Yakup Koray Budanaz

SoftHier July 14

Overiew of the Topics:

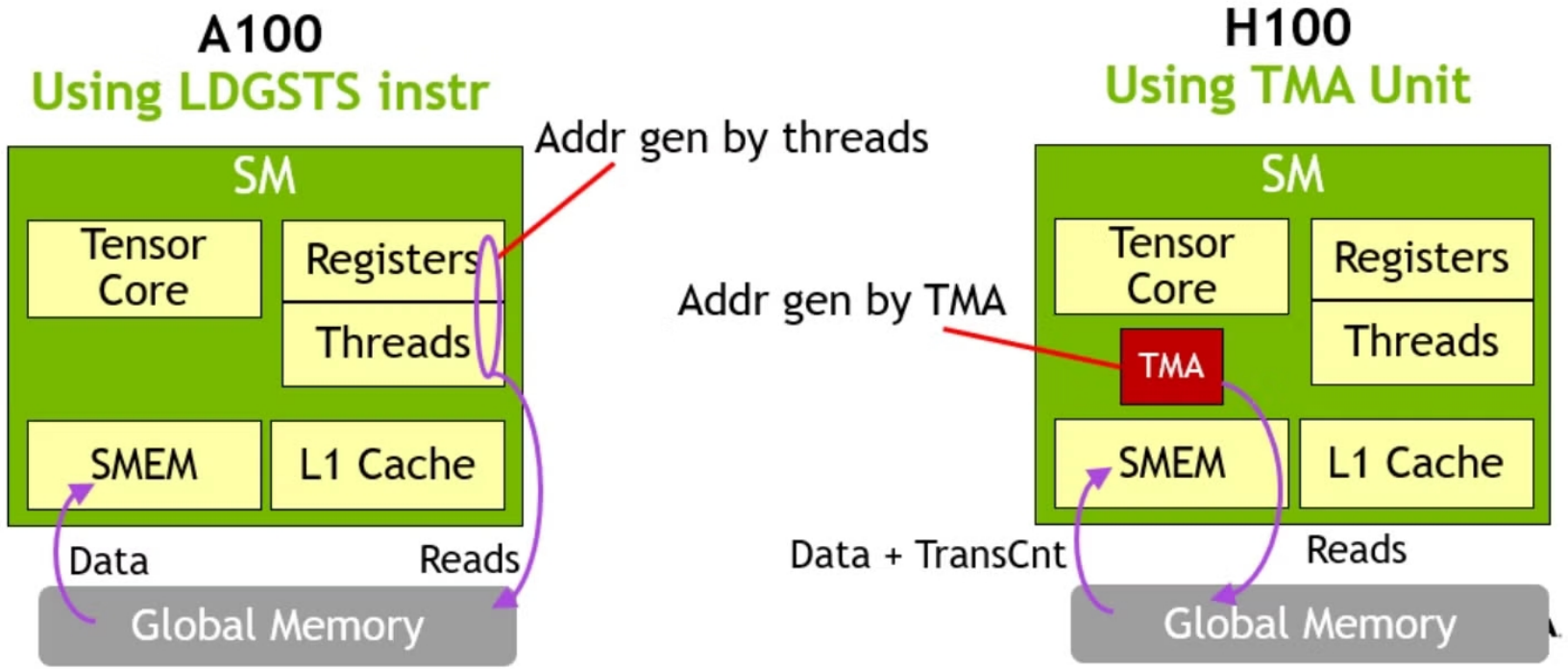
- **Possible Targets For The Next Phase:** *Hopper/Blackwell GPUs and AMD NPUs*
- **New DaCe + SoftHier Project:** *A Full-Fledged SoftHier Backend*
- **A Necessary Side-Quest:** *Re-design of Accelerator Codegen and Offloading Transformations in DaCe*

- Still no response from the legal team.

- **Possible Targets For The Next Phase:** *Hopper/Blackwell GPUs*

Programming Hopper Without Using the L1 Cache

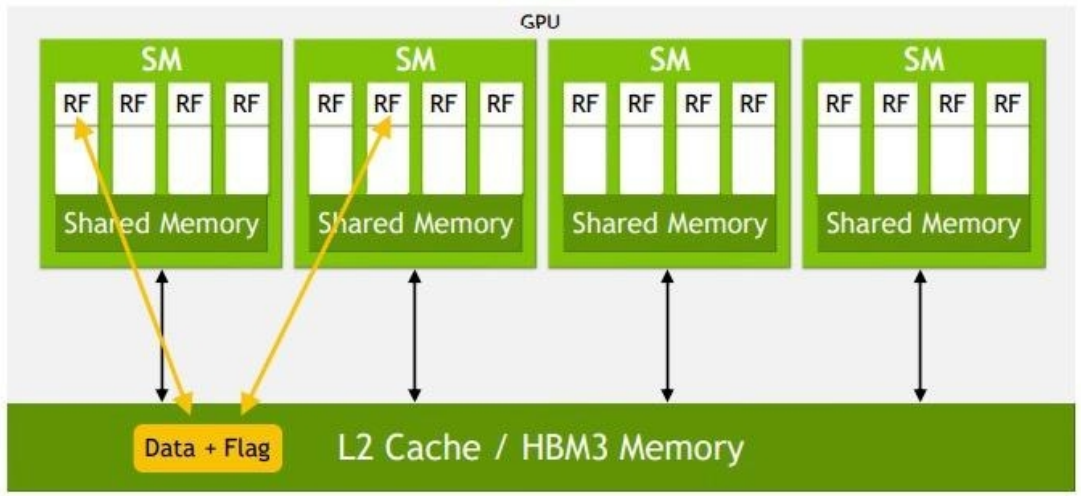
- TMA Engines enable asynchronous memory movement from GPU Global to GPU Shared:

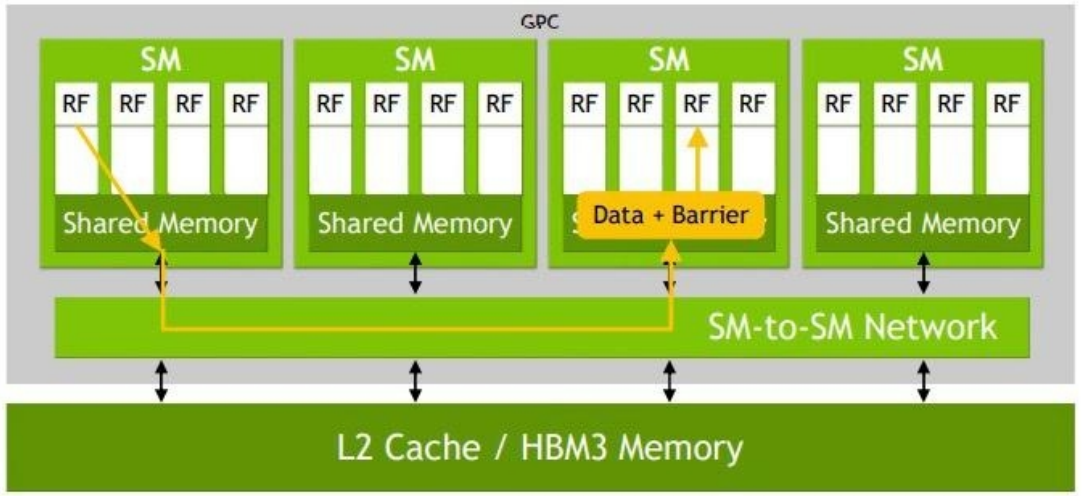


https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Programming Hopper Without Using the L1 Cache

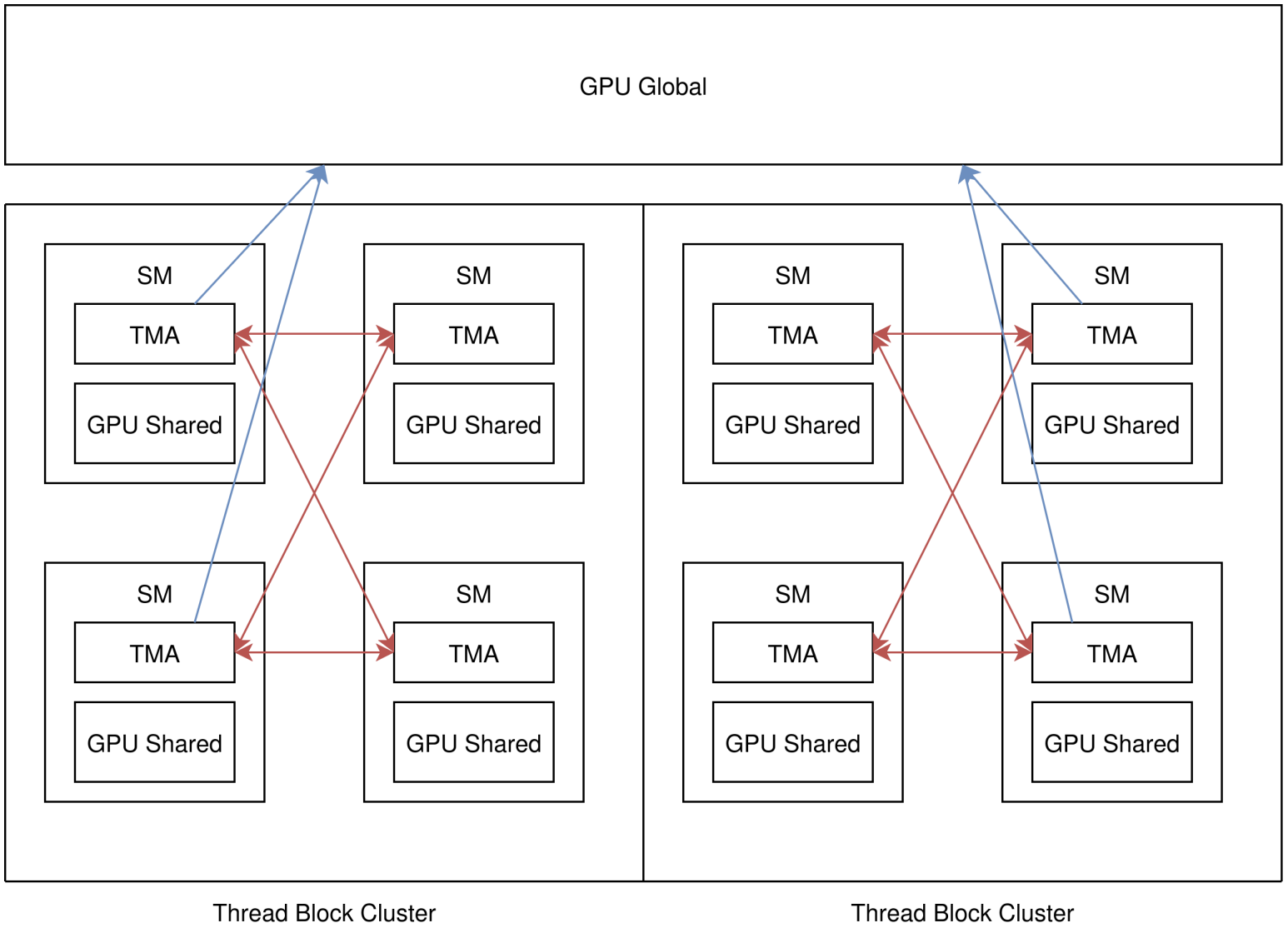
- TMA Engines also enable asynchronous SM-to-SM memory copies.





https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Programming Hopper Without Using the L1 Cache

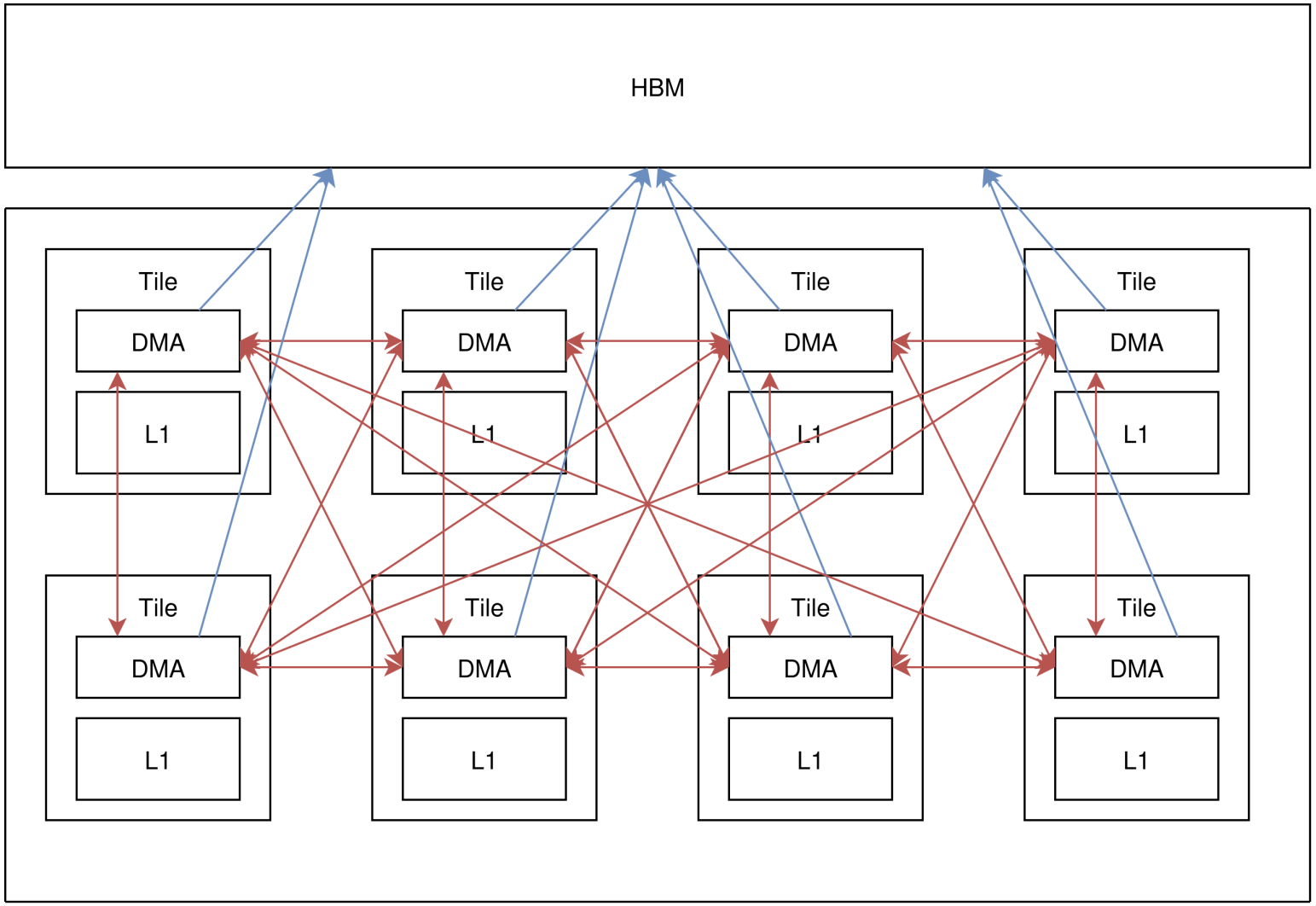


Async pipeline to move memory between GPU L2 cache and GPU Shared memory

Async pipeline to move between GPU shared memory of SMs

Programming SoftHier – And Its Similarities To Hopper GPUs

Even though the on-chip interconnect creates a 2D mesh of PEs, the programming abstractions treats the PEs building a complete graph.



Similarly every PE can access the HBM

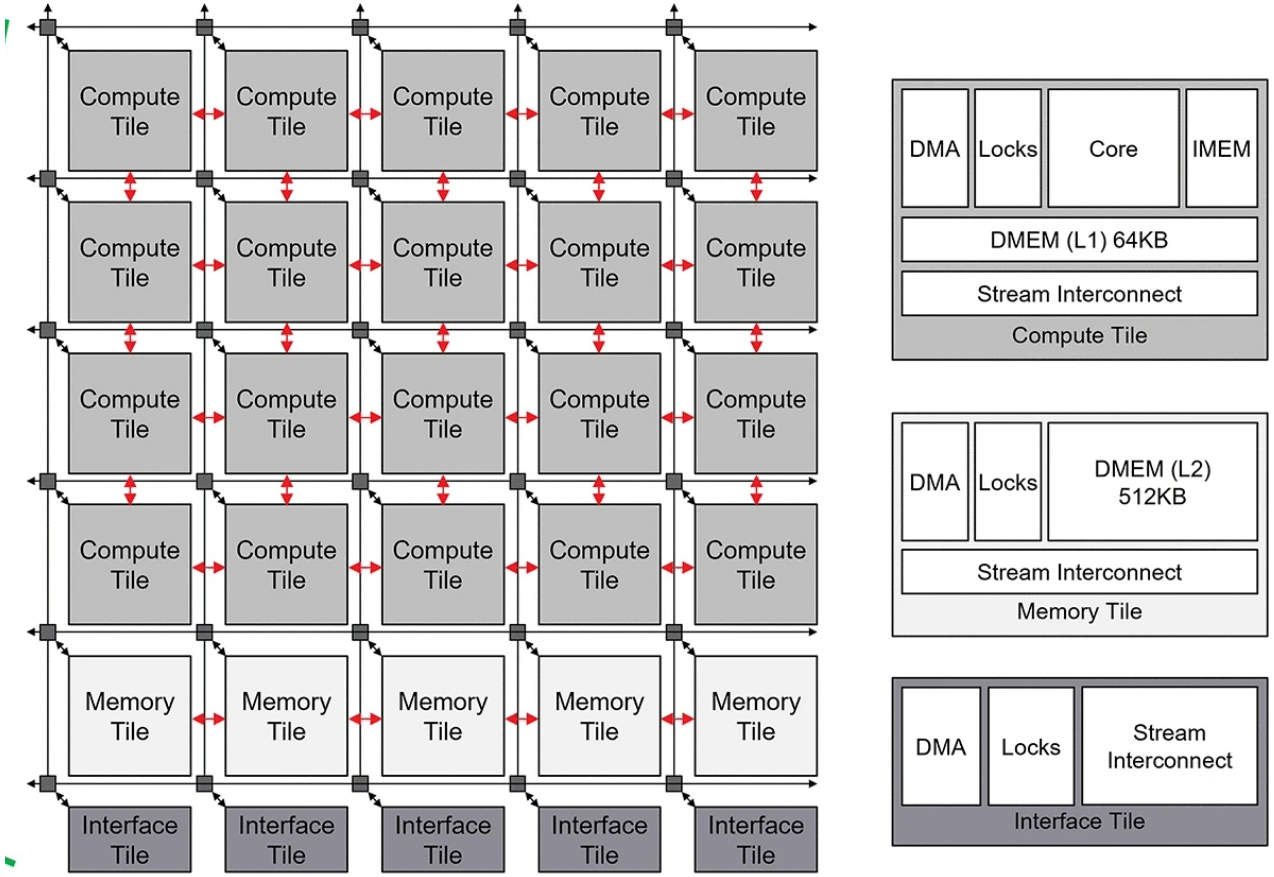
Programming Hopper Without Caches

- If all data access is performed by TMA engine and if no direct access to global memory is performed, then a Hopper GPU can be programmed without using L1 (necessary to use L2 to move from HBM to device).
- TMA engine is a DMA engine and SM-to-SM interconnect provides some the capability as the on-chip interconnect of SoftHier architecture, albeit not available to the complete accelerator.
- *Advantages:*
  - Nvidia has a mature software ecosystem.
  - Having Hopper support in DaCe will enable comparisons with SoftHier simulator for future publications
- *Disadvantages*:
  - There is already many existing work on Nvidia hardware

- **Possible Targets For The Next Phase:** *AMD NPUs*

Architecture Overview of AMD NPUs

- Is a 2D mesh of vector processors with dedicated DMA engines and multiple memory tiles/channels



https://www.computer.org/csdl/magazine/mi/2024/06/10592049/1YtaXNWFBqE

Programming Hopper Without Caches

- Can be programmed using: AIE Core C++ API, AIE MLIR Dialect, Riallto (High-level compared to other two)
- For the kernel switch-boxes need to be set to describe the routing over the on-chip interconnect
- Memory tiles serve as dedicated memory controllers.

Riallto: https://riallto.ai/
AIE C++: https://docs.amd.com/r/en-US/ug1079-ai-engine-kernel-coding/AI-Engine-API-Overview
AIE MLIR: https://github.com/Xilinx/mlir-aie

Programming Hopper Without Caches

- *Disadvantages:*
  - Requires low-level programming of the on-chip interconnect
  - Does not include dedicated matrix multiplication engines
  - Less mature software stack
- *Advantages:*
  - Can provide insight on providing software abstractions to programming on-chip interconnects

- **New DaCe + SoftHier Project:** *A Full-Fledged SoftHier Backend*

A Full-Fledged SoftHier Backend

- With Chi and Aofeng we did the a proof-of-concept for the SoftHier Backend on DaCe
  - We managed to gain insight on the hardware configurations needed under the Chi leading the project.
  - Hopefully we will extend the number of simulations and push for a paper. (Next HPCA or ISCA)
  - Student project proposal ready [1]

[1] Commentor link:
https://docs.google.com/document/d/1CfbUqIIaa4hYTY18cg64c-as2V6oiST5xcUUDqeP-gs/edit?usp=sharing

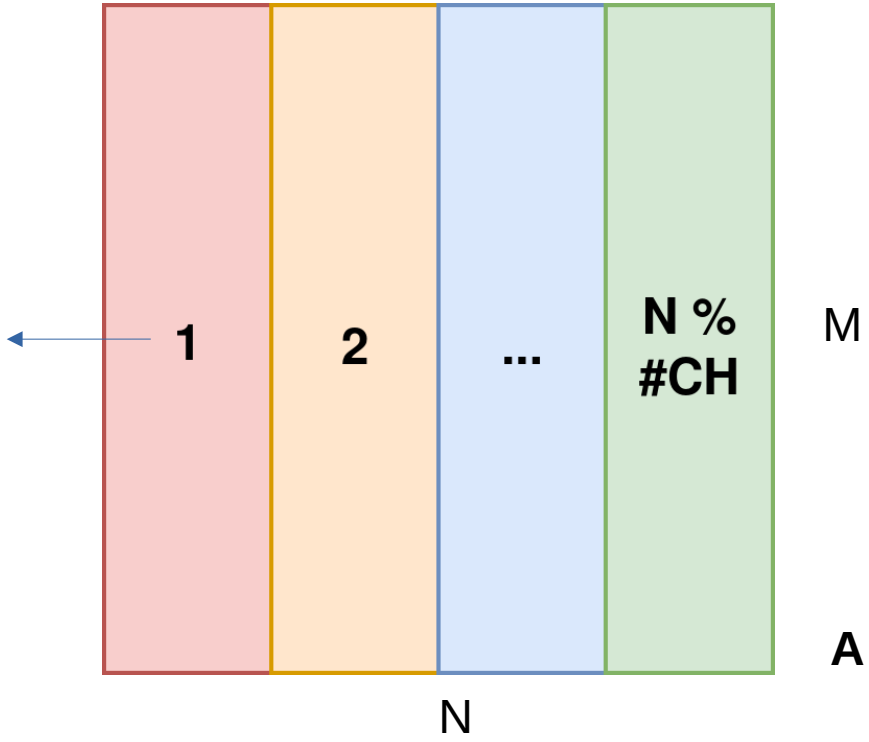
A Full-Fledged SoftHier Backend

- In its current state the back-end can't be integrate to upstream DaCe.
  - I plan to lift the current proof-of-concept implementation to a production-quality backend in a follow-up student project.
  - The focus will be the exploration of the design space and design of the framework instead of the hardware insights obtained.
  - I have already started interviewing students

A Full-Fledged SoftHier Backend

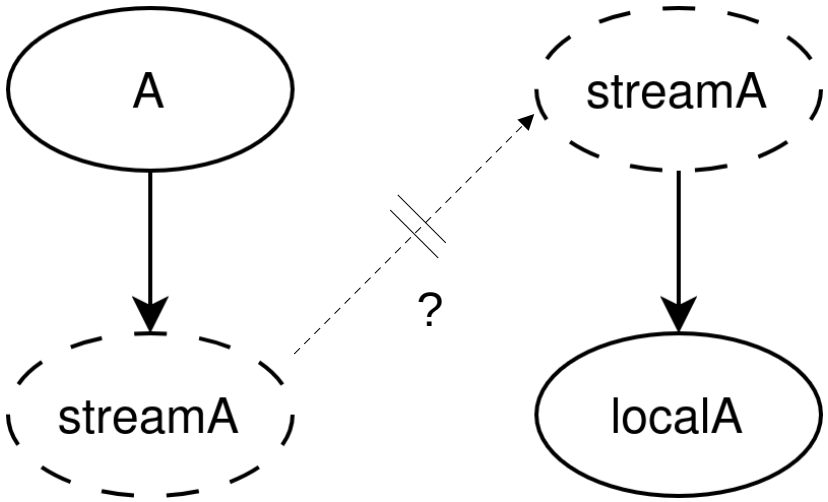
- ToHBMChannels Transformation:
  - The purpose is to assign an HBM data-channel location using a symbolic expression.
  - Will not initially consider optimizations regarding Bankgroups and Banks [I do not know how possible it is to do these optimizations in SoftHier simulator]
  - [Will discuss possible extensions with Chi]

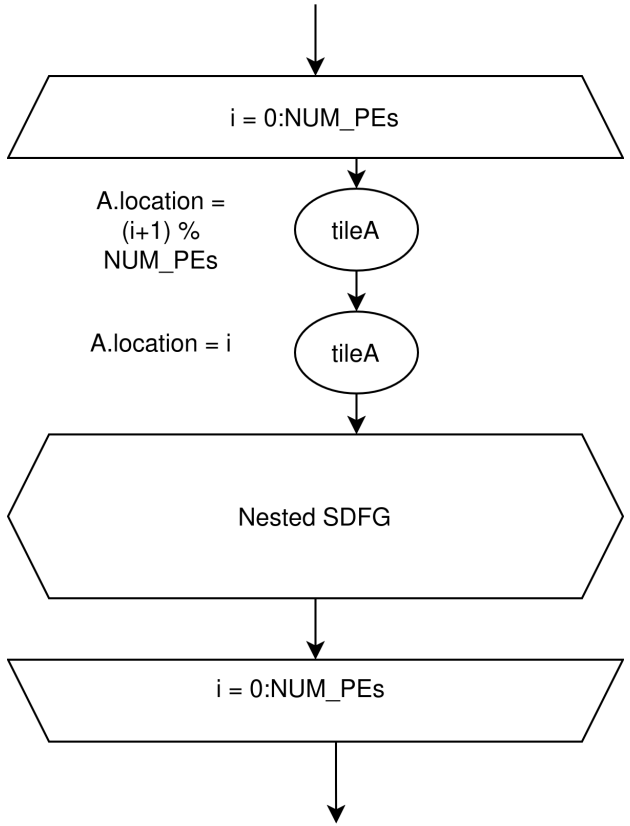
```
A.location = f(d0)
f(d0) = (d0 * CH) / N
```



A Full-Fledged SoftHier Backend

- BSP-Based Schedule Transformation Using `locations` instead of SDFG streams
  - For the SoftHier project the copies between PEs were implemented using streams. But this decouples the source of the copy from the destination of the copy.

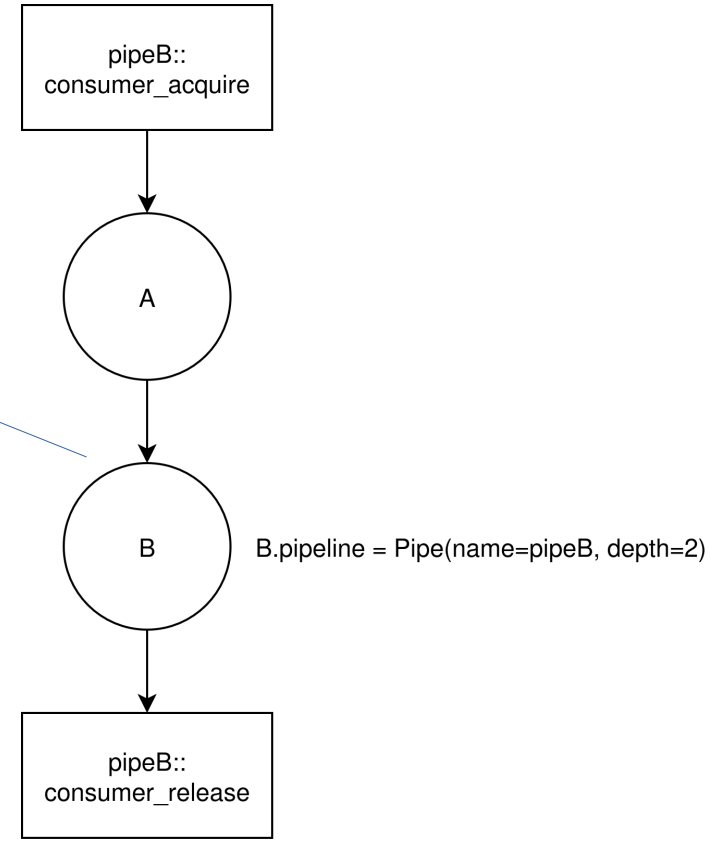




A Full-Fledged SoftHier Backend

- BSP-Based Schedule Transformation Using `locations` instead of SDFG streams
  - Will utilize `pipeline` objects to implement asynchronous copies.

```
// noop for SoftHier

dma_transfer(A[…], B[…], ...)

dma_sync()
```



A Full-Fledged SoftHier Backend

- The stretch goal of the thesis is the prune the transformation space and develop heuristics to speed-up the search of the kernel-optimizations and hardware-design candidates.
  - Planned is to include to integrate existing performance models (e.g. [1]) for the utilization of on-chip interconnect for the schedule transformations.

[1] https://arxiv.org/abs/2404.15888

DaCe Improvements – These Two Weeks' Bugfix PRs:

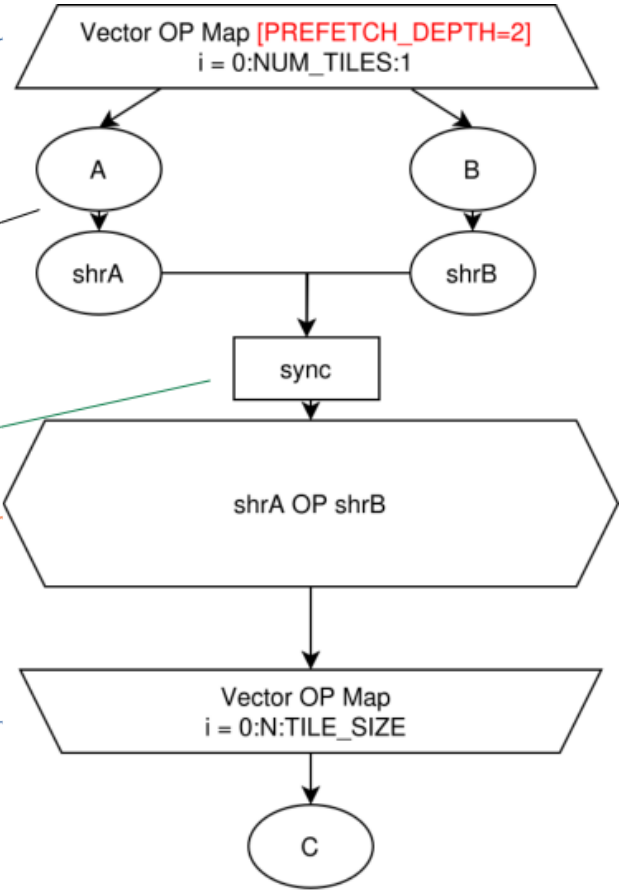
- [PR #2083] Fixes two issues in SDFG validation. In Merge Queue.
- [PR #2068] Fixes a bug in MapTiling and StripMining transformations, introduces more unit tests. Currently under review.
- [PR #2075] Fixes a bug in code generation for memlets. Merged.
- [PR #2067] Fixes bugs in kernel argument detection and generation, adds utility functions. Currently under development.

- [Backup Slides]

DaCe Improvements: Multiple Buffering Transformation

- [Ongoing] Multiple Buffering Pass (No PR Yet)
- The idea is to implement pipelined-for-loops (or pipelined sequential maps with sequential schedule)

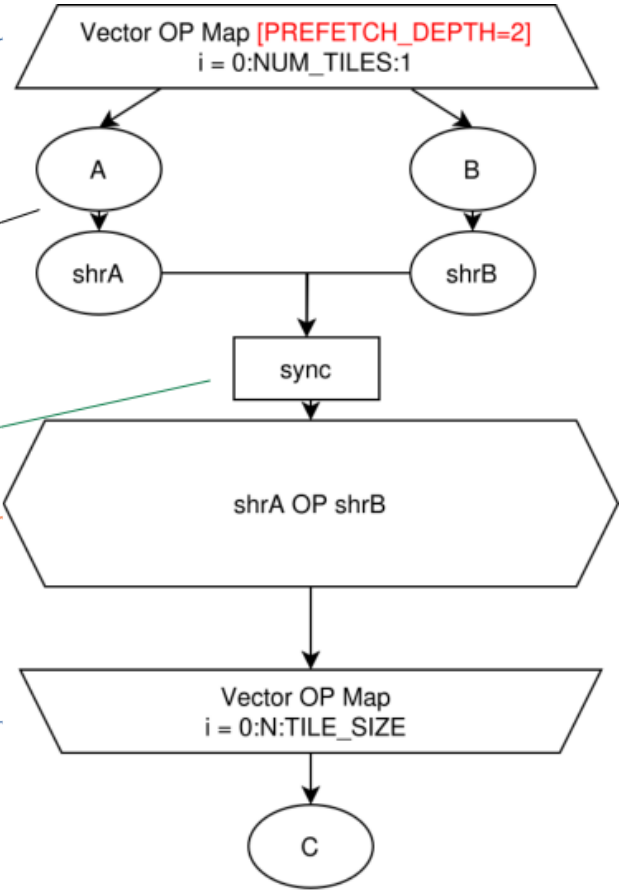
```
// Prefill Pipeline

for (i = 0; i < NUM_TILES; i++) {
 // FOR EACH PIPELINE OBJECT
 pipelineA.producer_acquire();
 if (not last_itereation(i+1)){
 memcpy_async(shrA[nextBufferIdx, A[...], ...,
 pipelineA);
 }
 pipelineA.producer_commit();

pipelineA.consumer_wait();

// Code for shrA OP shrB

pipelineA.consumer_release();

}
```



DaCe Improvements: Multiple Buffering Transformation

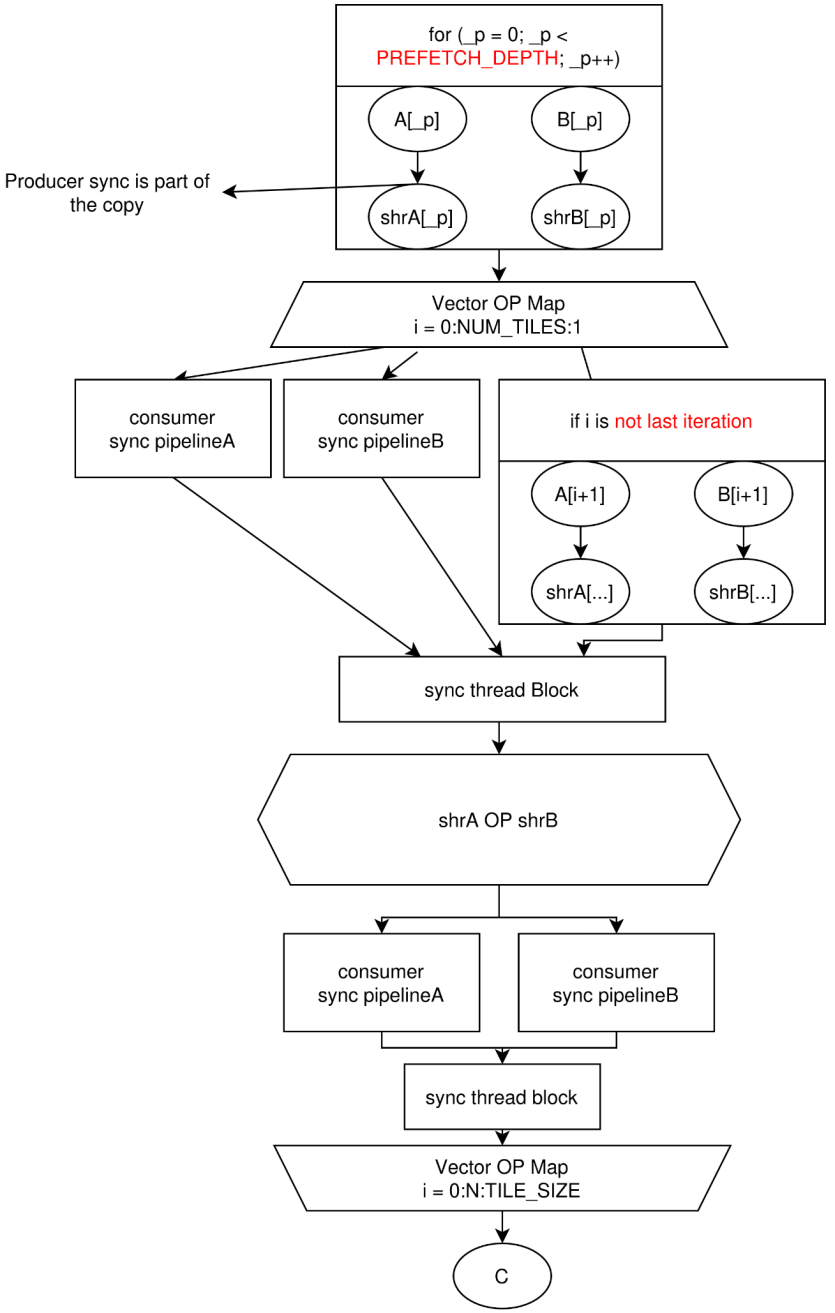
- [Ongoing] Multiple Buffering Pass (No PR Yet)
- A new SDFG element: Pipeline-For Block with a set PREFETCH_DEPTH

```
// Prefill Pipeline

for (i = 0; i < NUM_TILES; i++) {
 // FOR EACH PIPELINE OBJECT
 pipelineA.producer_acquire();
 if (not last_itereation(i+1)){
 memcpy_async(shrA[nextBufferIdx, A[...], ...,
 pipelineA);
 }
 pipelineA.producer_commit();

pipelineA.consumer_wait();

// Code for shrA OP shrB

pipelineA.consumer_release();

}
```



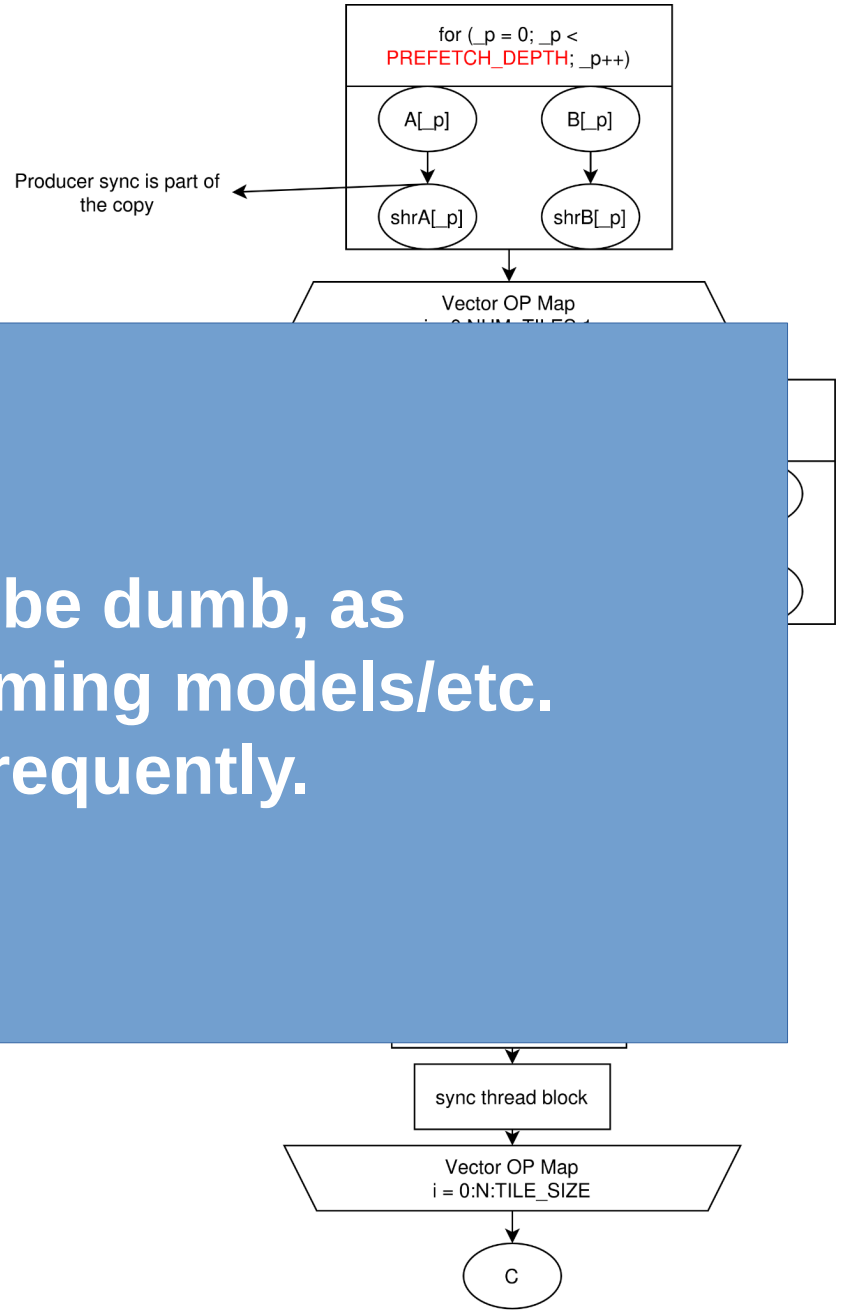
DaCe Improvements:
Multiple Buffering Transformation

- To keep the code-gen simple, the pipelined-for CFG is lowered to a set of:
  - Pipeline descriptors (attached to data descriptors)
  - Synchronization tasklets
  - For-CFGs



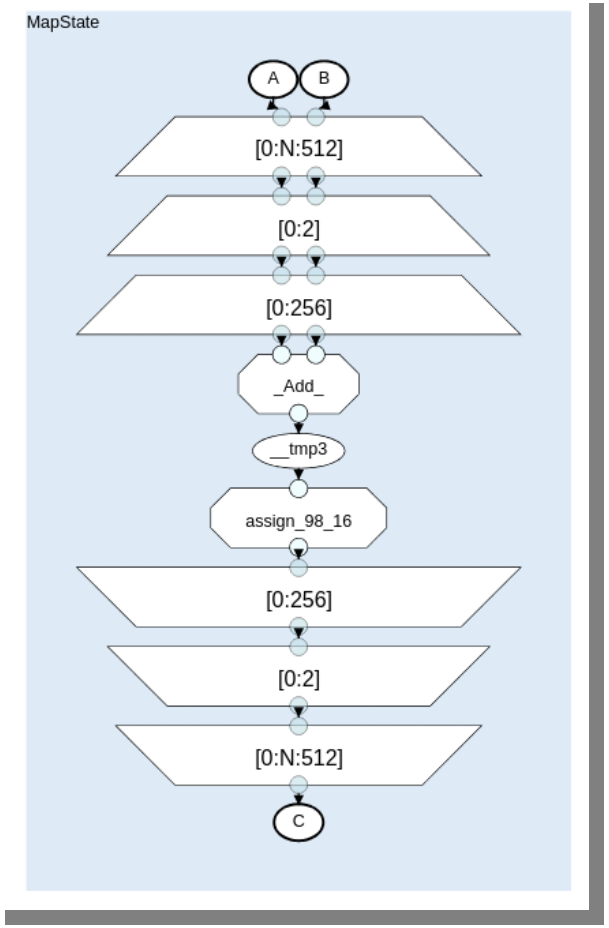
DaCe Improvements:
Multiple Buffering Transformation

- To keep the cc
  CFG is lowere
  - Pipeline desc
    descriptors)
  - Synchronizati
  - For-CFGs



Codegen should be dumb, as architecture/programming models/etc. change very frequently.

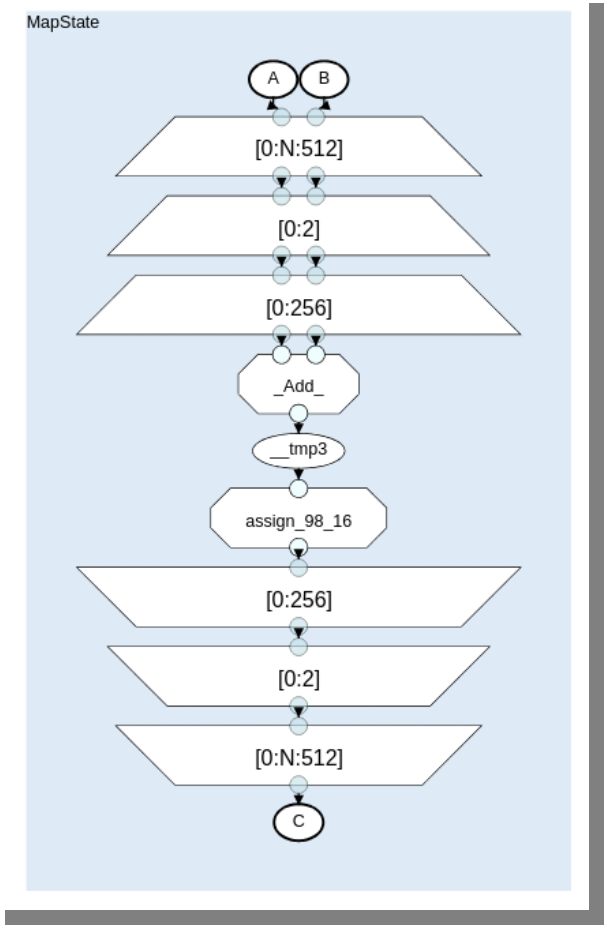
DaCe Improvements:
Multiple Buffering Transformation




```python
@dace.program
def kernel(
    A: dace.float64[N] @ dace.dtypes.StorageType.GPU_Global,
    B: dace.float64[N] @ dace.dtypes.StorageType.GPU_Global,
    C: dace.float64[N] @ dace.dtypes.StorageType.GPU_Global,
):
    for i in dace.map[0:N:512] @ dace.dtypes.ScheduleType.GPU_Device:
        for k in dace.map[0:2] @ dace.dtypes.ScheduleType.Sequential:
            for j in dace.map[0:256] @ dace.dtypes.ScheduleType.GPU_ThreadBlock:
                C[i + j + k * 256] = A[i + j + k * 256] + B[i + j + k * 256]
```

DaCe Improvements:

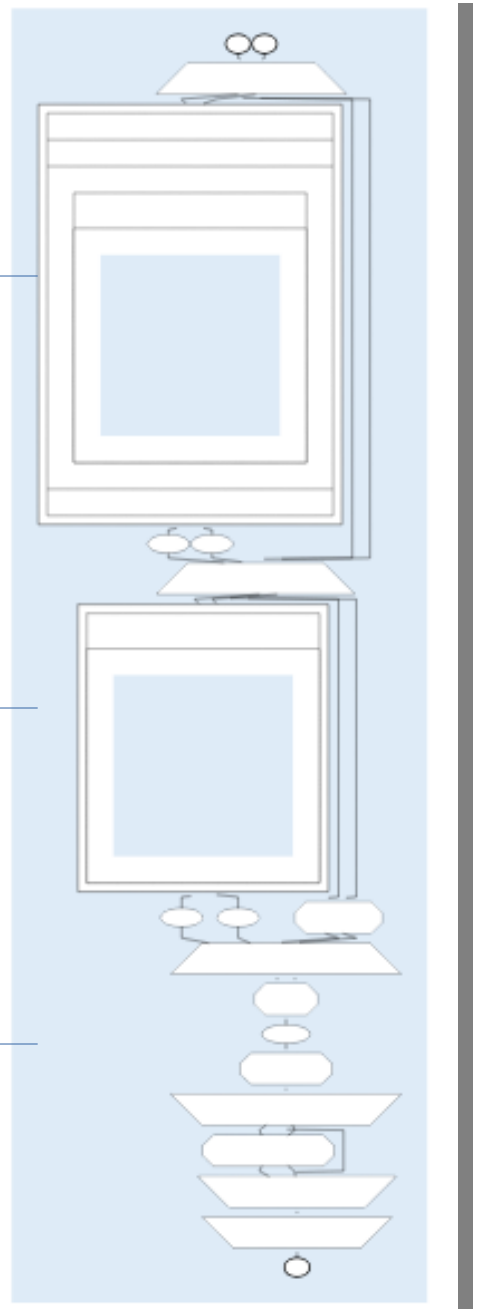
Multiple Buffering Transformation



Prefill State (The first `DEPTH - 1` iterations)

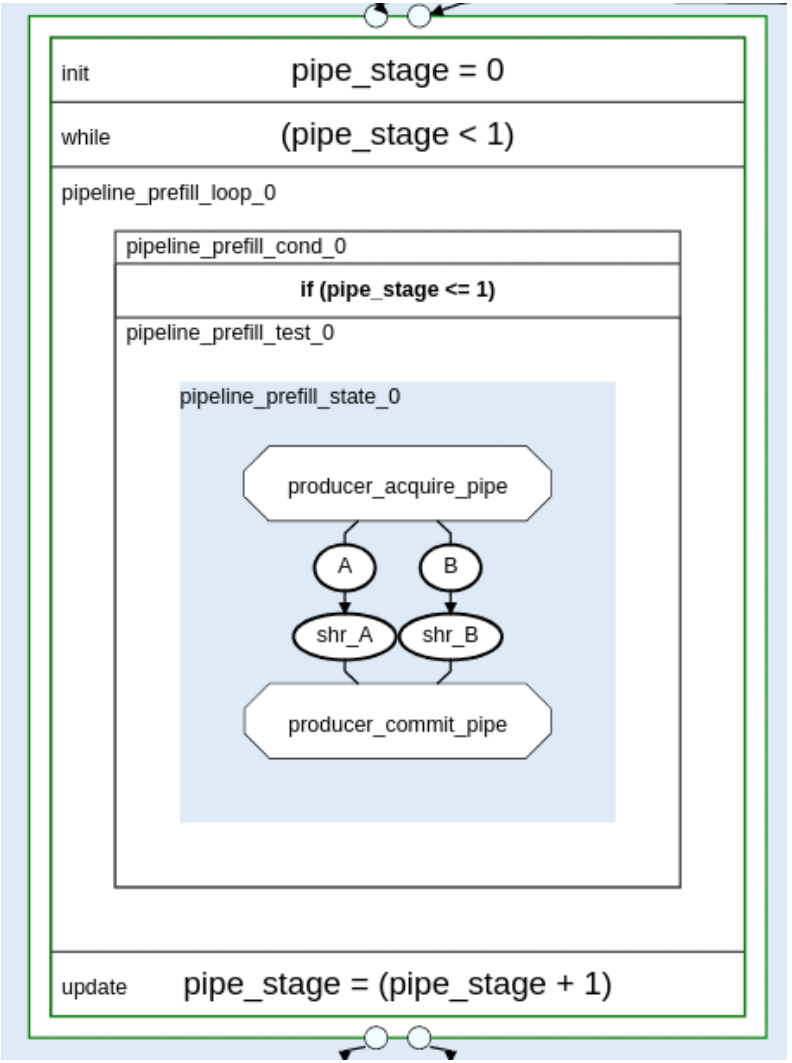
Prefetch state (`i+1` th iteration)

Kernel body

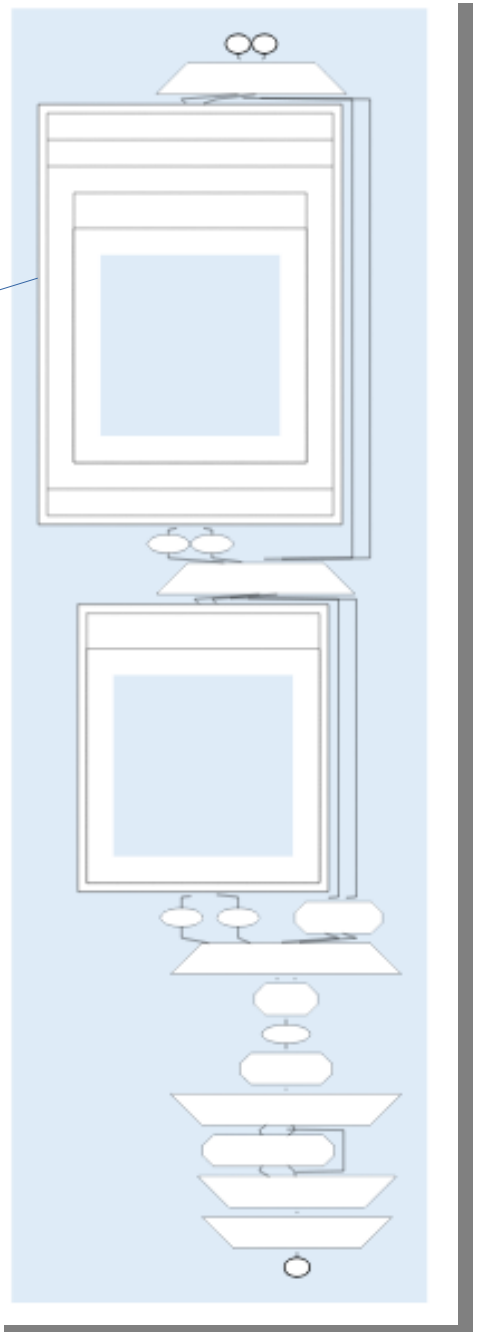


DaCe Improvements:

Multiple Buffering Transformation

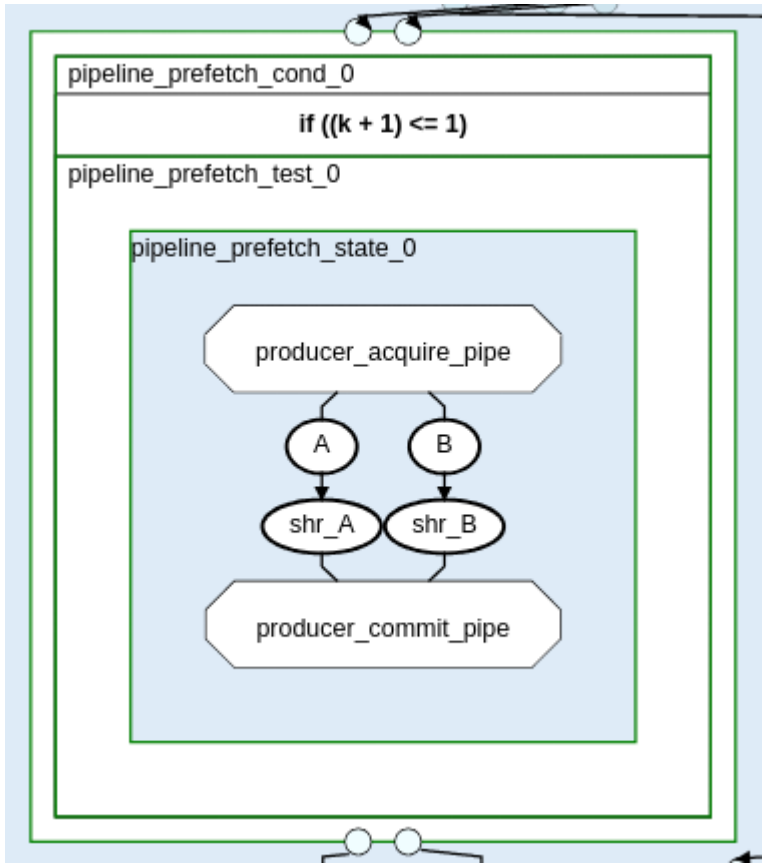


Pipeline acquire and commit are inserted as synchronization tasklets.



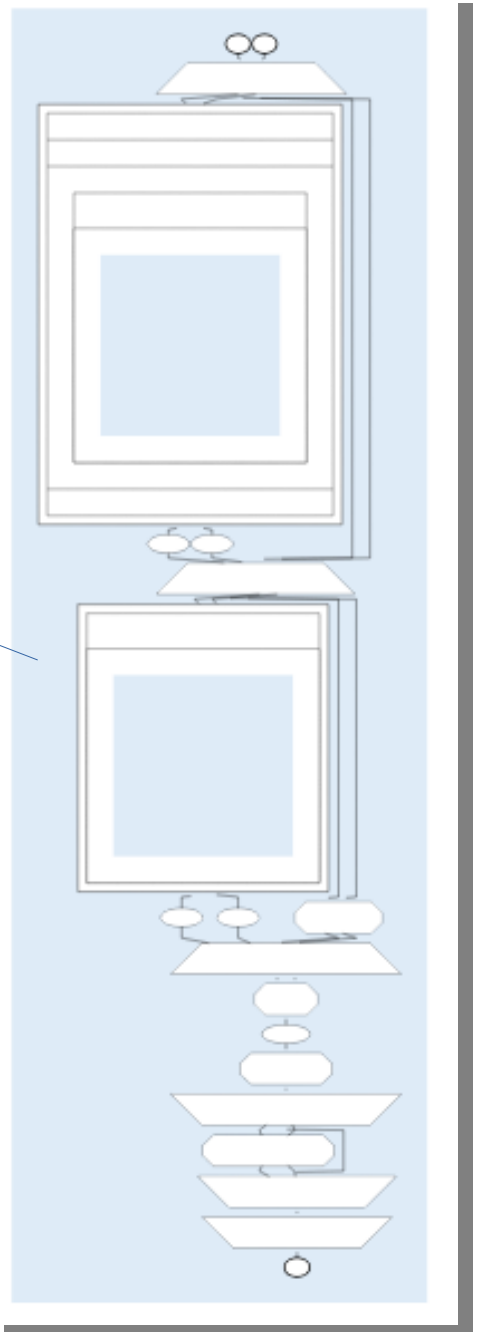
DaCe Improvements:

Multiple Buffering Transformation



Prefetch state is the same as the prefill.

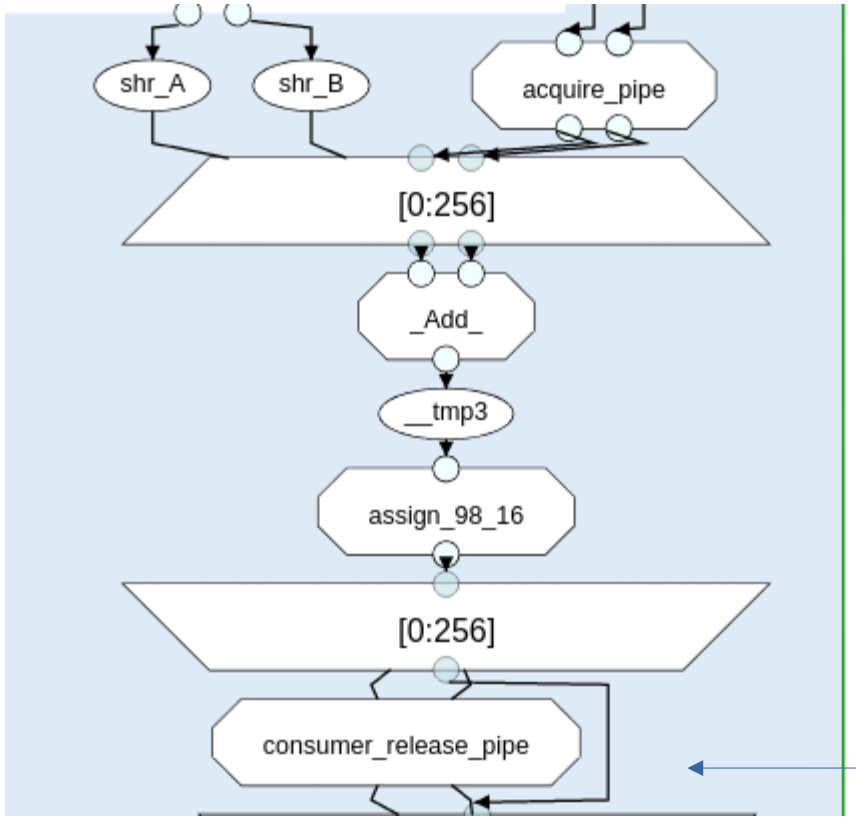
It prefetches memory required for the next iteration.

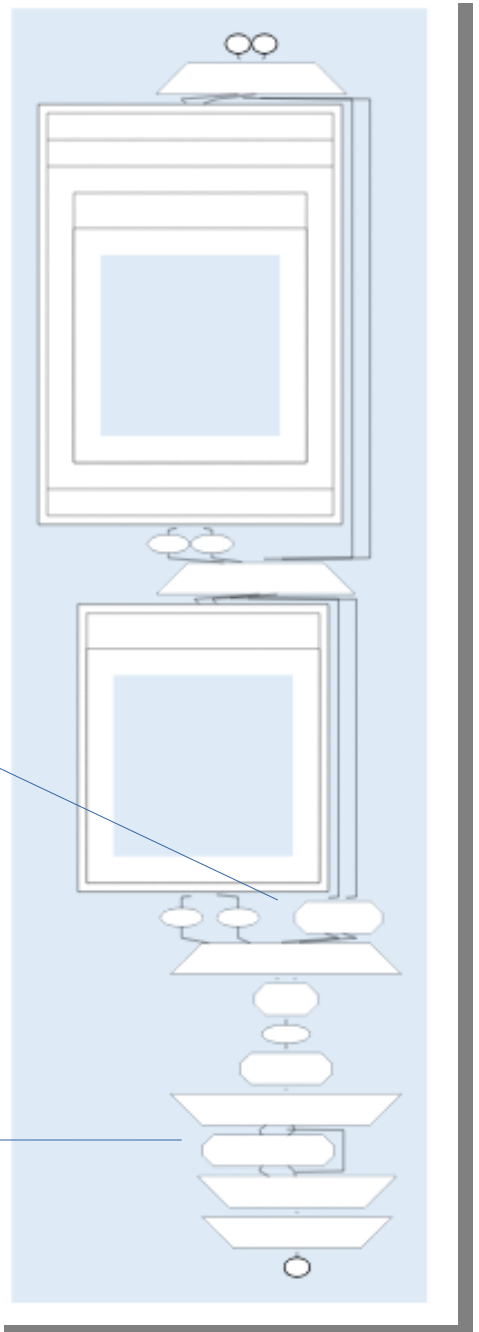


DaCe Improvements:

Multiple Buffering Transformation

Consumer pipeline is synchronized using tasklets too.





DaCe Improvements:

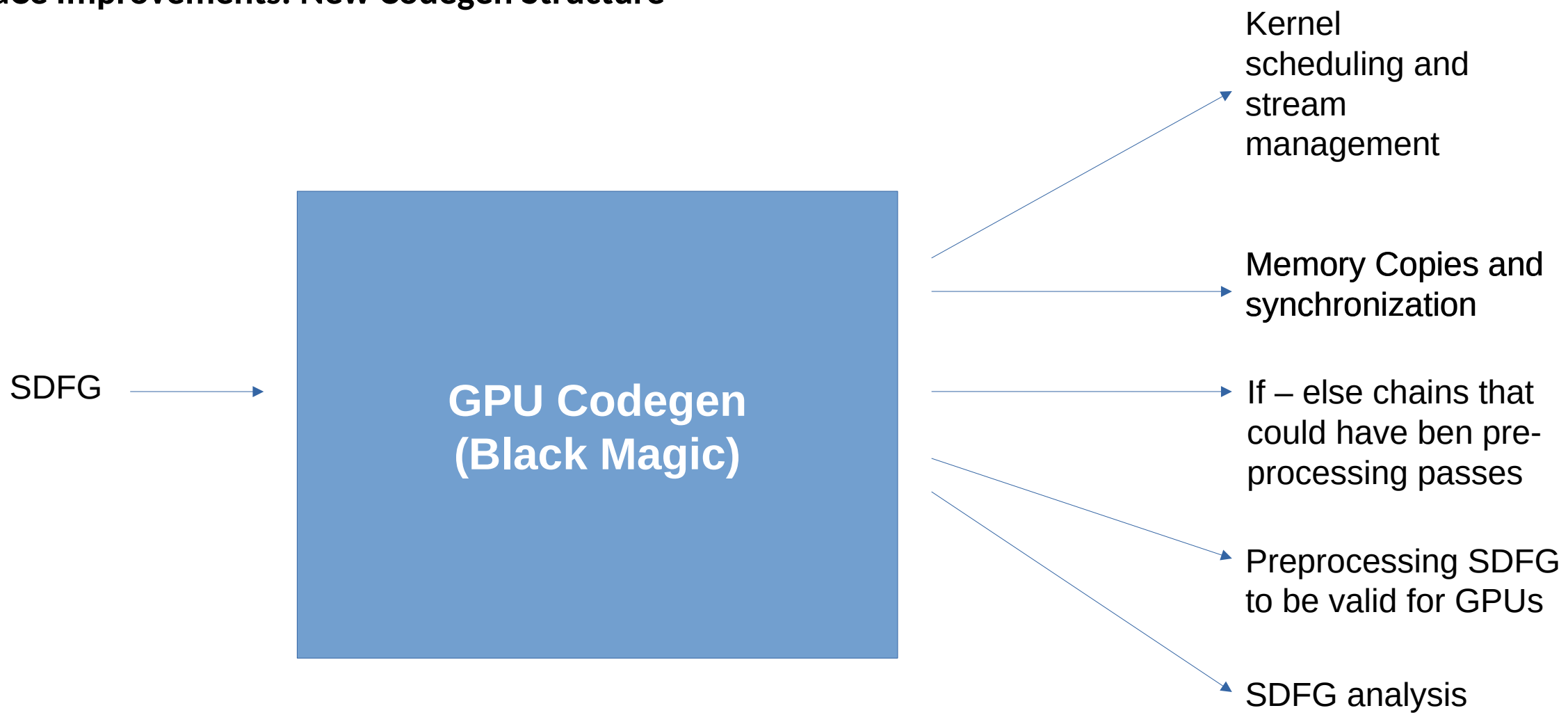
Multiple Buffering Transformation

DaCe has not anticipated the usage of tasklets for synchronization and the validation breaks on multiple points. Multiple small PRs and discussions are currently ongoing.

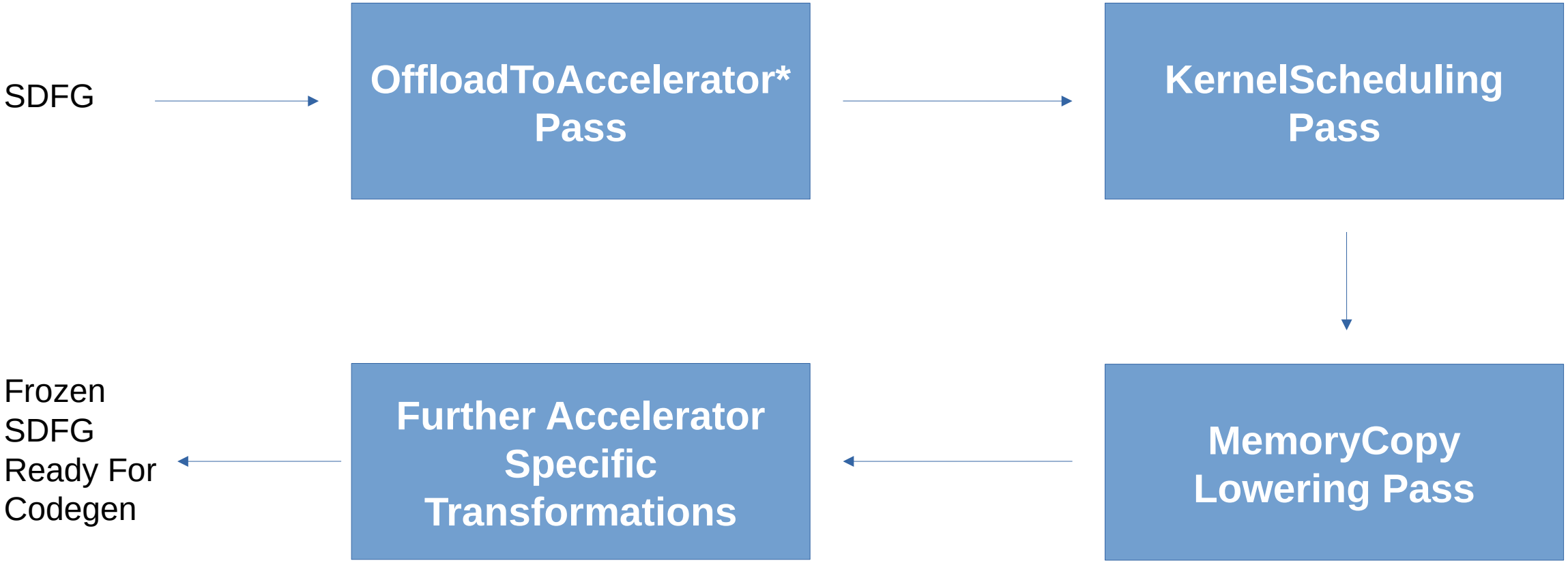
```
dace.sdfg.validation.InvalidSDFGEdgeError: Memlet creates an invalid path (sink node release_pipelines should be a data node) (at state
MapState, edge  (kernel_27_4_28_8_29[j=0:256]:None -> release_pipelines:None))
```

```
  File "/home/primrose/Work/dace/dace/codegen/targets/cpu.py", line 1037, in process_out_memlets
    raise SyntaxError("Cannot copy memlet without a local connector: {} to {}".format(
SyntaxError: Cannot copy memlet without a local connector: producer acquire pipe to B
```

DaCe Improvements: New Codegen Structure



DaCe Improvements: New Codegen Structure



*https://docs.google.com/document/d/13Pl4A8u5YJgTkZvhKCoJp8t693tvHeEYXfUvDtyonDI/edit?usp=sharing

Modularizing Codegen: OffloadToAccelerator Pass

- We have a ***hacked*** ToGPU pass
  - It fails for many non-toy SDFGs
  - Port to any accelerator currently starts by copy-pasting the codegen and the ToGPU pass and replace names (very inefficient)
  - During the ICON project I have identified many shortcomings, and I wrote a design document for the improved pass – to support multiple accelerators.
  - ~10 pages long

  - => Looking for a student to start in September to implement the improved pass

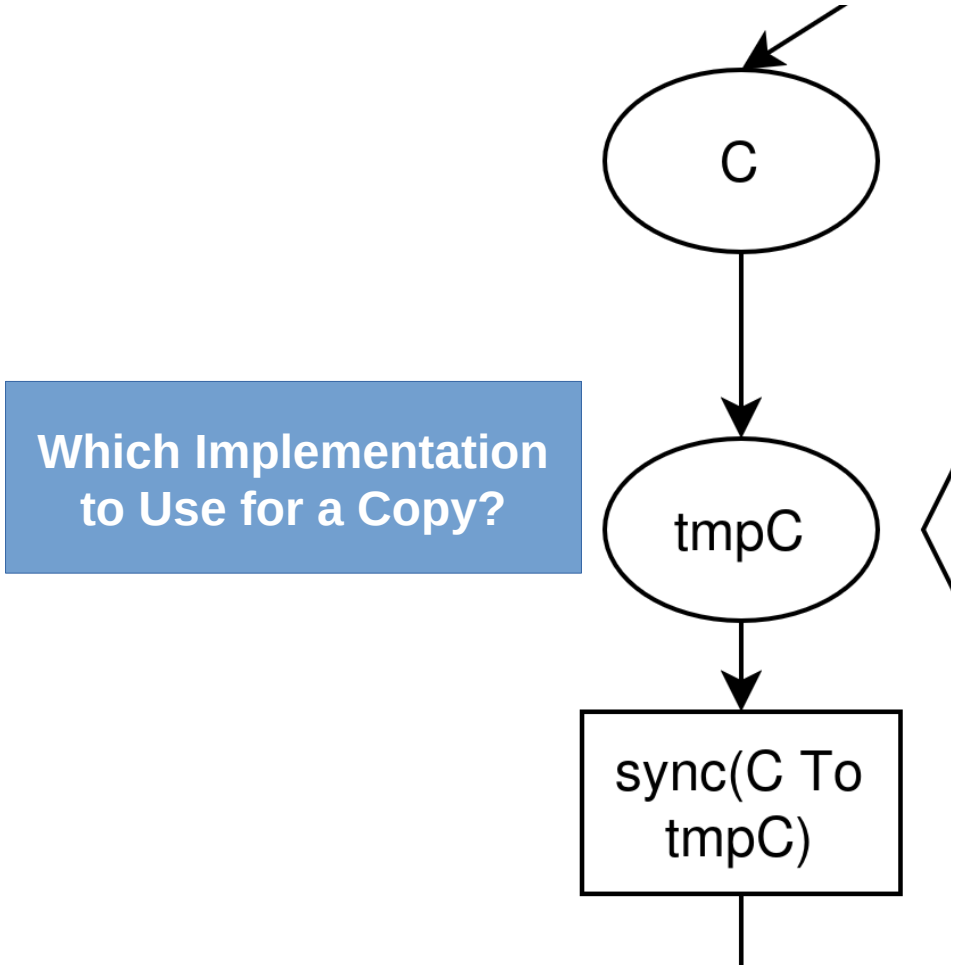
Modularizing Codegen: KernelScheduling Pass

- GPU codegen tries to automatically assign kernels to streams
  - Only the option to offload everything to stream 0 works
  - Modularizing the schedule can spawn a multitude of parallel projects, many algorithms exist for graph coloring and scheduling to use here.
  - => I have a student working on the infrastructure to support scheduling kernels at the SDFG level

Modularizing Codegen: MemoryCopy Lowering Pass

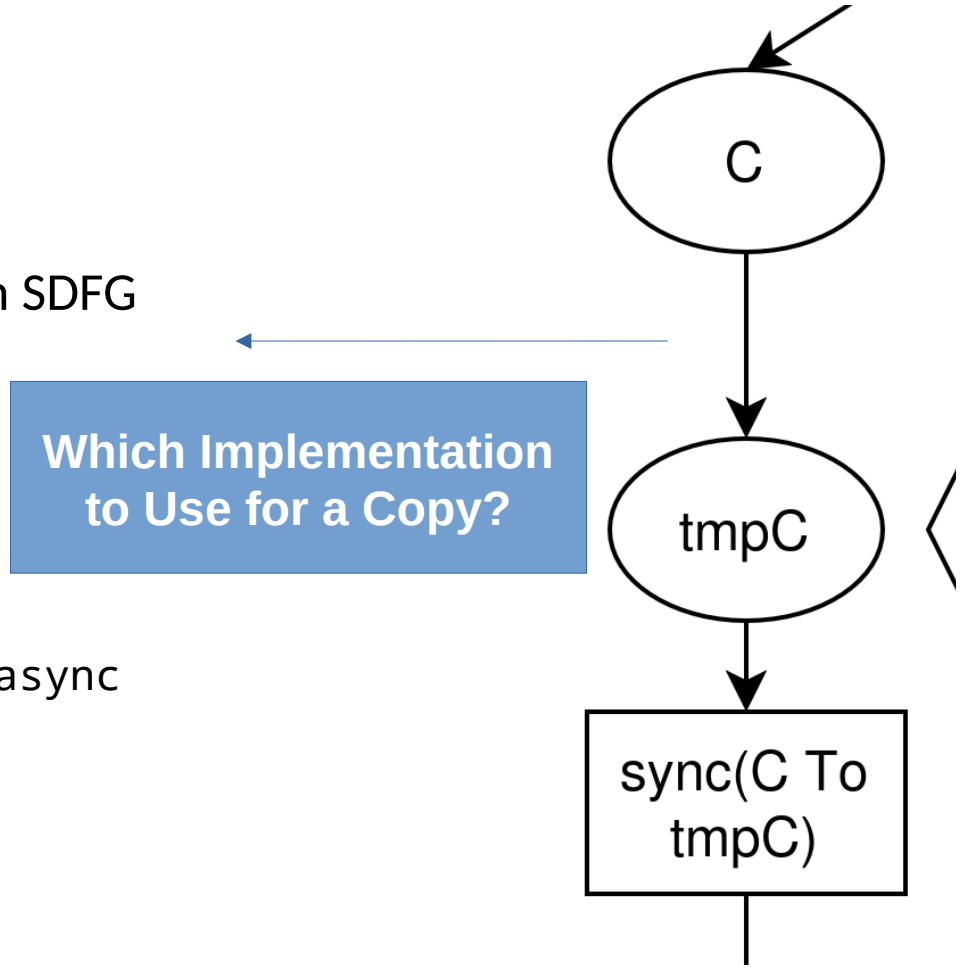
- GPU Codegen supports only synchronous copies
  - Multiple strategies exist to copy memory from location A to B, where it is infeasible (possibly impossible) to detect which implementation will perform the best.

Copy Strategies



Copy Strategies

- Copy strategies on how to lower copies from SDFG IR to code:
  - Sync using registers
  - Async using → `cuda::memcpy_async`
  - Async + Pipeline using → `cuda::memcpy_async`
  - Using TMA → `CUtensorMap + cuda::memcpy_async`
  - Using CuTe library (Asnyc or With TMA) → `Copy_Atom<SM90_TMA...>`

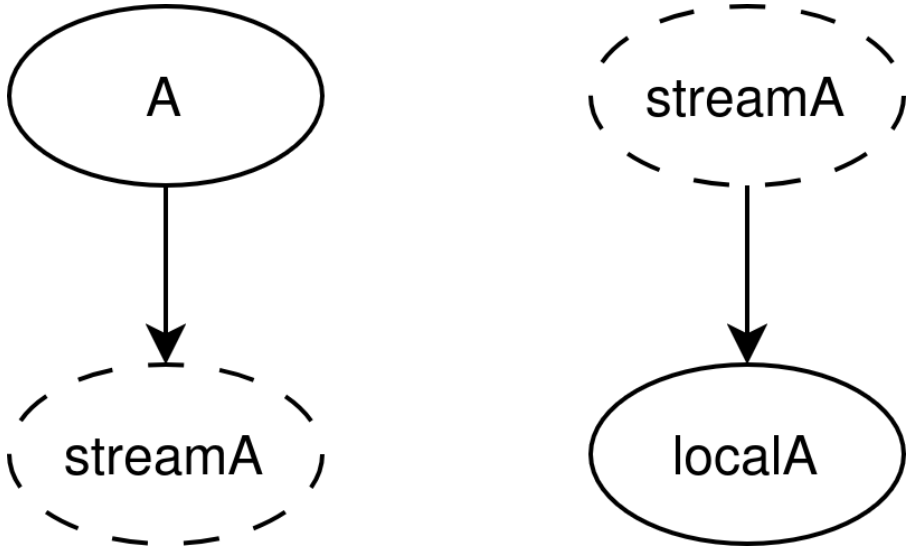


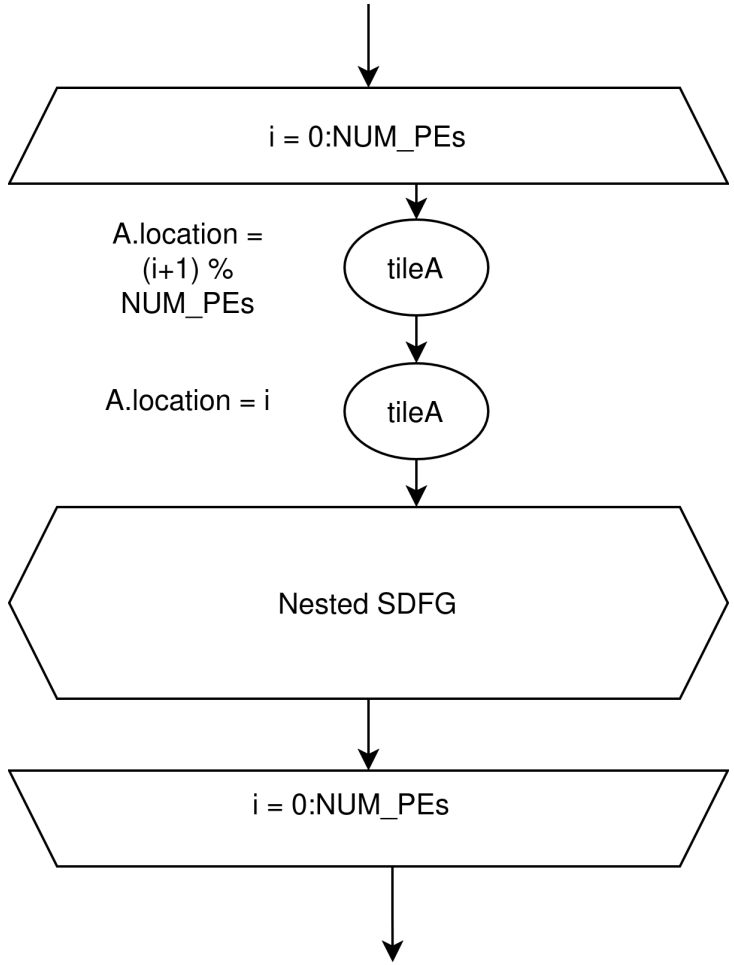
DaCe Quality-of-Life Improvements: Memory Locations

- [Ongoing] Memory Locations For PE→PE Memory Transfers (No PR Yet)

For the SoftHier project the copies between PEs were implemented using streams.

But this decouples the source of the copy from the destination of the copy.



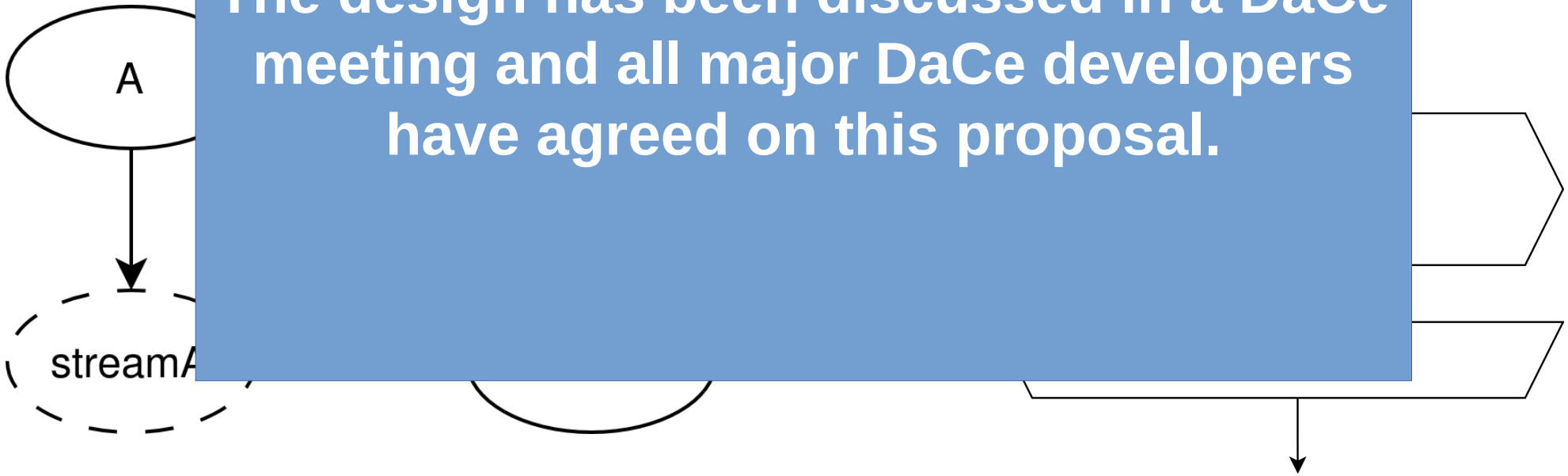


DaCe Quality-of-Life Improvements: Memory Locations

- [Ongoing] Memory Locations For PE→PE Memory Transfers (No PR Yet)

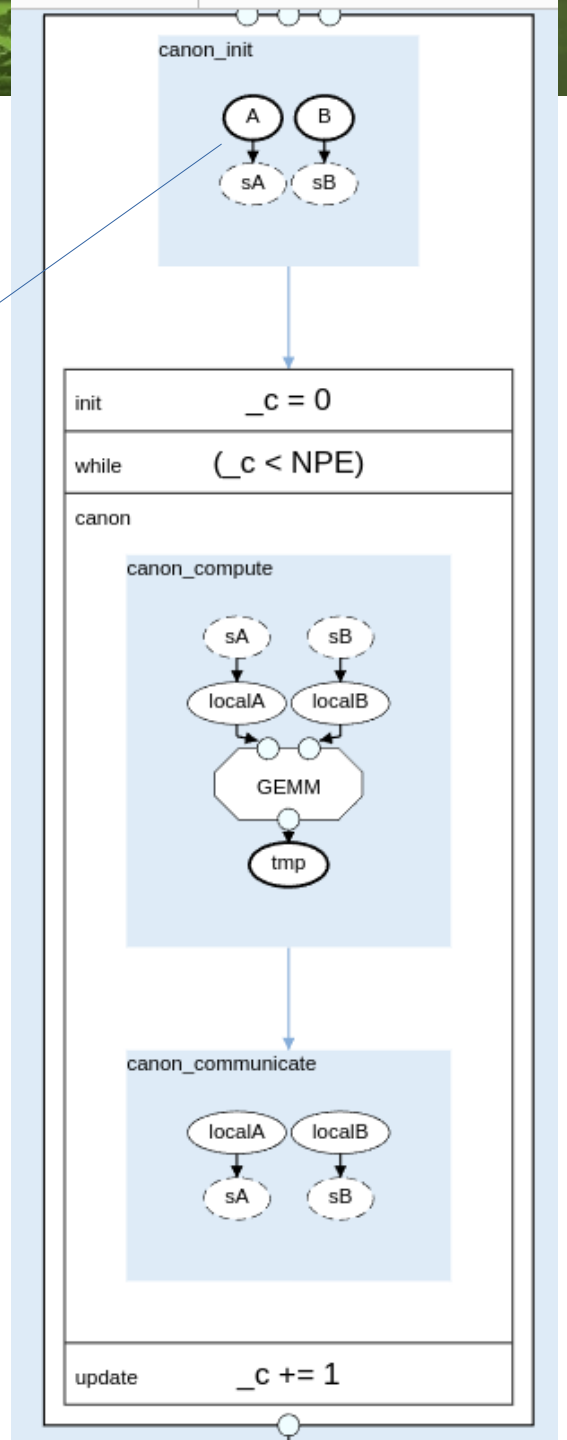
For the SoftHier
implemented us

But this decoupl
destination of th



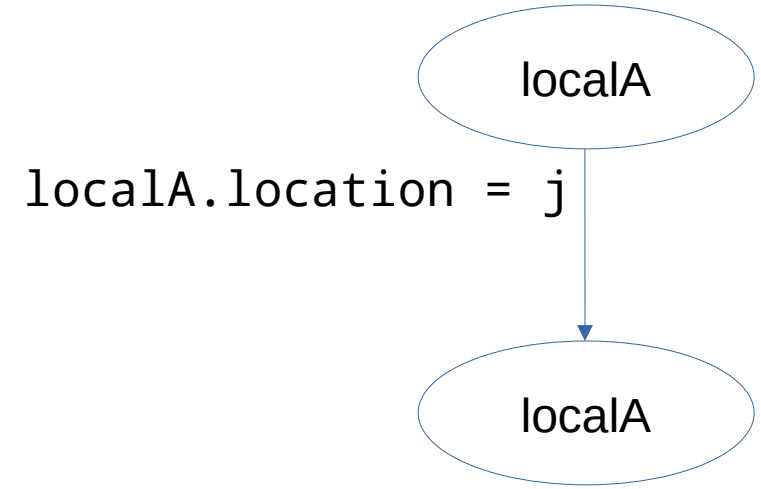
The design has been discussed in a DaCe meeting and all major DaCe developers have agreed on this proposal.

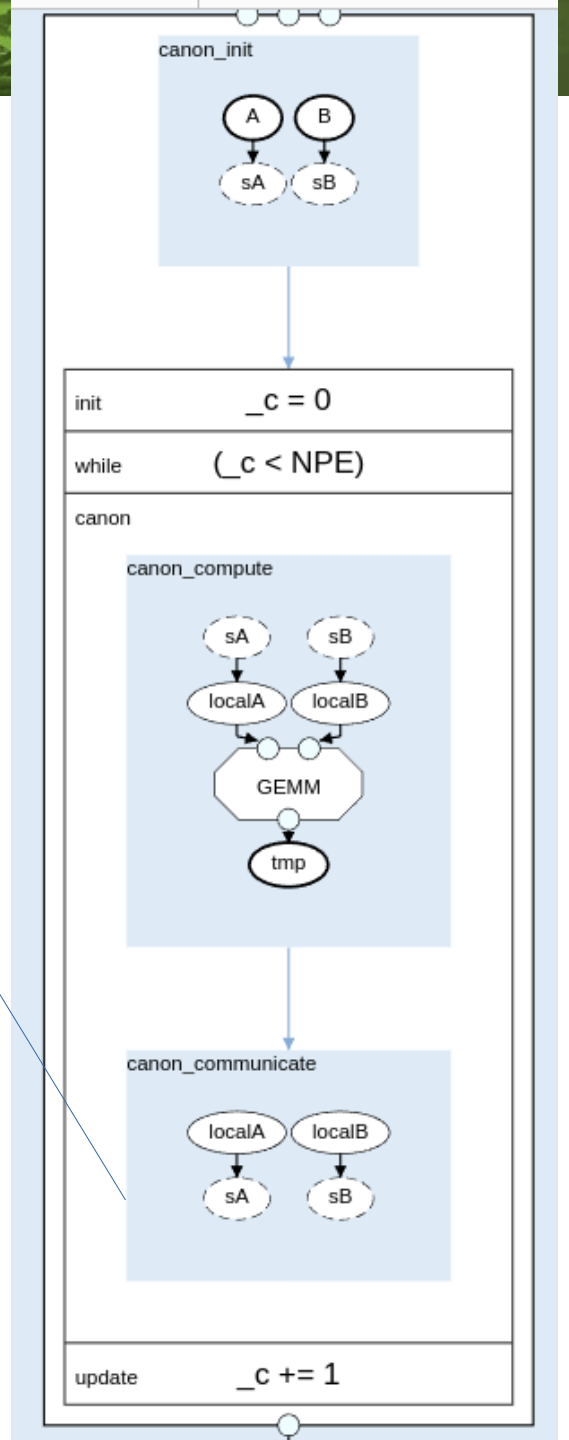
Intra-PE communication can be performed through streams, but this feature will be deprecated.



The copy will be performed as:
`localA[j]` → `localA[i]`

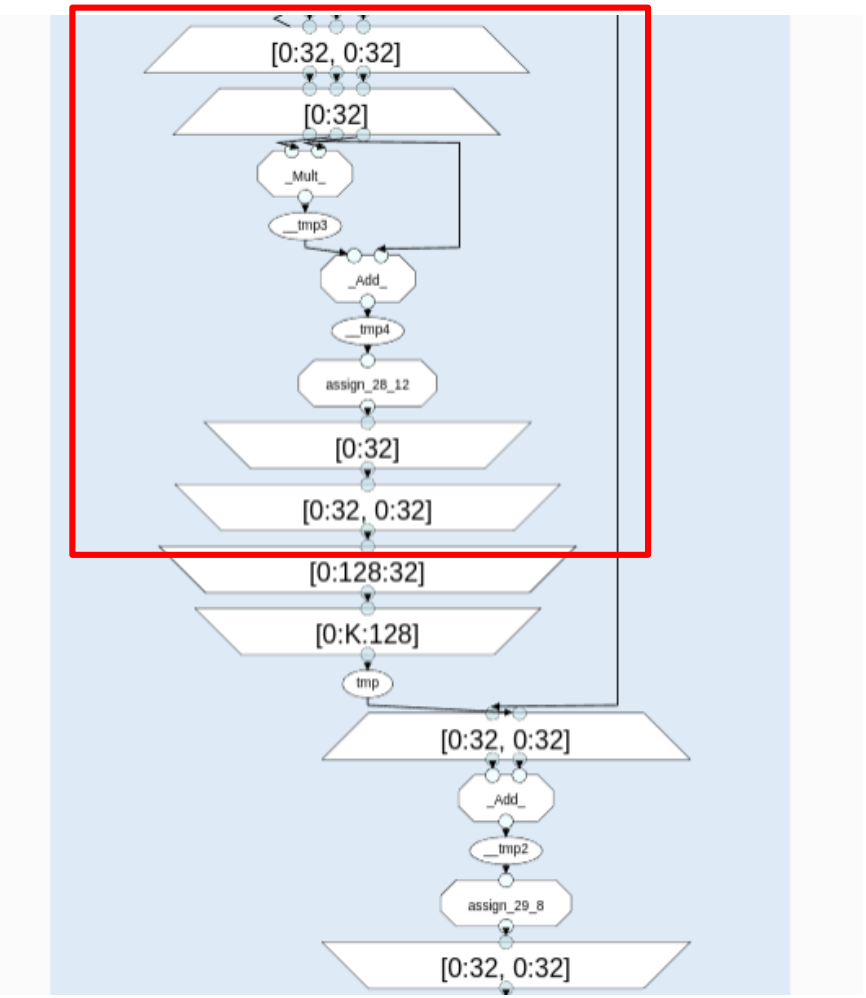
Where **`i`** and **`j`** are PE ids.

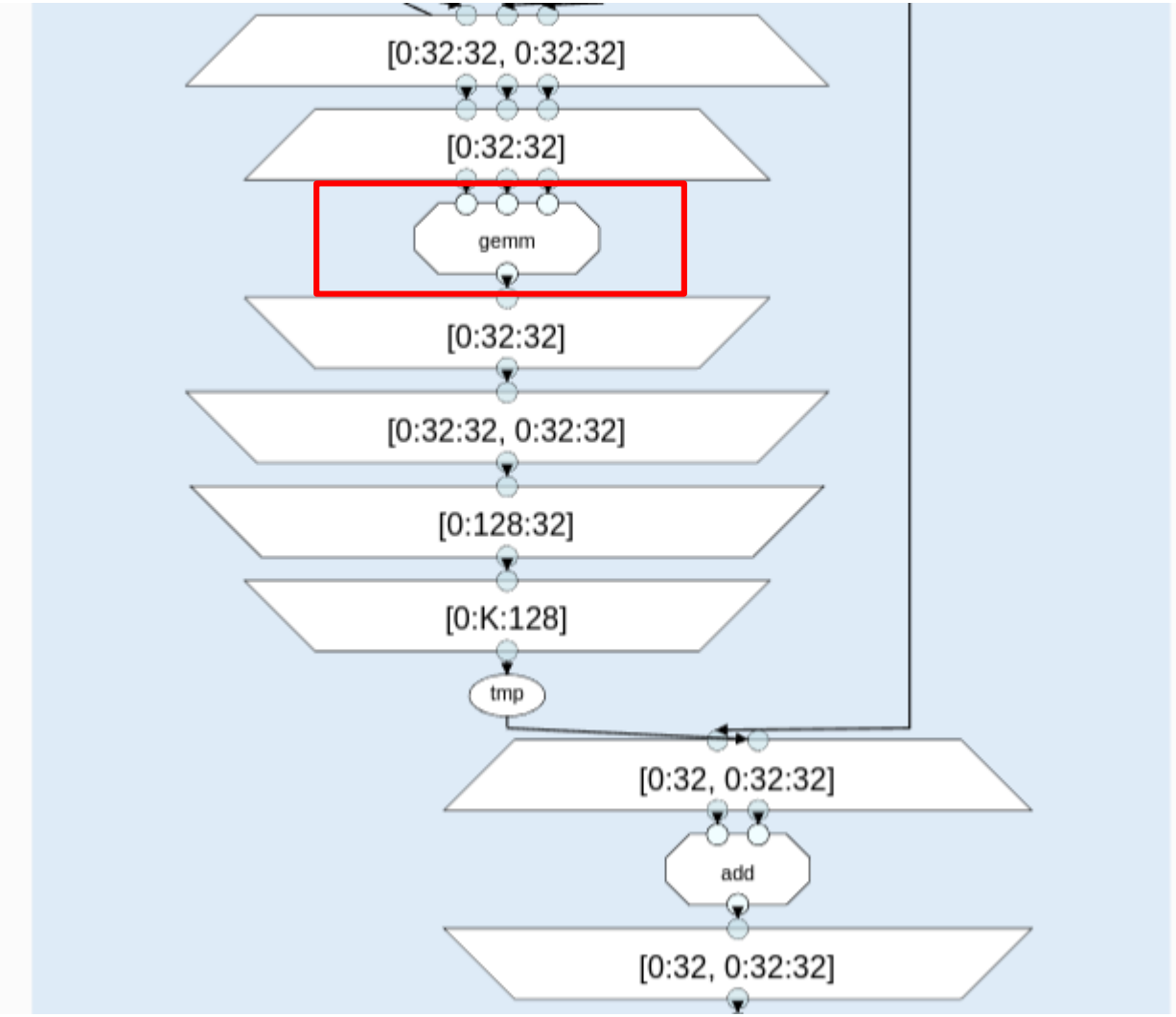




Transformation Updates

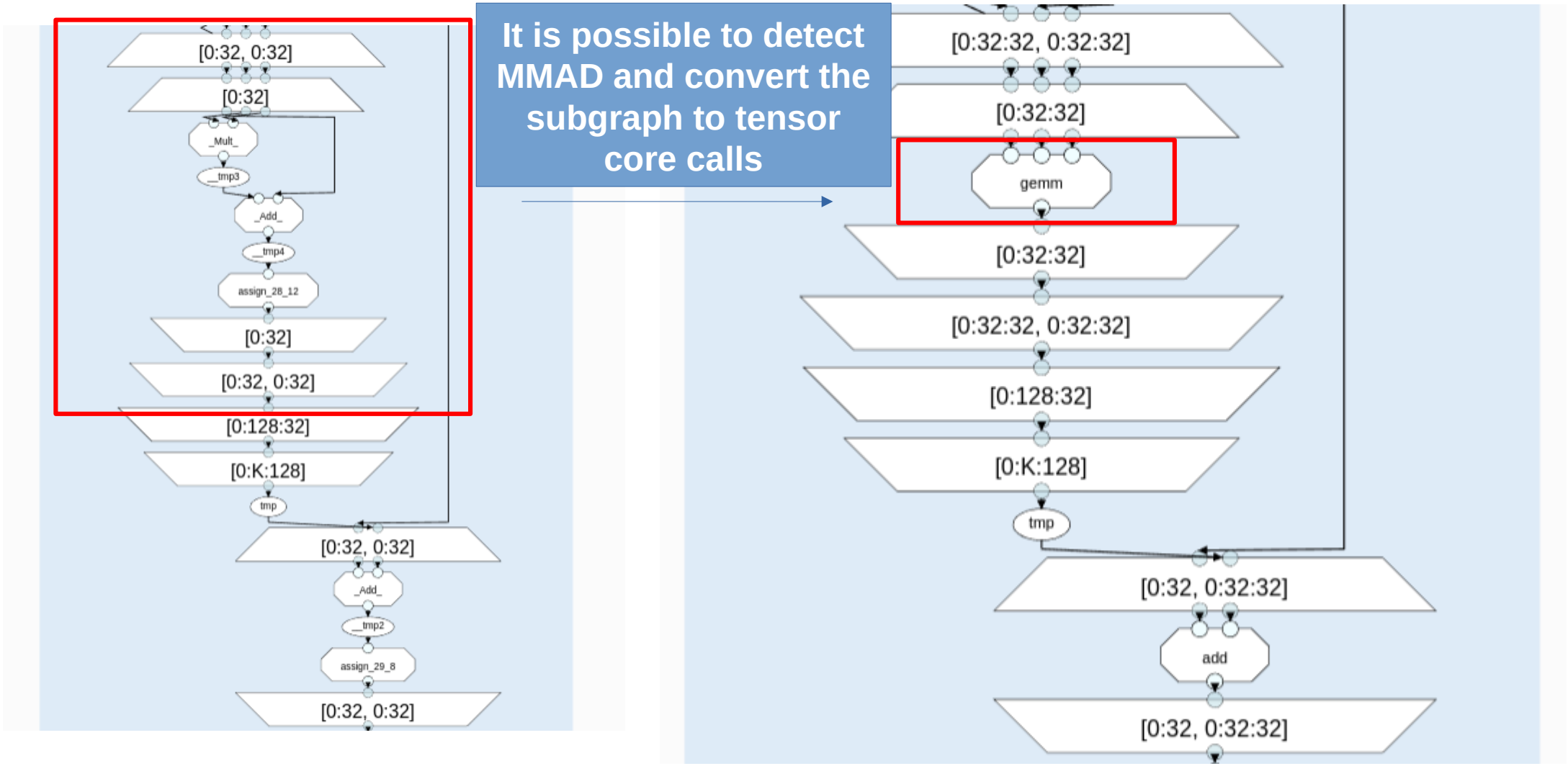
- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.





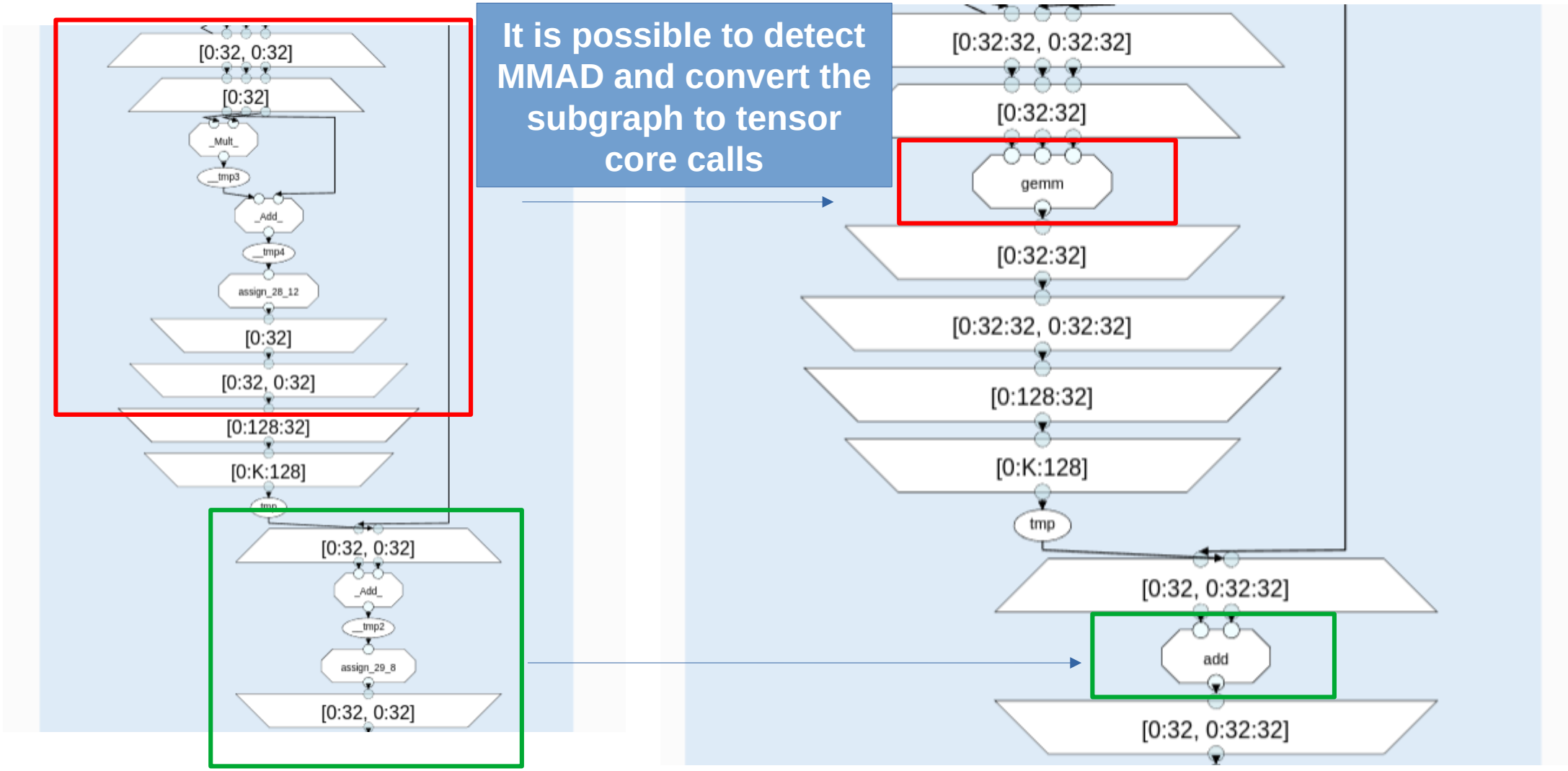
Transformation Updates

- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.



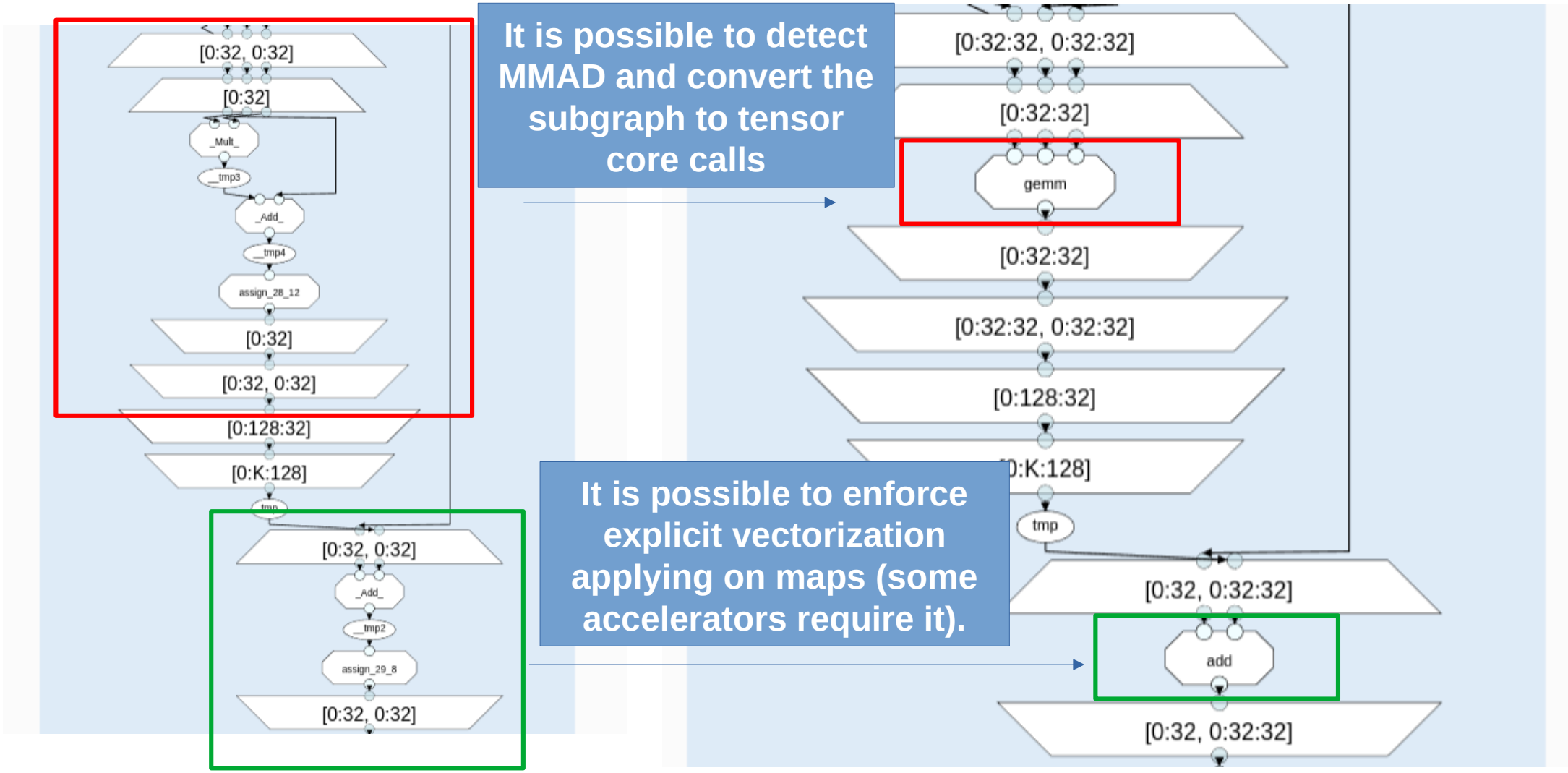
Transformation Updates

- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.



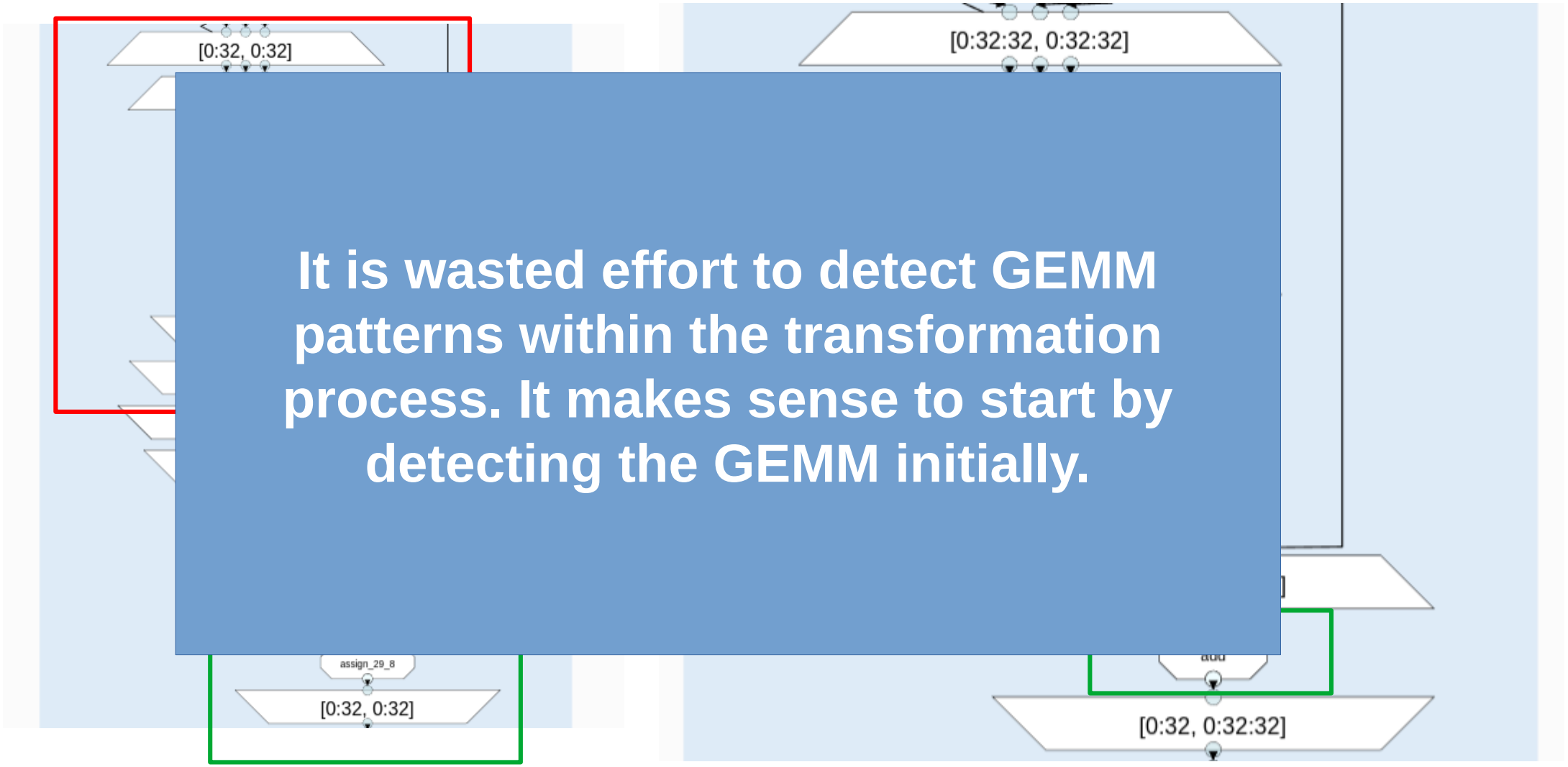
Transformation Updates

- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.



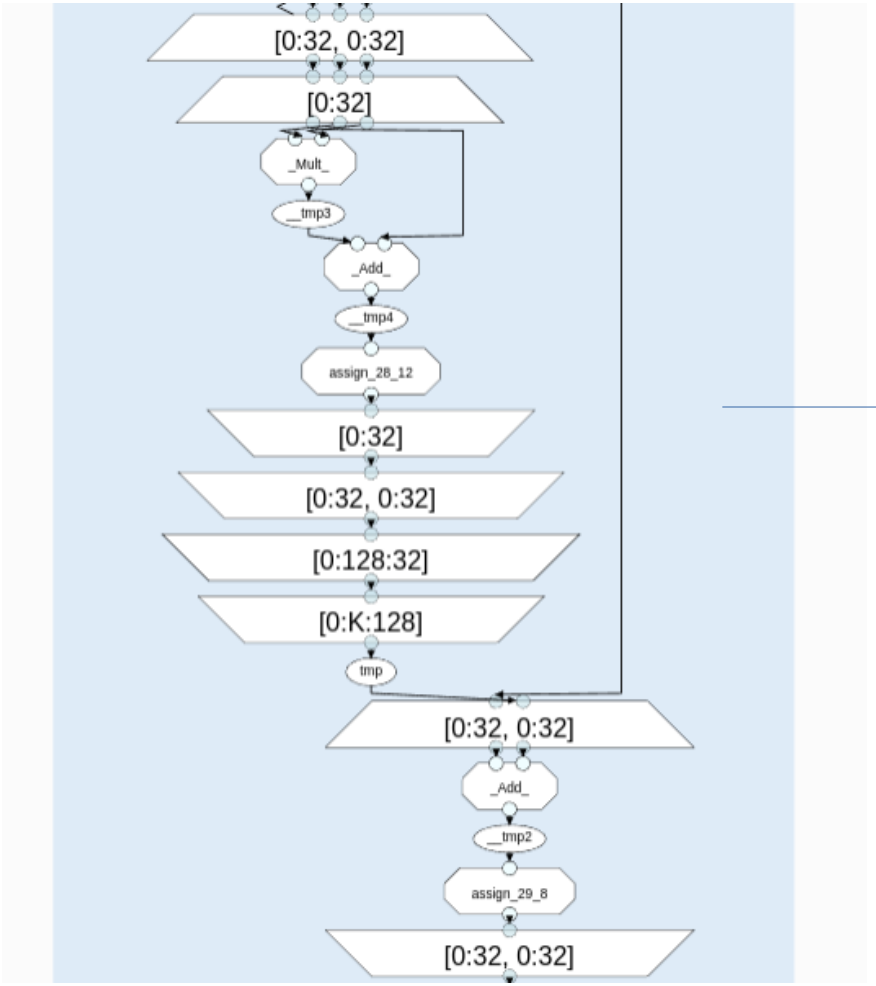
Transformation Updates

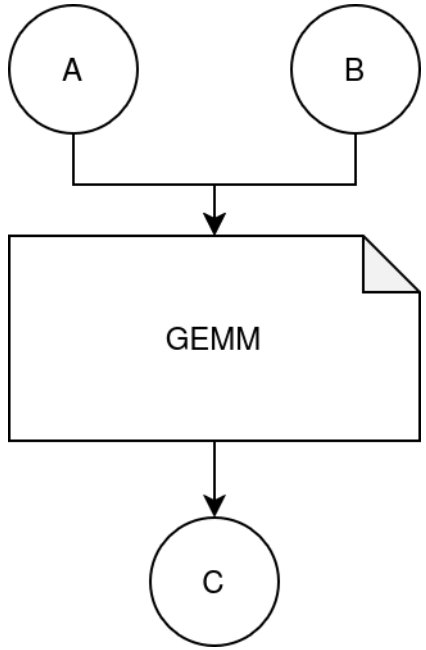
- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.



Transformation Updates

- The pattern-matcher can map MMAD and vectorizable operations to their respective computational units.





Lower it already knowing that this kernel is a GEMM kernel.

- Decoupling FP-Implementations From Source Code
  - Why HPC code always uses double precision?

A Real Type For SDFGs

Towards a Compiler for Reals

EVA DARULOVA and VIKTOR KUNCAK, Ecole Polytechnique Federale de Lausanne

8

Numerical software, common in scientific computing or embedded systems, inevitably uses a finite-precision approximation of the real arithmetic in which most algorithms are designed. In many applications, the round-off errors introduced by finite-precision arithmetic are not the only source of inaccuracy, and measurement and other input errors further increase the uncertainty of the computed results. Adequate tools are needed to help users select suitable data types and evaluate the provided accuracy, especially for safety-critical applications.

We present a source-to-source compiler called Rosa that takes as input a real-valued program with error specifications and synthesizes code over an appropriate floating-point or fixed-point data type. The main challenge of such a compiler is a fully automated, sound, and yet accurate-enough numerical error estimation. We introduce a unified technique for bounding roundoff errors from floating-point and fixed-point arithmetic of various precisions. The technique can handle nonlinear arithmetic, determine closed-form symbolic invariants for unbounded loops, and quantify the effects of discontinuities on numerical errors. We evaluate Rosa on a number of benchmarks from scientific computing and embedded systems and, comparing it to the state of the art in automated error estimation, show that it presents an interesting tradeoff between accuracy and performance.

CCS Concepts: • **Software and its engineering** → **Formal software verification**; *Specification languages*; Source code generation;

Additional Key Words and Phrases: Roundoff error, floating-point arithmetic, fixed-point arithmetic, verification, compilation, sensitivity analysis, discontinuity error, loops

ACM Reference Format:
Eva Darulova and Viktor Kuncak. 2017. Towards a compiler for reals. ACM Trans. Program. Lang. Syst. 39, 2, Article 8 (March 2017), 28 pages.
DOI: http://dx.doi.org/10.1145/3014426

1. INTRODUCTION

Much of today's software is numerical in nature. While domains such as scientific computing and embedded systems may appear to differ considerably, they have in common that many of their algorithms are designed in real arithmetic but ultimately have to be implemented in finite precision. This inevitable approximation introduces roundoff errors, which individually may be small but can quickly accumulate or get magnified

This work is supported in part by the European Research Council (ERC) project "Implicit Programming".
A preliminary version of one part of this work appeared in the conference paper "Sound Compilation of Reals," presented at the 2014 ACM SIGPLAN-SIGACT International Conference on Principles of Programming Languages (POPL). The current submission is a complete rewrite (except for algorithm 6) of the preliminary version, presents new and improved techniques, as well as new experimental evaluation.
Authors' addresses: E. Darulova, Campus E1.5, 66125 Saarbrucken, Germany; email: eva@mpi-sws.org; V. Kuncak, EPFL IC LARA INR318, Station 14, CH-1015 Lausanne, Switzerland; email: viktor.kuncak@epfl.ch.

DOI: http://dx.doi.org/10.1145/3014426

- Motivation taken from "Towards a Compiler for Reals" paper and tool "Rosa"

A `Real` Type For SDFGs

1. Define input intervals for input arguments

```
1  def rigidBodyControl1(x1: Real, x2: Real, x3: Real): Real = {
2    require (-15 <= x1 && x1 <= 15 && -15 <= x2 && x2 <= 15 && -15 <= x3 && x3 <= 15)
3    -x1*x2 - 2*x2*x3 - x1 - x3
4  } ensuring (res => res <= 705.0 && res +/- 6e-13)
5
```

A `Real` Type For SDFGs

1. Define input intervals for input arguments

```
1  def rigidBodyControl1(x1: Real, x2: Real, x3: Real): Real = {
2    require (-15 <= x1 && x1 <= 15 && -15 <= x2 && x2 <= 15 && -15 <= x3 && x3 <= 15)
3    -x1*x2 - 2*x2*x3 - x1 - x3
4  } ensuring (res => res <= 705.0 && res +/- 6e-13)
5
```

2. Describe precision requirements on the outputs

A `Real` Type For SDFGs

1. Define input intervals for input arguments

```
1  def rigidBodyControl1(x1: Real, x2: Real, x3: Real): Real = {
2    require (-15 <= x1 && x1 <= 15 && -15 <= x2 && x2 <= 15 && -15 <= x3 && x3 <= 15)
3    -x1*x2 - 2*x2*x3 - x1 - x3
4  } ensuring (res => res <= 705.0 && res +/- 6e-13)
5
```

3. Write the function body on the supported subset of Scalar (needs to be purely functional)

2. Describe precision requirements on the outputs

A Real Type For SDFGs

1. Define input intervals for input arguments

```
1  def rigidBodyControl1(x1: Real, x2: Real, x3: Real): Real = {
2    require (-15 <= x1 && x1 <= 15 && -15 <= x2 && x2 <= 15 && -15 <= x3 && x3 <= 15)
3    -x1*x2 - 2*x2*x3 - x1 - x3
4  } ensuring (res => res <= 705.0 && res +/- 6e-13)
5
```

3. Write the function body on the supported subset of Scalar (needs to be purely functional)

2. Describe precision requirements on the outputs

Rosa returns to required FP-implementation to use (FP32, FP64, FP128)

A `Real` Type For SDFGs: Issues On Proving Error Bounds and Using Functional Language

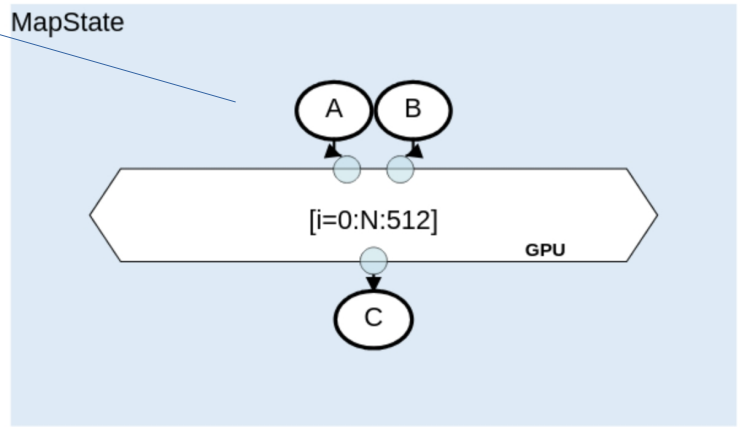
- Rosa (and all similar FP-error-bound analysis tools) uses SAT-solvers to prove the error-bounds – and it does not terminate (or work) for non-toy examples
- (Probably) Nobody uses Scala in HPC
- Proving error bounds is not necessary for many codebases*
  - *GPU enables fast-fp-arithmetic by default and seems like this is very rarely an issue
  - *FP operations are not commutative, no-ordering and thus almost no optimizations would be possible if having no error was that important

- => Plan to replace proving error-bounds with empirical simulation based approach

A `Real` Type For SDFGs:

1. Define Input Intervals and distributions of data

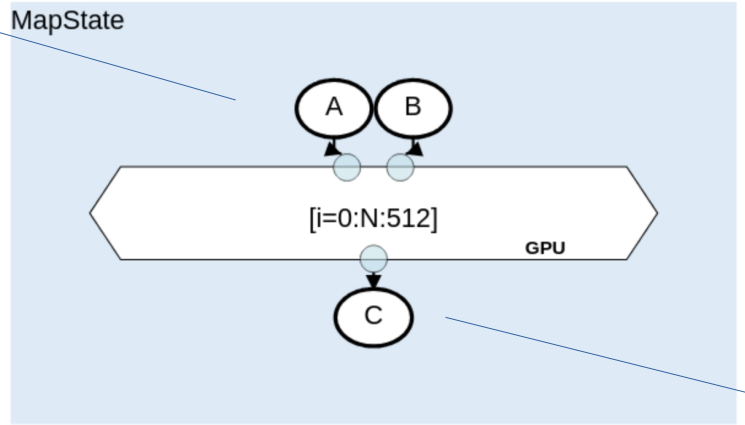
```
sdfg.arrays[“A”].dtype =
dace.Real

input_intervals = {
   “A” = [a_low, a_high],
   “B” = [b_low, b_high]
}

input_distributions = {
  “A” = uniform(low, high),
  “B” = normal(mean, var),
}
```



A `Real` Type For SDFGs:

1. Define Input Intervals and distributions of data

```
sdfg.arrays[“A”].dtype =
dace.Real

input_intervals = {
   “A” = [a_low, a_high],
   “B” = [b_low, b_high]
}

input_distributions = {
  “A” = uniform(low, high),
  “B” = normal(mean, var),
}
```



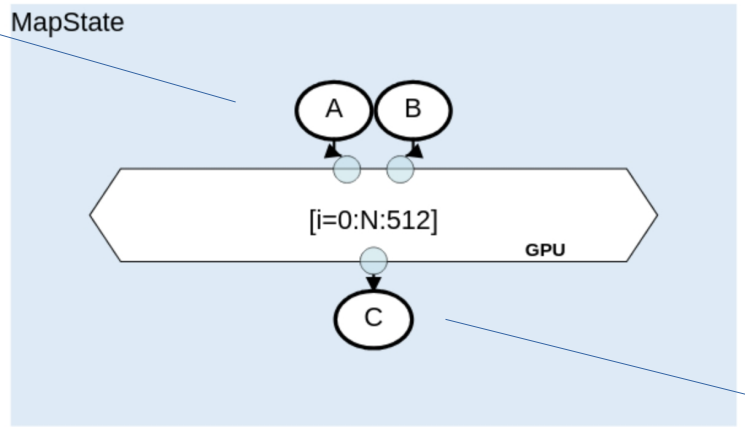
2. Describe Precision Requirements

C.error_bound = 1e-16

A `Real` Type For SDFGs:

1. Define Input Intervals and distributions of data

```
sdfg.arrays[“A”].dtype =
dace.Real

input_intervals = {
   “A” = [a_low, a_high],
   “B” = [b_low, b_high]
}

input_distributions = {
  “A” = uniform(low, high),
  “B” = normal(mean, var),
}
```



3. No restrictions on the SDFG where the pass can be applied

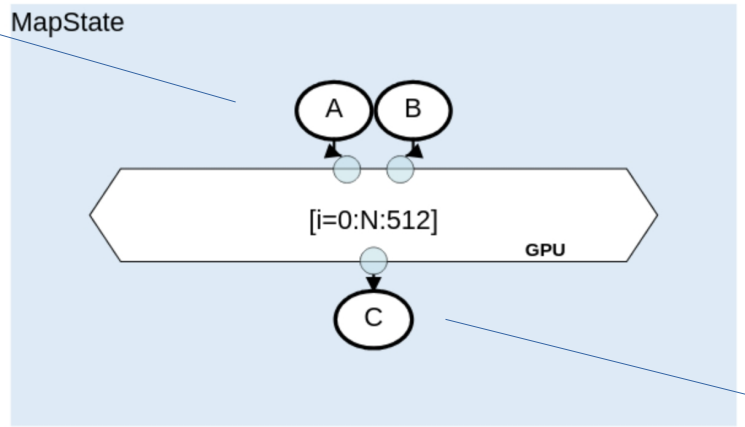
2. Describe Precision Requirements

C.error_bound = 1e-16

A `Real` Type For SDFGs:

1. Define Input Intervals and distributions of data

```
sdfg.arrays[“A”].dtype =
dace.Real

input_intervals = {
    “A” = [a_low, a_high],
    “B” = [b_low, b_high]
}

input_distributions = {
   “A” = uniform(low, high),
   “B” = normal(mean, var),
}
```



3. No restrictions on the SDFG where the pass can be applied

2. Describe Precision Requirements

C.error_bound = 1e-16

DaCe runs emprical simulation and lowers the Real type to a FP implementation

A Real Type For SDFGs

- => Looking for a student to start on this project too ;)

Outcome and TODOs:

- These two weeks I will need to mainly focus on ICON project again.
  - => I can do a presentation on the optimization journey of ICON using DaCe next time
- I have started a major movement to fix DaCe features
  - Codegen part of DaCe needs a rewrite, while also modularizing codegen and distinctly delegating features to passes (Similar to LLVM)
  - Following Features Will Become Passes on SDFGs implemented as part of DaCe:
    - Allocation Scopes → *Default allocation does not work, and could be parallel projects*
    - Kernel Scheduling → *Default schedule does not work, and could be parallel projects*
    - Accelerator-Independent Offloading Pass → *Major migration overhead when writing new backends*
    - Memory Copy Lowering → T*esting multiple implementations is impossible right now*
- Multiple Transformations I am working on:
  - Multiple Buffering
  - [Will work on it in the future] Improvement/Fixes to the BSP-based schedule transformation Aofeng implemented to use data locations instead of Streams
- I am working multiple QoL features and bugfixes for DaCe
  - ~15 Issues Open, ~5 PRs open