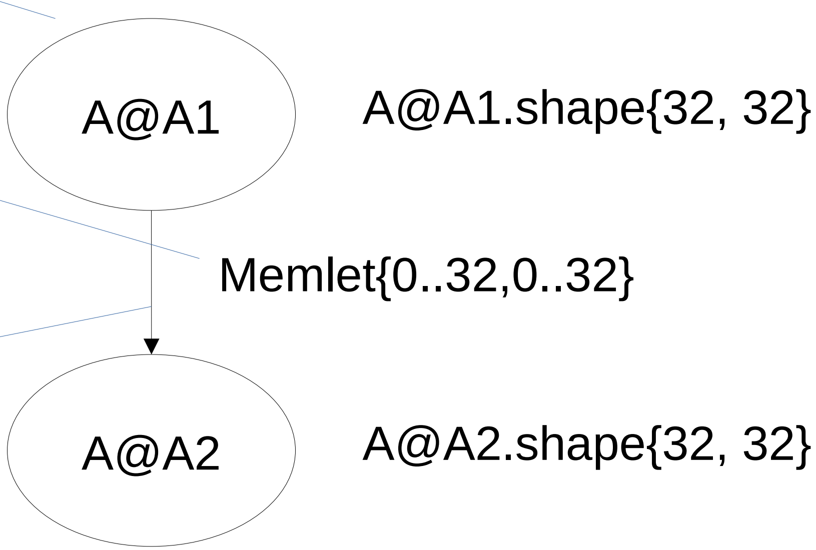
1. Deque src

2. generate the for loop to copy the memlet shape as a unit of 16x16 blocks

```
LoadDataParams;
LoadDataParams.repeatTimes = 32 / 16;
LoadDataParams.srcStride = 32 / 16;
LoadDataParams.ifTranspose = true;
```

3. generate the copy call

```
LoadData(A_A2[dstOffset], A_A1[srcOffset],
LoadDataParams);

}

queue_A_A2.EnQue(A_A2);
queue_A_A1.FreeTensor(A_A1);
```



Anatomy of a Copy in SDFG:

```
A_A1 = queue_A_A1.DeQue<>()
```

1. Deque src

```
For (...) {
int srcOffset = …, int dstOffset = …
```

2. generate the for loop to copy the memlet shape as a unit of 16x16 blocks

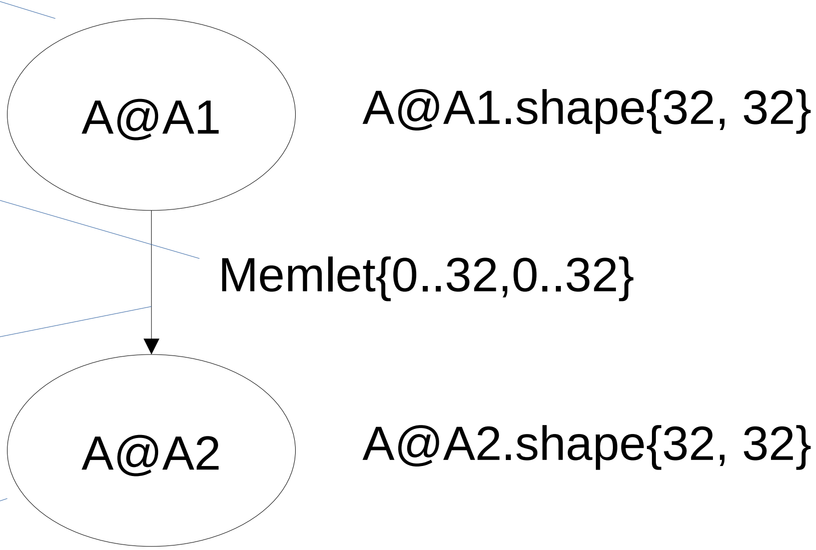
```
LoadDataParams;
LoadDataParams.repeatTimes = 32 / 16;
LoadDataParams.srcStride = 32 / 16;
LoadDataParams.ifTranspose = true;
```

3. generate the copy call

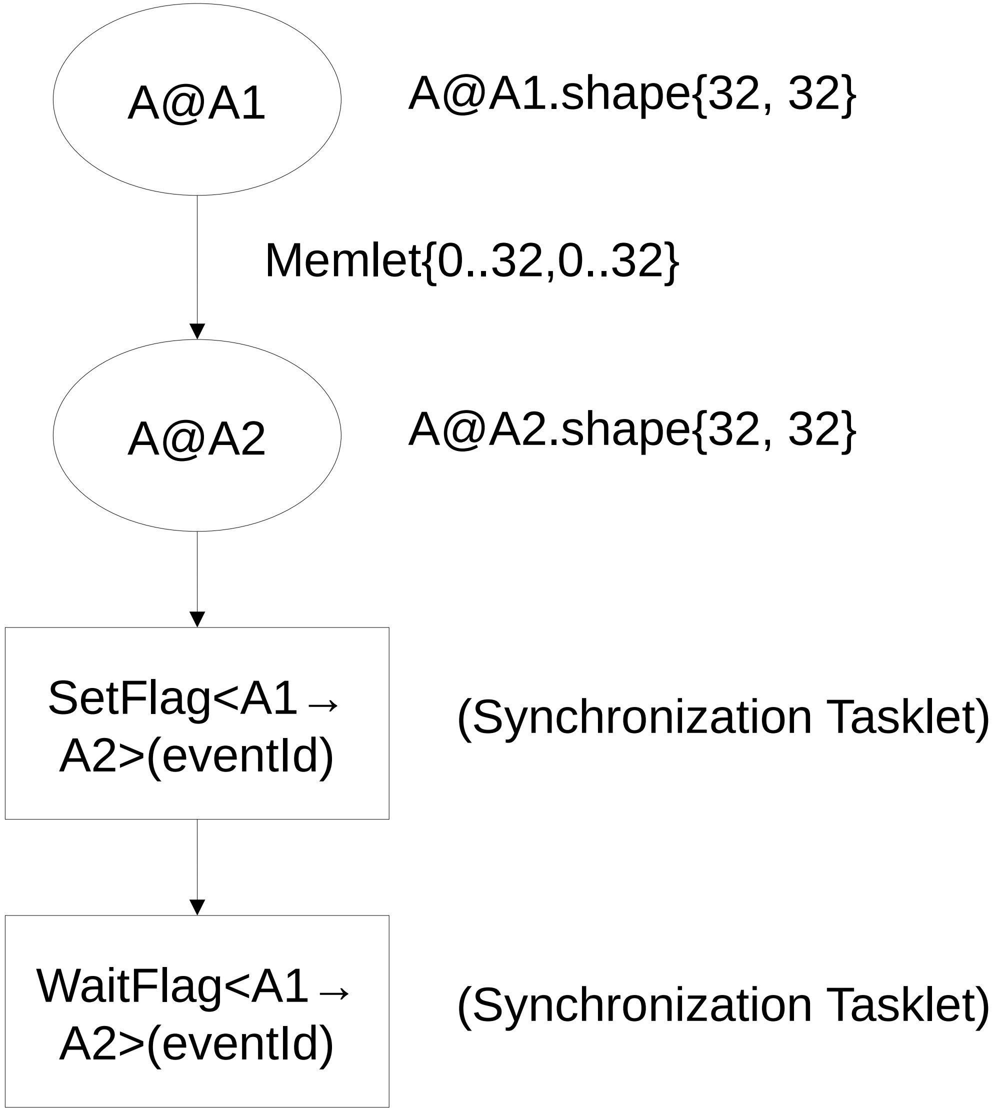
```
LoadData(A_A2[dstOffset], A_A1[srcOffset],
LoadDataParams);

}
```

4. register copy, free src (alloc of local tensor omitted)

```
queue_A_A2.EnQue(A_A2);
queue_A_A1.FreeTensor(A_A1);
```



Async Copy using AscendC Binary Semaphores:

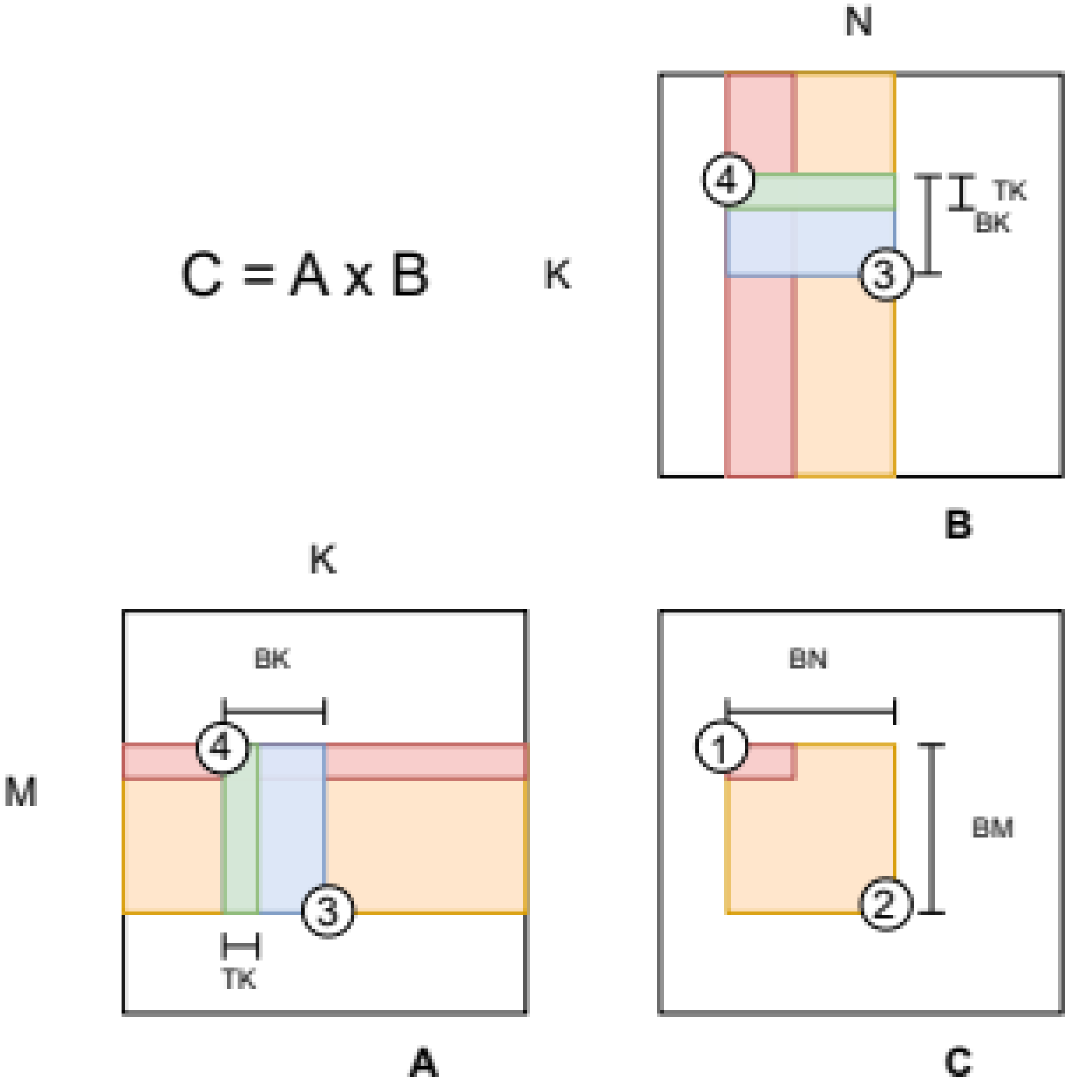


Can be schedule to be just before the first read of A2.

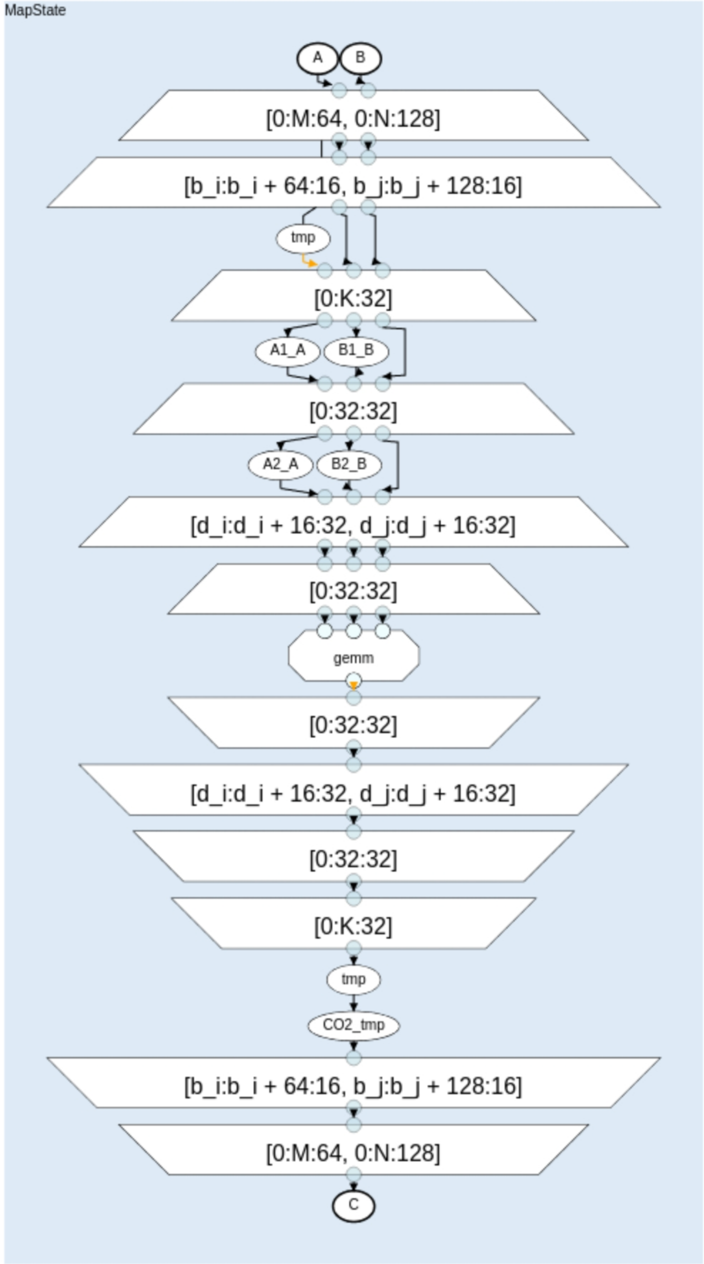
Current Issues:

- Can generate syntactically correct AscendC code for the cube unit.
- Kernel timeouts (identified issues and currently working on fixes)

Current Issues:



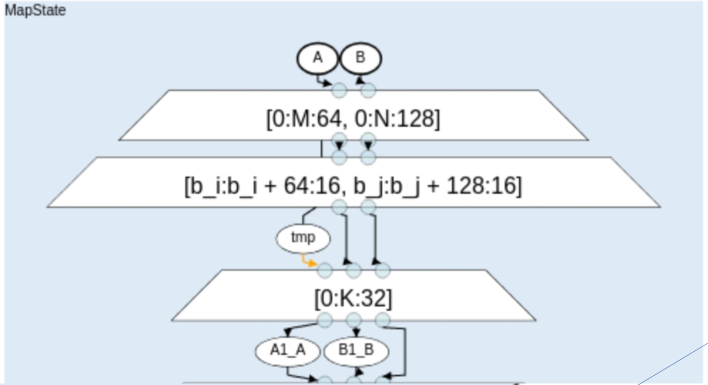
- Transformation (1) defines the per-thread computation domain ($TM \times TN$)
- Transformation (2) establishes the computation of the core-group (e.g., 32 for 910A, 20 for 910B4) domain ($BM \times BN$).
- The first tiling transformation (3) enables explicit data movement from global memory to A1 and B1 memory locations.
- The second tiling transformation (4) orchestrates movement from A1/B1 to A2/B2 respectively.

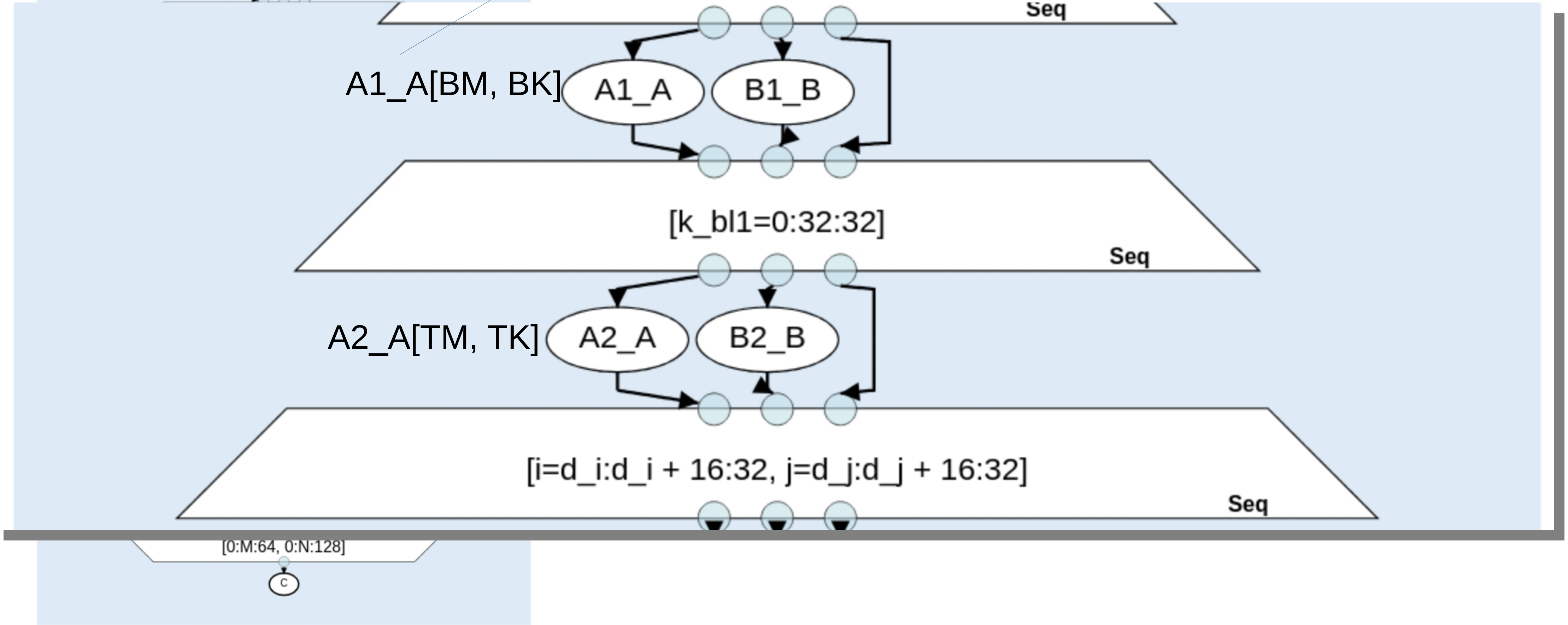


SDFG of 2D-blocktiled baseline GEMM

SDFG of 2D-blocktiled baseline GEMM

Currently A1_A is created by the insert copies transformation as data accessed by all cores (32 cores for 910A)
In this case BM = 4*TM, BK=8*BN, 4*8=32





Current Issues (GitHub Issues Will Be Created Soon):

- For BlockTiling transformation (3: Global → A1, Global → B1) I created a hotfix to create the Glb → A1 → A2 and Glb → B1 → B2 directly within the same map (and only use thread-local tile) where A1 and A2 has the same size. As the current dimensions of A1 and B1 only make sense if cores can access other core’s A1 or B1 storage.
  - If all cores copies the same memory locations from GM to A1 is there a broadcast mechanism?
  - Is it possible for a core to control/see the data another core loads to A1/B1 storage?
  - (The question is: if all levels of the memory locations expect GM are core-local, then will and how data locality effect the performance?)

Current Issues (GitHub Issues Will Be Created Soon):

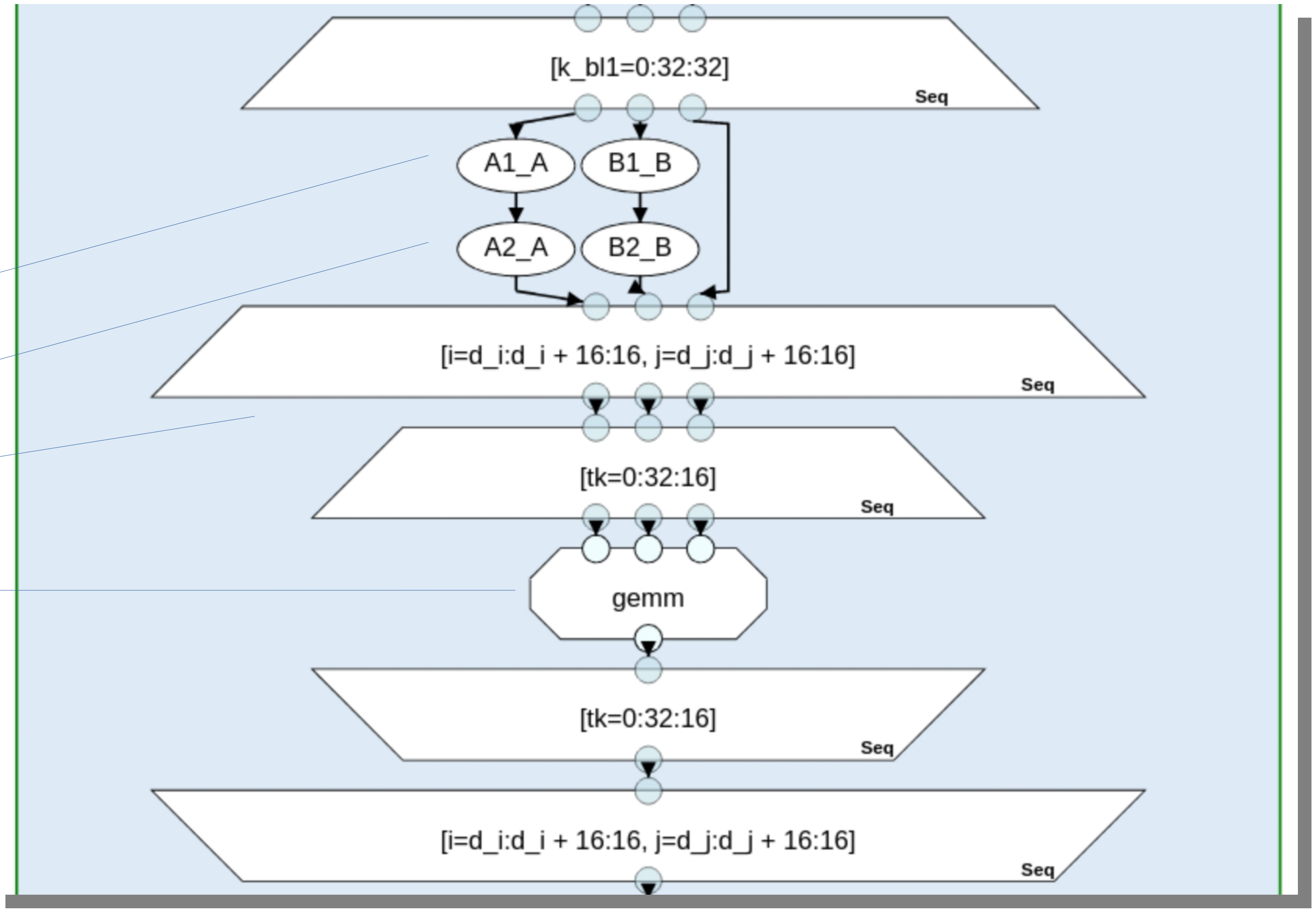
Is moving partial deque from A1 to A2 allowed in the Que & Pipe syntax? The Deque of A1_A and the following implicit synchronization indicates not.

What about if I use binary semaphores?

A1_A[16,32]

A2_A[16,32]

3 Calls the AscendC::Mmad
With M=N=K=16



Current Issues (GitHub Issues Will Be Created Soon):

- Then, do the steps A1 → A2 and B1 → B2 exist purely to make the layout conform with the layout required by the cube unit? (Such that the whole A2 data needs to be consumed within a call to the CubeUnit)
- Is it possible to load load the A1 to A2 in tiles within a for loop with data movement using Ques*

Current Issues (GitHub Issues Will Be Created Soon):

- From the DeQue and Enque operations I consider

```
pipe.InitBuffer(queue_A1_A, 1, 32 * 32 * sizeof(dace::float16));
pipe.InitBuffer(queue_A2_A, 1, 16 * 16 * sizeof(dace::float16));
...
for (int i = 0; i < 4; ++i) { // Num Blocks
    ...
    A1_A = queue_A1_A.DeQue<dace::float16>();
    A2_A = queue_A2_A.AllocTensor<dace::float16>();
    int xOffset = (i%2)*16;
    int yOffset = (i/2)*16;

    AscendC::LoadData2DParams  A2_ALoadDataParams;
    A2_ALoadDataParams.repeatTimes   =  16 / 16; // 16 byze is the unit size
    A2_ALoadDataParams.srcStride   =  16 / 16;
    A2_ALoadDataParams.ifTranspose   =  false; // No transpose for A

    AscendC::LoadData(A2_A, A1_A[xOffset + yOffset * 16], A2_ALoadDataParams);
    if (i == 3) {
        queue_A2_A.EnQue(A1_A);
    } else {
        queue_A2_A.FreeTensor(A1_A);
    }
    queue_A2_A.EnQue(A2_A);
    ...
}
```

This code would timeout / fail due to out of bounds on A2_A?

Next Steps:

- Complete GitHub Issues
- Fix the issues in the generated code
- If time permits:
- Work on the vectorized-stencil microbenchmark for the Vector Unit
- Introduce copies using binary semaphores for the vector unit