Yakup Koray Budanaz

SoftHier Progress Report February 10

Overiew of the Topics:

- Programming Models
- DaCe + SoftHier
- Outlook For Next Weeks

- **AI Accelerators and Programming Models**

AI Accelerators – An Overview of Hardware Features

Accelerator	Cache	Programmable NoC	Explicit Memory Levels	Dedicated MMU	Oversubscription to PEs	Double Precision Suppport
Nvidia A100	✓	×	2	✓	✓	✓
Nvidia H100	✓	✓*	2	✓	✓	✓
Ascend 910B	×	×	3	✓	×	×
Graphcore IPU Mk2	×	✓	2	×	✓	×
AMD VC2802**	×	✓	2	×	×	×
Microsoft Maia 100***	×	?	?	✓	?	×
Cerebras WSE-2	×	✓	2	×	×	×
Intel Gaudi v3	✓	×	2	✓	✓	×
AWS Trainium2	×	×	2	✓	×	×

Information collected from available white-papers, official tutorials and HotChips presentations.

AI Accelerators

- TPU is not there because I found nothing on the low-level programming of TPUs.
- Only see how to write in Hight Level Operator (HLO) IR for Accelerated Linear Algebra (XLA) DL compiler used as backend for JAX and TensorFlow.

Programming Models: High-Level IRs:

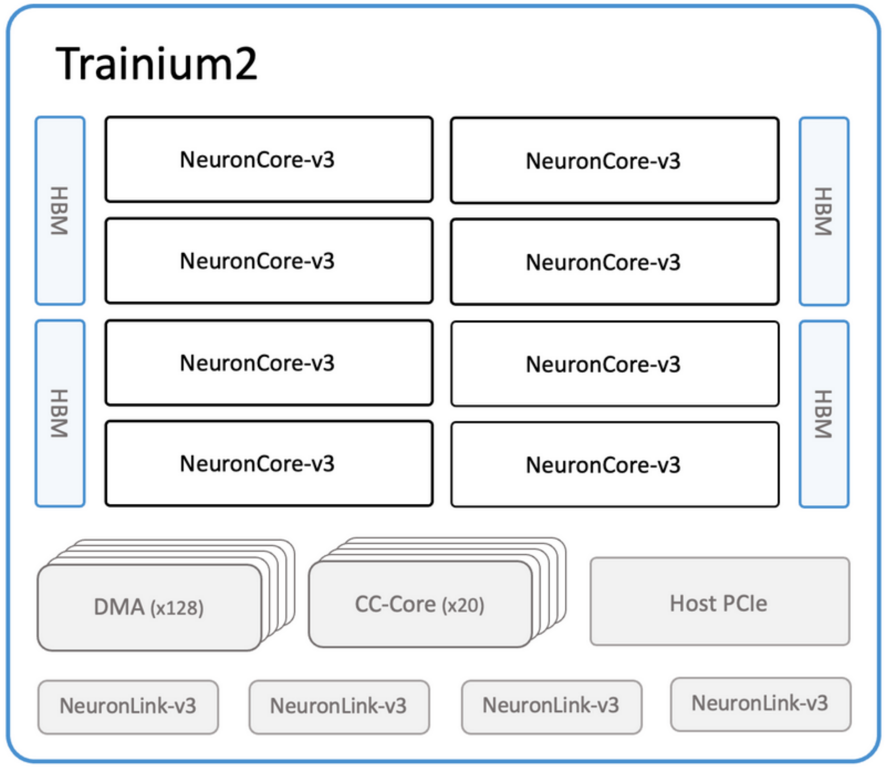
- A data-flow graph consisting of operations and input/outputs.
- Not really different a graph of consisting of ONNX operators. It is a variant of MLIR. Looks like this:

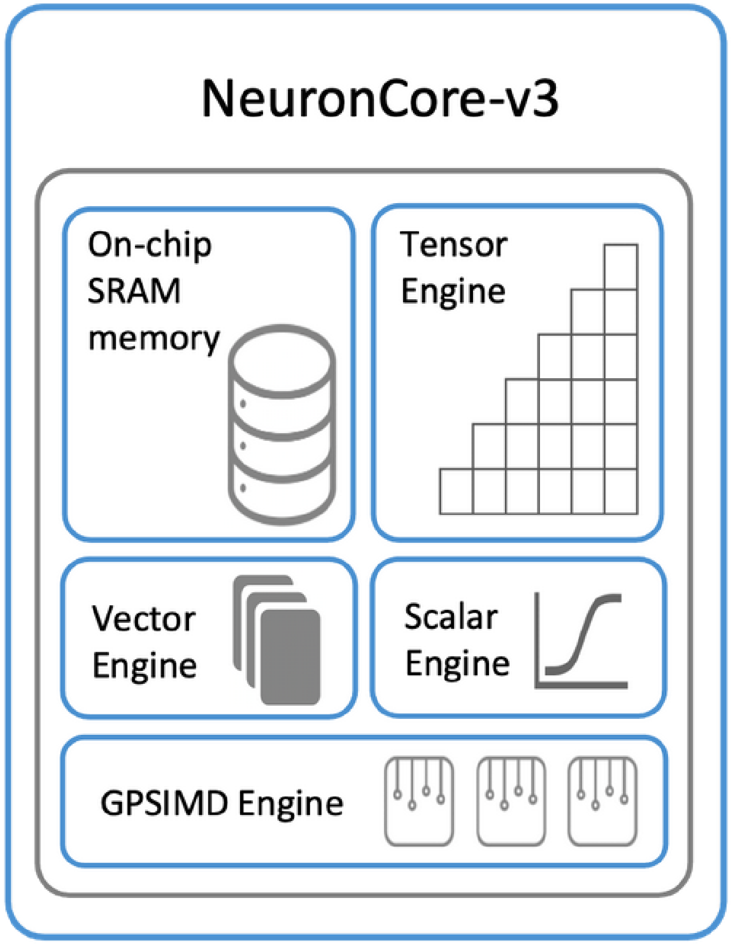
```
func @add_tensors(%arg0: tensor, %arg1: tensor) -> tensor {
  %0 = "mhlo.add"(%arg0, %arg1) : (tensor, tensor) -> tensor
  return %0 : tensor
}
```

Abstracting Away Complexity – Method 1: Provide a list of Operations on Tensors + Efficient Implementations for them

Google TPU only supports this, most AI accelerators support this

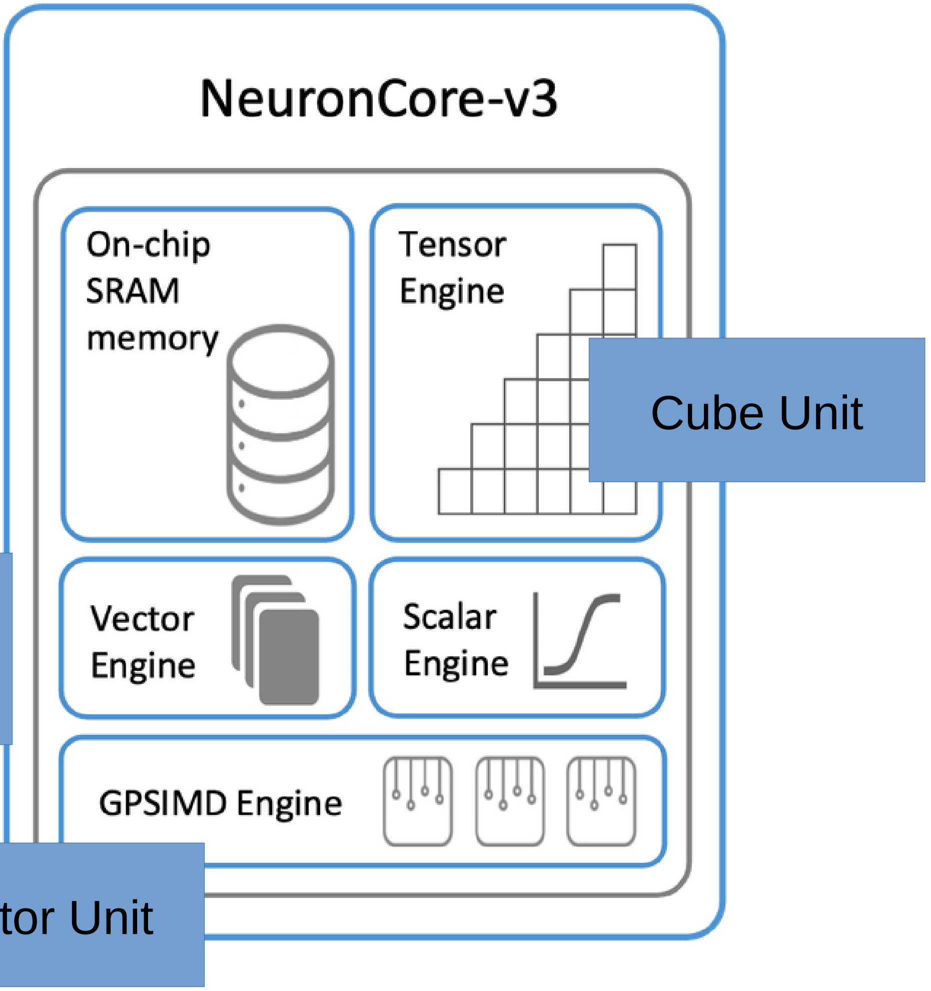
A very similar looking device - AWS Trainium2





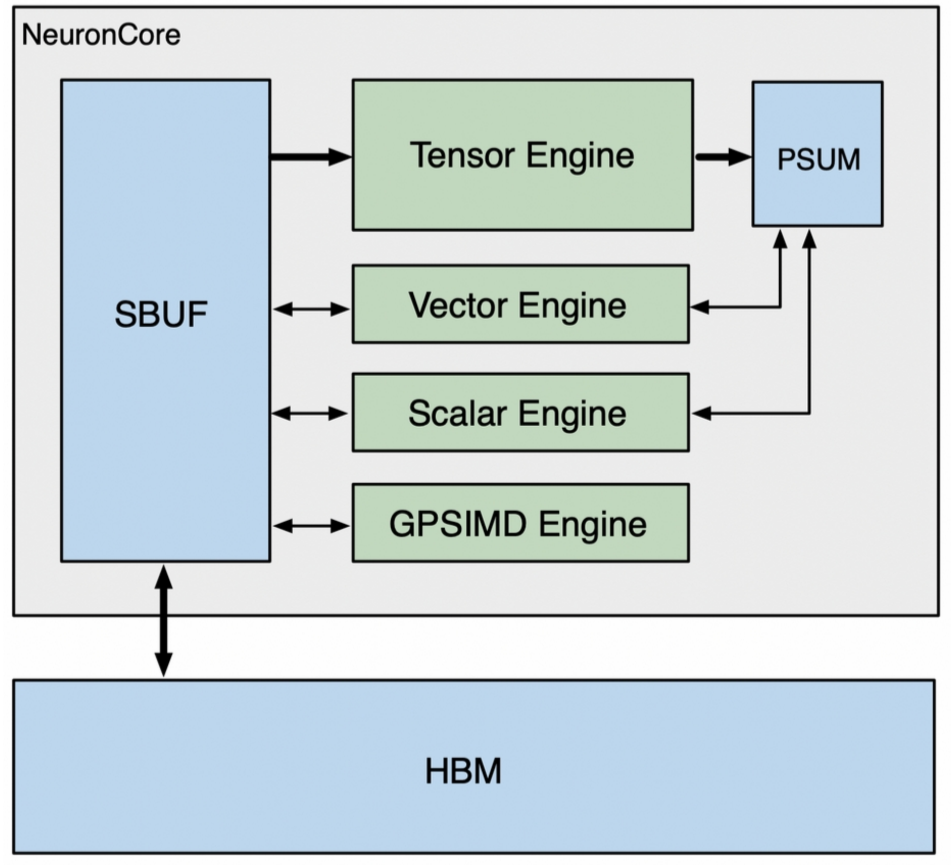
https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/arch/neuron-hardware/trainium2.html

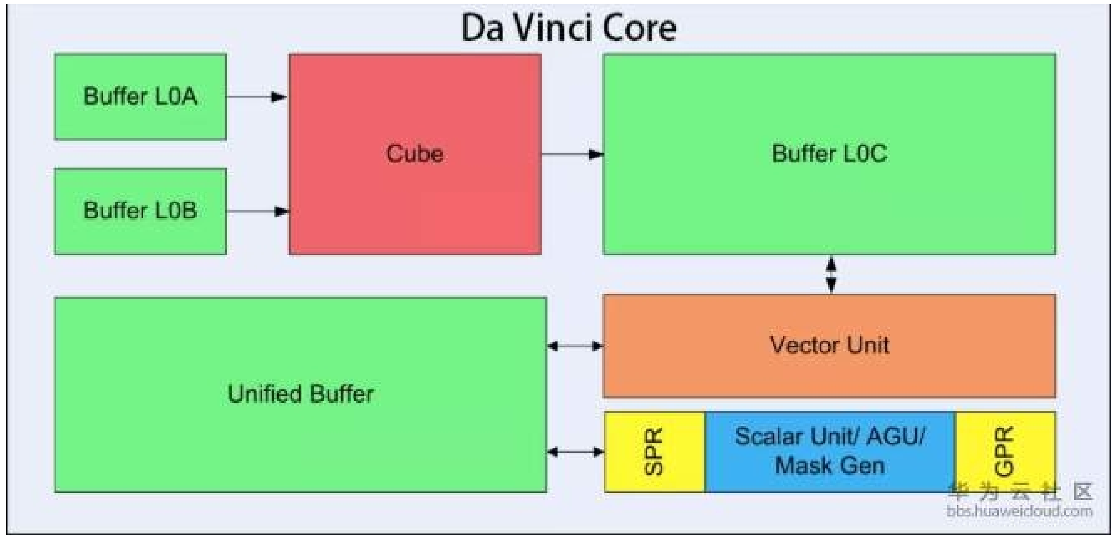
A very similar looking device - AWS Trainium2



Vector Unit for special ops (e.g., axpi)

A very similar looking device - AWS Trainium2





https://awsdocs-neuron.readthedocs-hosted.com/en/latest/general/nki/programming_model.html#nki-programming-model

https://bbs.huaweicloud.com/blogs/106229

A very similar looking device - AWS Trainium2

- Provides a tile-based programming interface similar to Triton named NKI
- No low-level programming interface for other computational units.

Abstracting Away Complexity – Method 1: Provide a list of Operations on Tensors + Efficient Implementations for them

Programming Models

- Betting on providing efficient implementations for specific tensor operations, or betting on tile-based languages.

Triton

- Load tiles and describe computation on tiles.
- Hopefully Triton back end + transformations will come-up with the most efficient way to move the memory

Abstracting Away Complexity – Method 2: Tile-Based language + a lot of memory movement optimizations

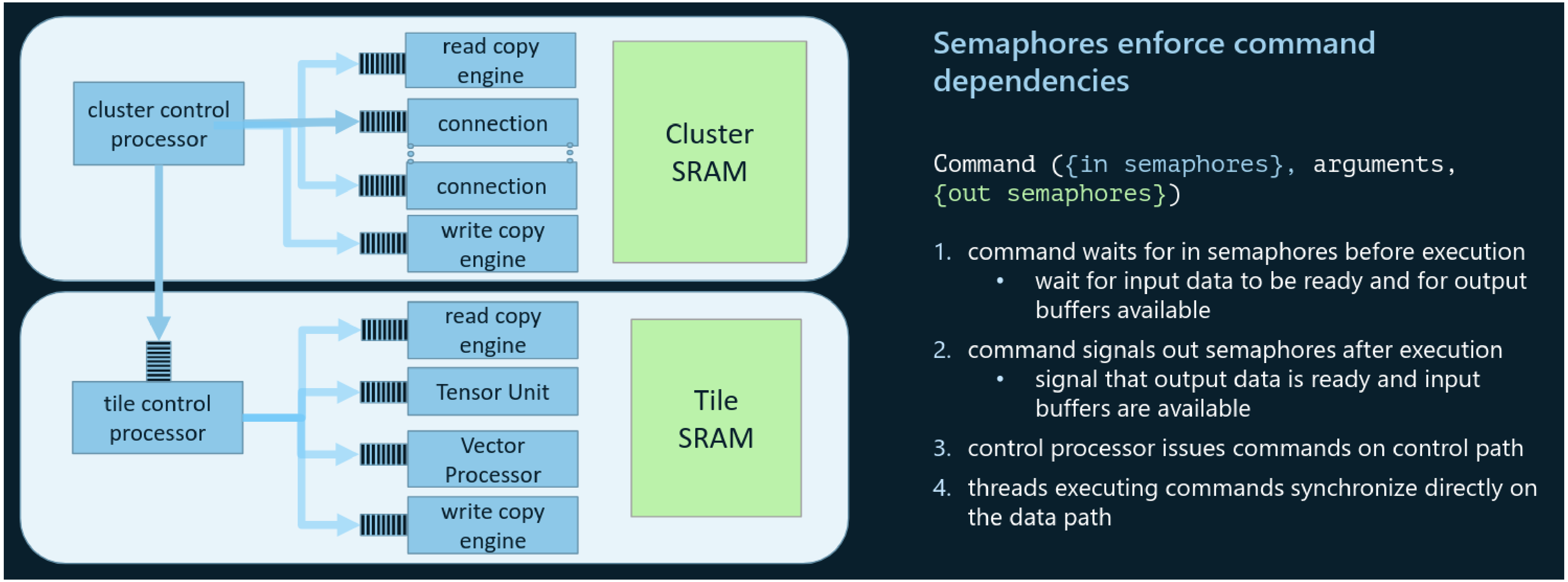
Triton

- Tiled-based CUDA Wrapper (+embedded DSL in Python to generate Triton IR and into LLVM IR to ptx)
- Example stencil kernel.

```python
A_bottom = triton.language.load(
    A_ptr + (i + 2) * N + (j + 1), mask=(i < N - 2) & (j < N - 2), other=0.0
)

...
B_new = 0.20 * (A_left + A_right + A_top + A_bottom + A_center)
triton.language.store(
    B_ptr + (i + 1) * N + (j + 1), B_new, mask=(i < N - 2) & (j < N - 2)
)

```

Microsoft Maia 100

- From the hotchips presentation. Looks like Maia will have a model similar to asynchronous tasks, dependencies will be satisfied using semaphores.



Semaphores enforce command dependencies

```
Command ({in semaphores}, arguments,
{out semaphores})
```

1. command waits for in semaphores before execution
   - wait for input data to be ready and for output buffers available
2. command signals out semaphores after execution
   - signal that output data is ready and input buffers are available
3. control processor issues commands on control path
4. threads executing commands synchronize directly on the data path

https://hc2024.hotchips.org/assets/program/conference/day2/81_HC2024.Microsoft.Xu.Ramakrishnan.final.v2.pdf

Microsoft Maia 100

- It also plans to provide Triton as a programming model

Abstracting Away Complexity – Method 3: Asynchronous Tasks Based Language?

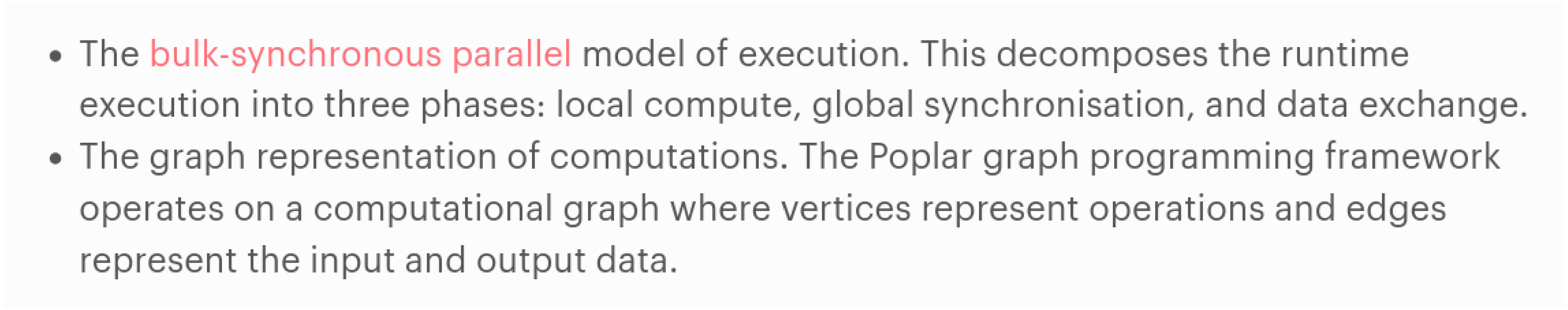
Intel Gaudi 3

- It provides TPC language for low-level programming of the device
- Almost identical to CUDA with more intrinsics. It exposes warp (vector processor) more than the thread.

Abstracting Away Complexity – Method 4: Copy CUDA

Graphcore:

- Provide poplar C++ API to create a dataflow graph consisting of tensor operations and input-output depdencies.
- Looks like SDFG with library-nodes only.
- The operation implementation uses a BSP-like model, the communication is implicitly generated using tiles. (It describes how computation is mapped to cores using a tile abstraction)
- Inter-PE communication is facilitated by the compiler (derived from tiling in form of graph partitioning).
- For more explicit control they provide assembly for their ISA.



- The bulk-synchronous parallel model of execution. This decomposes the runtime execution into three phases: local compute, global synchronisation, and data exchange.
- The graph representation of computations. The Poplar graph programming framework operates on a computational graph where vertices represent operations and edges represent the input and output data.

Graphcore

Abstracting Away Complexity – Method 5: Graph-based language + Graph-partitioning

AMD FPGAs

- Super low-level
- Super chaotic report
- Looks like assembly with some wrappers
- Inter-PE communication through an API similar to streams
- Provides a dataflow graph abstraction for operators

Abstracting Away Complexity – Method 6: Just Don’t?

Asynchronous Memory Pipelines

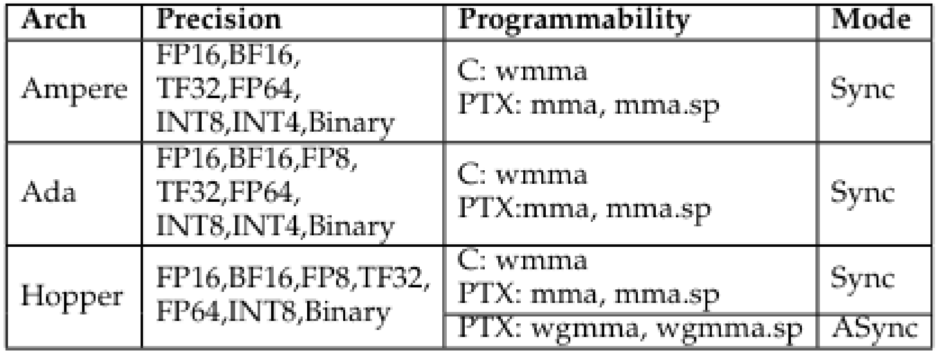
- Pipeline as a FIFO Queue following producer-consumer pattern (asynchronous).

```
  cuda::pipeline<cuda::thread_scope_thread> pipe = cuda::make_pipeline();

  cuda::memcpy_async(&smem[stage][threadIdx.x], &src[idx], sizeof(int), pipe);
    pipe.producer_commit();

  for (...) {
    cuda::pipeline_consumer_wait_prior<num_stages - 1>(pipe);
    __syncthreads();

    compute();

    __syncthreads();
    pipe.consumer_release();

    pipe.producer_acquire();
    cuda::memcpy_async(&smem[stage][threadIdx.x], &src[idx], sizeof(int), pipe);

    pipe.producer_commit();

    stage = (stage + 1) % num_stages;
  }
}
```

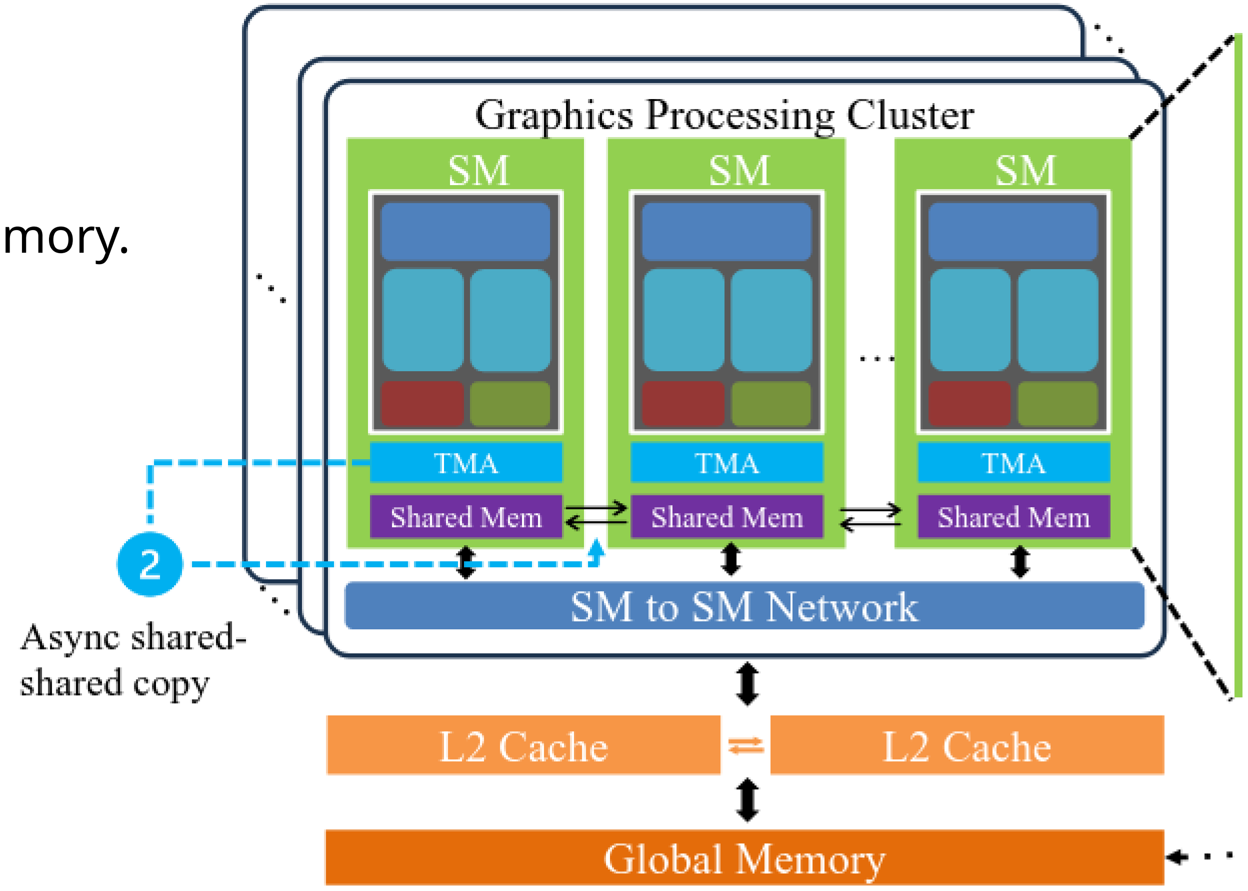
https://developer.nvidia.com/blog/controlling-data-movement-to-boost-performance-on-ampere-architecture/

Asynchronous DMA Engine:

- Tensor cores are now asynchronous too.
- One TMA per SM to asynchronously move memory.



Arch	Precision	Programmability	Mode
Ampere	FP16,BF16, TF32,FP64, INT8,INT4,Binary	C: wmma PTX: mma, mma.sp	Sync
Ada	FP16,BF16,FP8, TF32,FP64, INT8,INT4,Binary	C: wmma PTX:mma, mma.sp	Sync
Hopper	FP16,BF16,FP8,TF32, FP64,INT8,Binary	C: wmma PTX: mma, mma.sp	Sync
		PTX: wgmma, wgmma.sp	ASync



https://arxiv.org/pdf/2501.12084

Asynchronous DMA Engine:

- CuTe / CUTLASS provides convenient ptx wrappers for common optimizations such as memory swizzling, and calls to tensor cores / TMA.

Abstracting Away Complexity – Method 7: Providing Assembly Wrappers for Common Use-cases

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html?highlight=TMA#using-tma-to-transfer-one-dimensional-arrays

Asynchronous DMA Engine:

- TMA without high-level wrappers and only with ptx wrappers:
- Create, and copy tensor map
- Create many barriers, semaphores
- Async copies, set barriers, wait on barriers
- …

Abstracting Away Complexity – Method 7: Even more convenient(?) assembly wrappers

https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html?highlight=TMA#using-tma-to-transfer-one-dimensional-arrays

My Take on Programming Models:

- If I were to design a programming language I would only treat asynchronous tasks (defined over tiles) with input and output dependencies and construct a graph of asynchronous tasks.
- Graph IRs are technically asynchronous. The dependencies are expressed as edges. The compiler or transformations need to find optimal memory movement pattern.

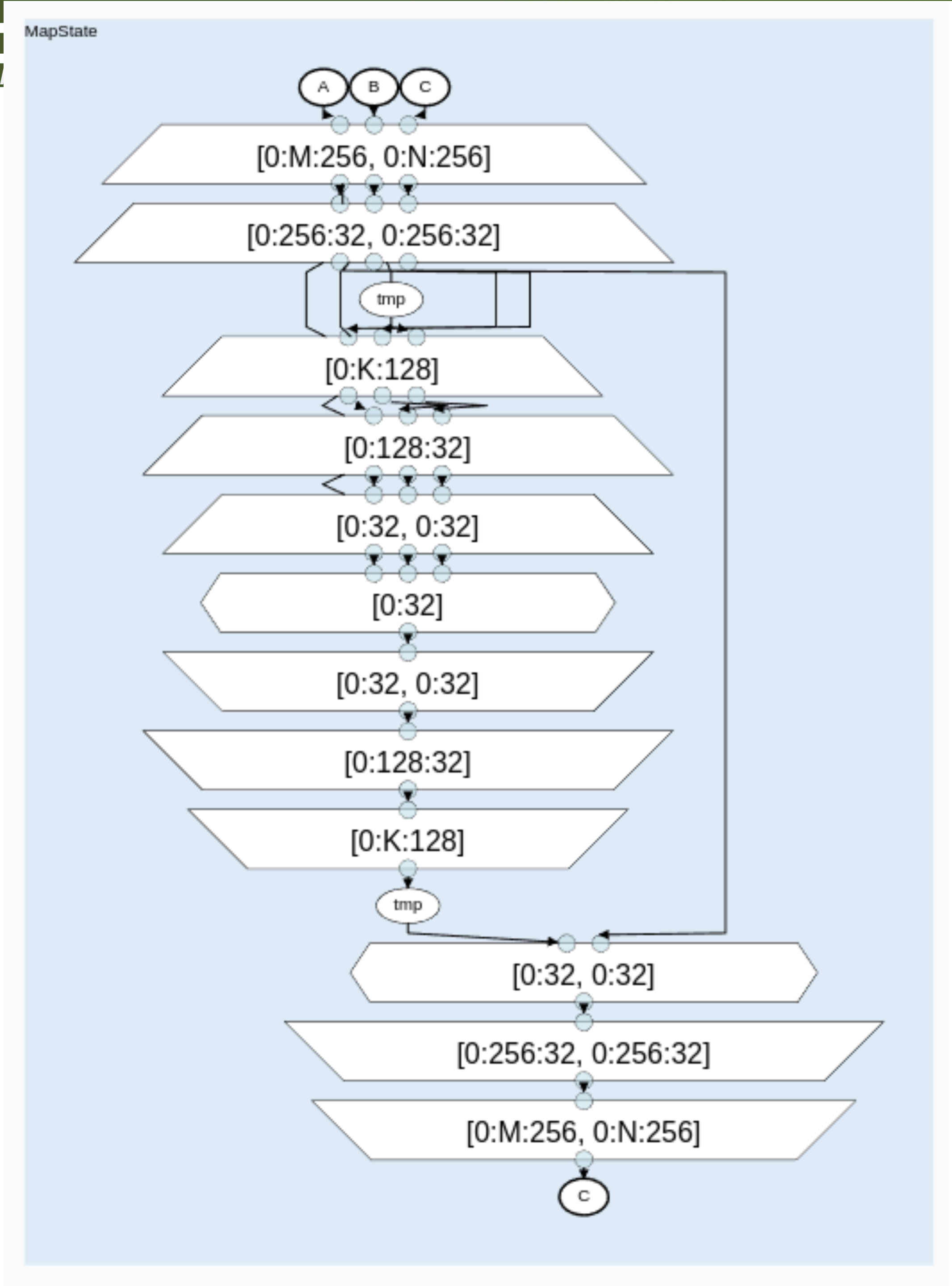
- **DaCe + SoftHier**

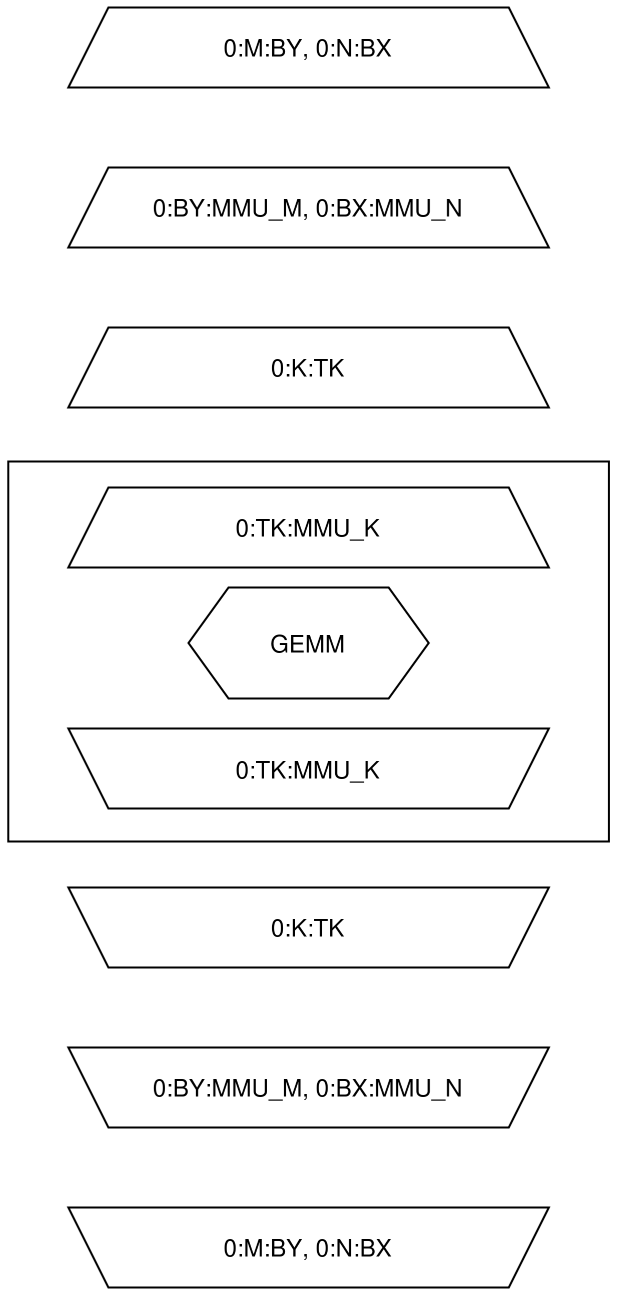
Expressing NoCs in SDFGs:

- In my opinion, the best way of expressing NoC communication is using streams.

Expressing NoCs in SDFGs:

- In my opinion, the best way of expressing NoC communication is using streams.
- Stream abstraction used in SDFGs is a FIFO queue with multiple senders and one receiver.
- How to transform SDFGs?
  - Designed a transformation that transforms a subgraph to use streams using a BSP-model-like description



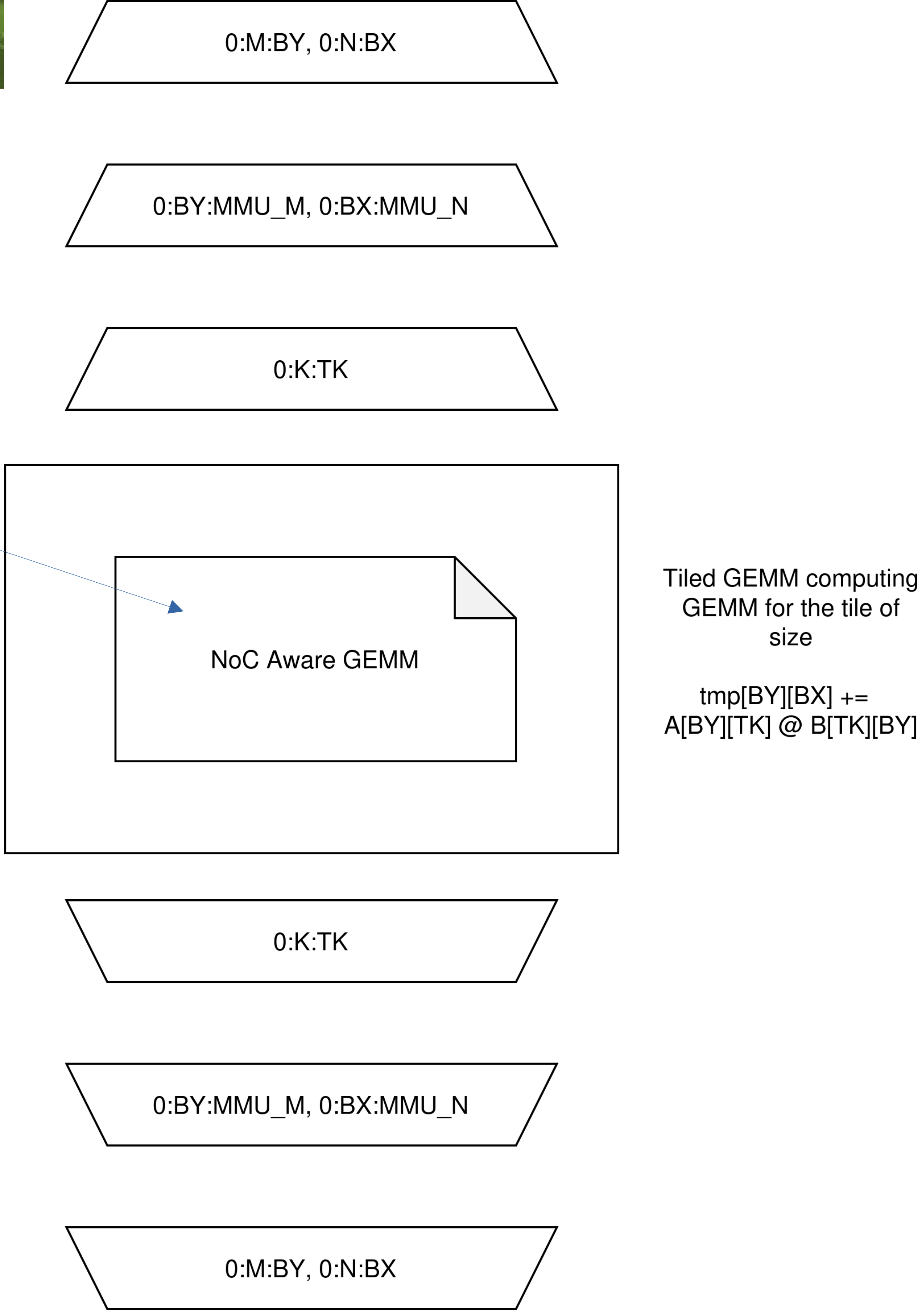


Tiled GEMM computing GEMM for the tile of size

tmp[BY][BX] += A[BY][TK] @ B[TK][BY]

Input is a BSP-model based description of the algorithm running inside.

1. Initial data dist (which core owns data)
2. Communication scheme between cores
3. Synchronization (if necessary in the architecture)



1. Initial distribution:
sA and sB are 2 dimensions streams, both of size [N][N], N=NX=NY

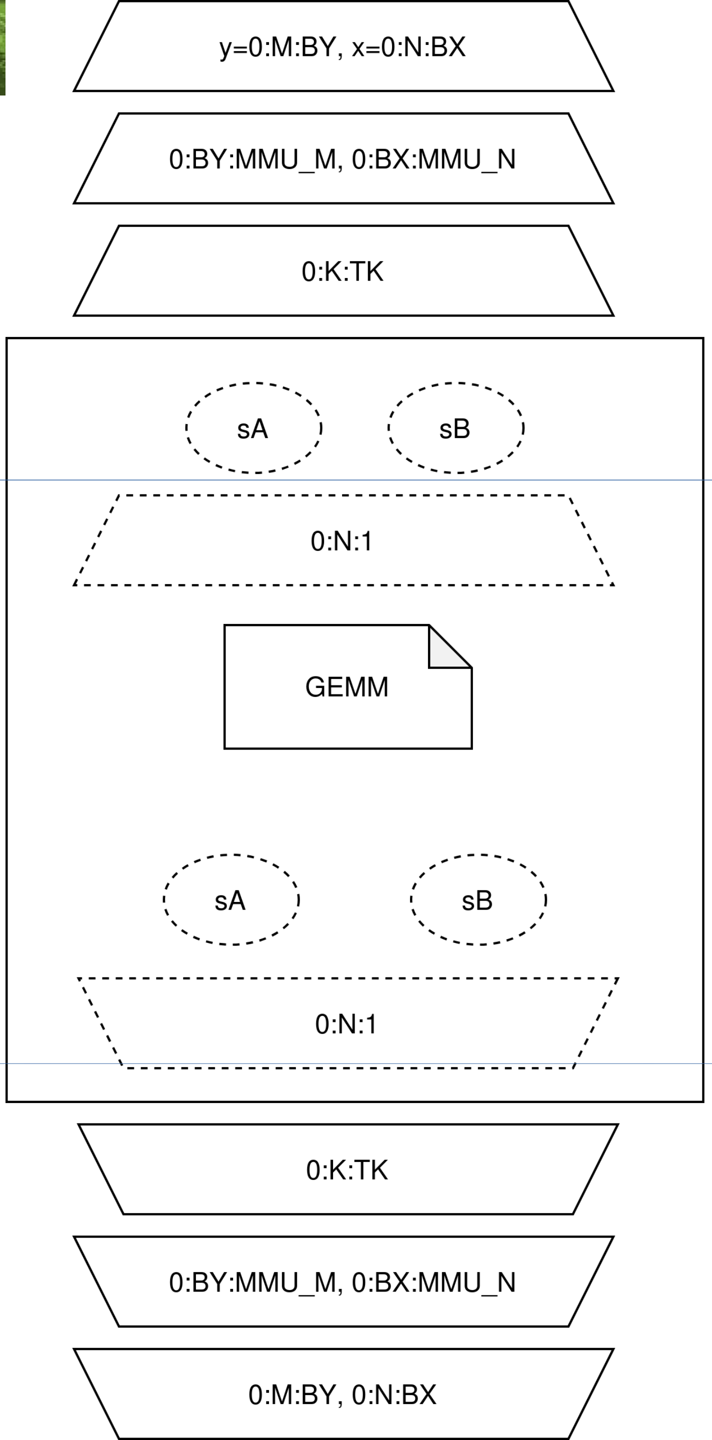
A[i * tileSize][((i+j) % N) * tileSize] → sA[i][(i+j)%N][:][:]
B[((i+j) % N) * tileSize][j * tileSize] → sB[(i+j)%N][j][:][:]

2. Compute:
localA = sA.pop(), localB = sB.pop()
tmp += localA * localB

3. Communication

LocalA[:][:] →sA[i][(j+N-1)%N][:][:]
LocalB[:][:] →sB[(i+N-1)%N][j][:][:]

4. Synchronization
Yes|No depending on the architecture



Assume
NumCoreY == NumCoreX
and BY == TK == BX

Tiled GEMM computing GEMM for the tile of size

tmp[BY][BX] += A[BY][TK] @ B[TK][BY]

No issues in part 1

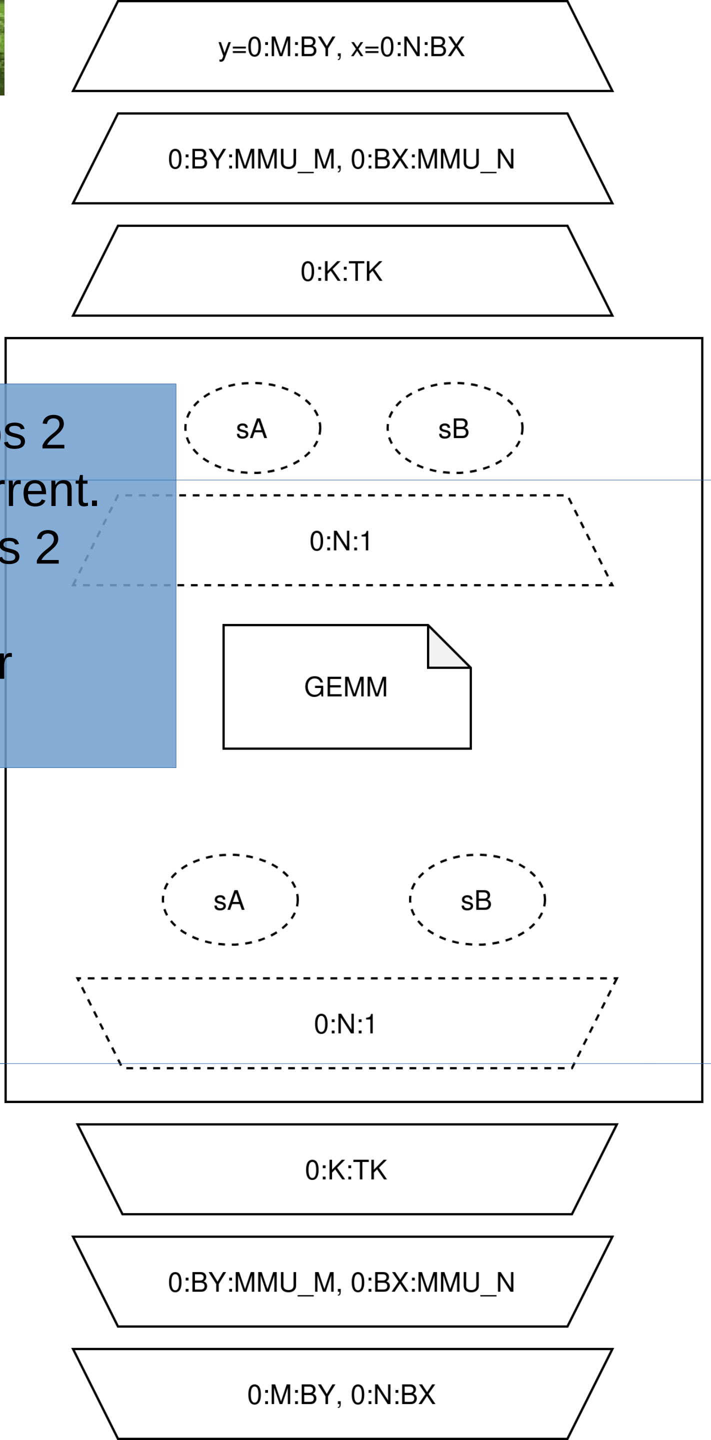
2. Compute:
localA = sA.pop(), localB = sB.pop()
tmp += localA * localB

3. Communication

LocalA[:][:] →sA[i][(j+N-1)%N][:][:]
LocalB[:][:] →sB[(i+N-1)%N][j][:][:]

4. Synchronization
Yes|No depending on the architecture

For GEMM, the steps 2 and 3 can be concurrent. For Stencil, the steps 2 and 3 can't be concurrent. The user



Assume
NumCoreY == NumCoreX
and BY == TK == BX

Tiled GEMM computing GEMM for the tile of size

tmp[BY][BX] +=
A[BY][TK] @ B[TK][BY]

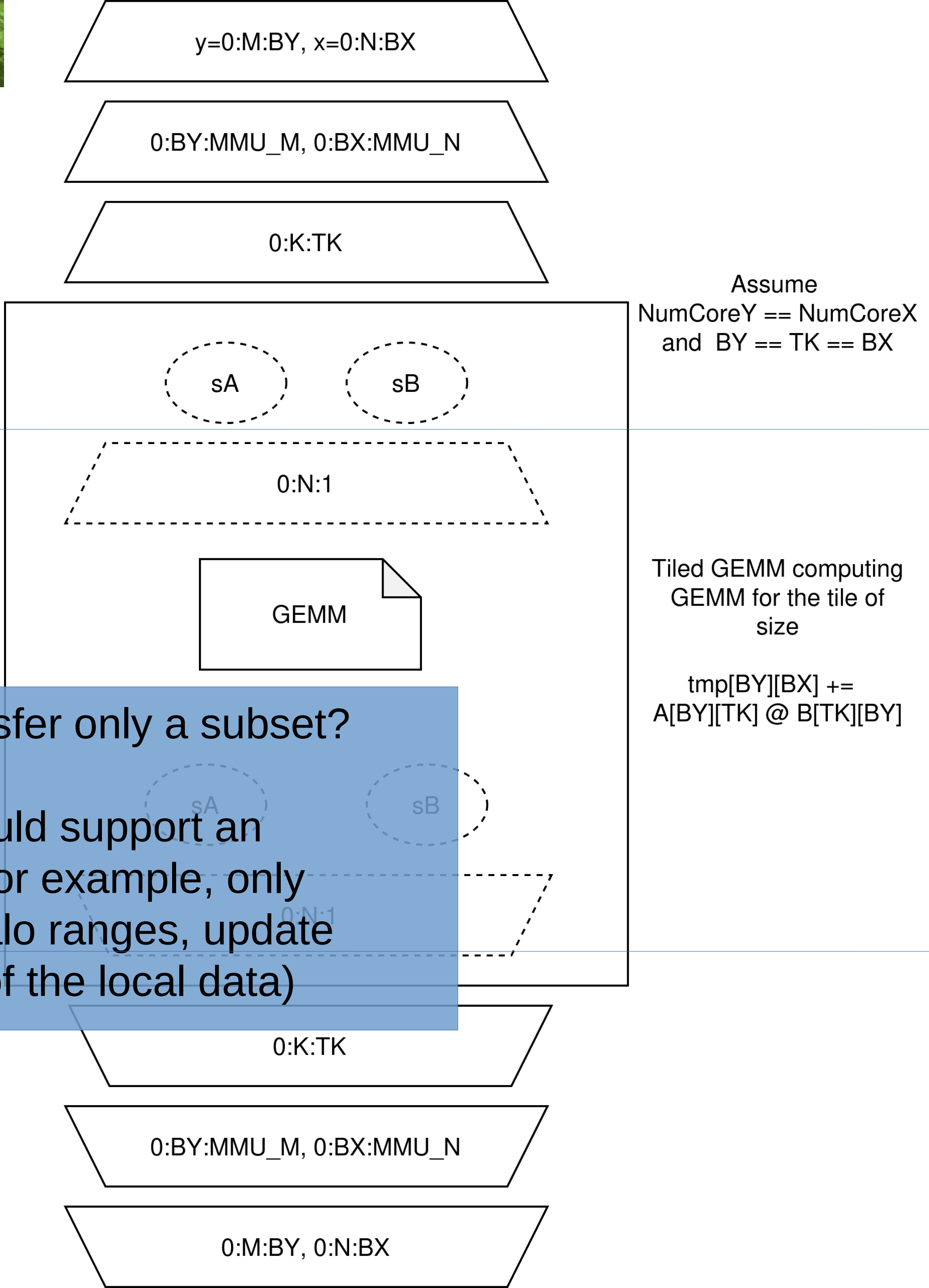
No issues in part 1

2. Compute:
localA = sA.pop(), localB = sB.pop()
tmp += localA * localB

3. Communication

LocalA[:][:] →sA[i][(j+N-1)%N][:][:]
LocalB[:][:] →sB[(i+N-1)%N][j][:][:]

No issues in part 4



Assume NumCoreY == NumCoreX and BY == TK == BX

Tiled GEMM computing GEMM for the tile of size

tmp[BY][BX] += A[BY][TK] @ B[TK][BY]

What if need to transfer only a subset?

The description should support an update. (In stencil, for example, only communicate the halo ranges, update the local boundary of the local data)

1. Initial distribution:
sA and sB are 2 dimensions streams, both of size [N][N], N=NX=NY,
A[i * tileSize][((i+j) % N) * tileSize] → sA[i][(i+j)%N][:][:]
B[((i+j) % N) * tileSize][j * tileSize] → sB[(i+j)%N][j][:][:]

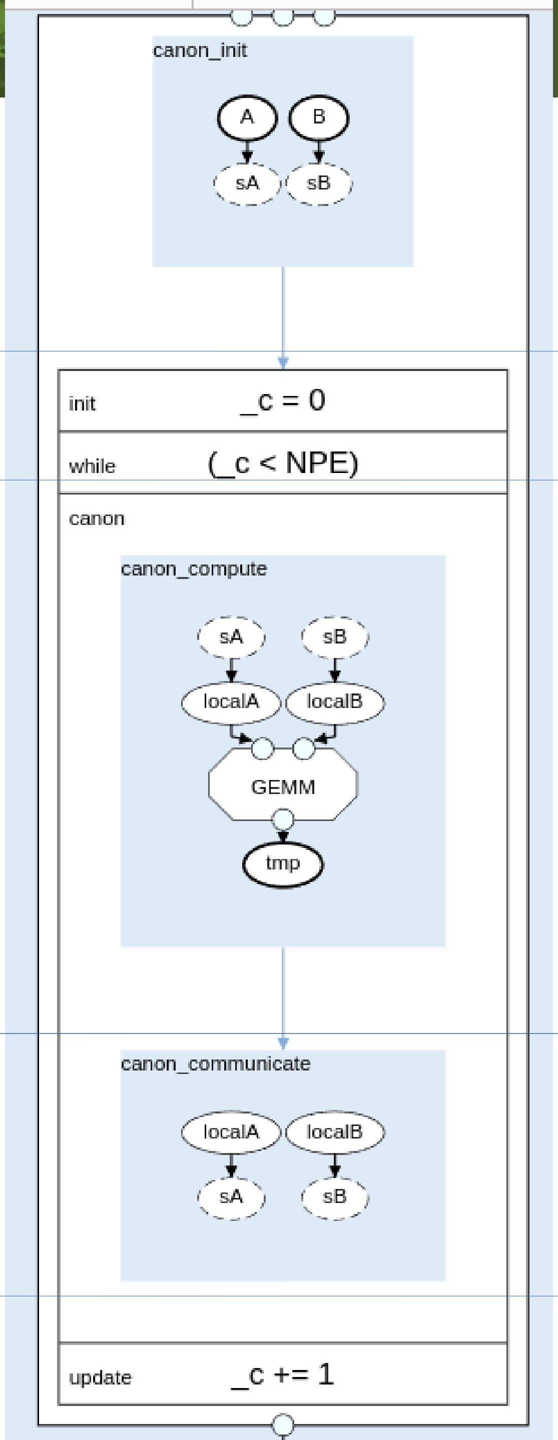
2. Loop Range

2. Compute:
localA = sA.pop(), localB = sB.pop()
tmp += localA * localB

3. Communication

LocalA[:][:] →sA[i][(j+N-1)%N][:][:]
LocalB[:][:] →sB[(i+N-1)%N][j][:][:]

4. Synchronization
Generated as a library node on a 3rd state or as a loop property



- Scalar → MMU Pass (Pattern detection + Reading Tasklet ASTs), working for couple of patterns
- Scalar → Vector Unit Pass (Pattern detection + Reading Tasklet ASTs), works for couple of patterns, need to extend patterns later as necessary
- GEMM Input Type information is added to access nodes which is used by the controller to apply the correct ExplicitMemoryMove transformation
- Passing “input purpose” to SDFGs using access nodes manually (done)
- Automating this information by extending Scalar → MMU Pass (mostly done)
- Pass to insert memory movement between different computational units (under work)

Outlook / TODOs:

- Work further on transformations
- Extend cube unit parts of the code-gen

My Take on Programming Models:

Sketched how it could look like. Many considerations are necessary. Message buffers, supporting out-of-order messages, low-level implementation of completions of one-sided communication.

```
__kernel__ vadd<N>(in A[N*1024], in B[N*1024], inout C[N*1024]){
    decl id = PEs<N>
    decl TS = 1024
    a1 = reserve(localB[1024], TCM)
    a2 = reserve(localA[1024], TCM)
    a3 = reserve(localC[1024], TCM)
    l1 = a1.then(load(in:A[i*TS:(i+1)*TS], out:localA[:]))
    l2 = a2.then(load(in:B[i*TS:(i+1)*TS], out:localB[:]))
    t1 = join(l1, l2).then(add(in:localA[:], in:localB[:], out:localC[:]))
    s1, r1 = t1.then(send(to:(i+N-1)%N, in:localC[:], out:localC[:]))
    c1 = r1.then(store(in:localC[:], out:C[i*TS:(i+1)*TS]))
    return c1
}

...

vadd_kernel = vadd<N>(A, B, C)
vadd_kernel.wait()
```