

PyTorch

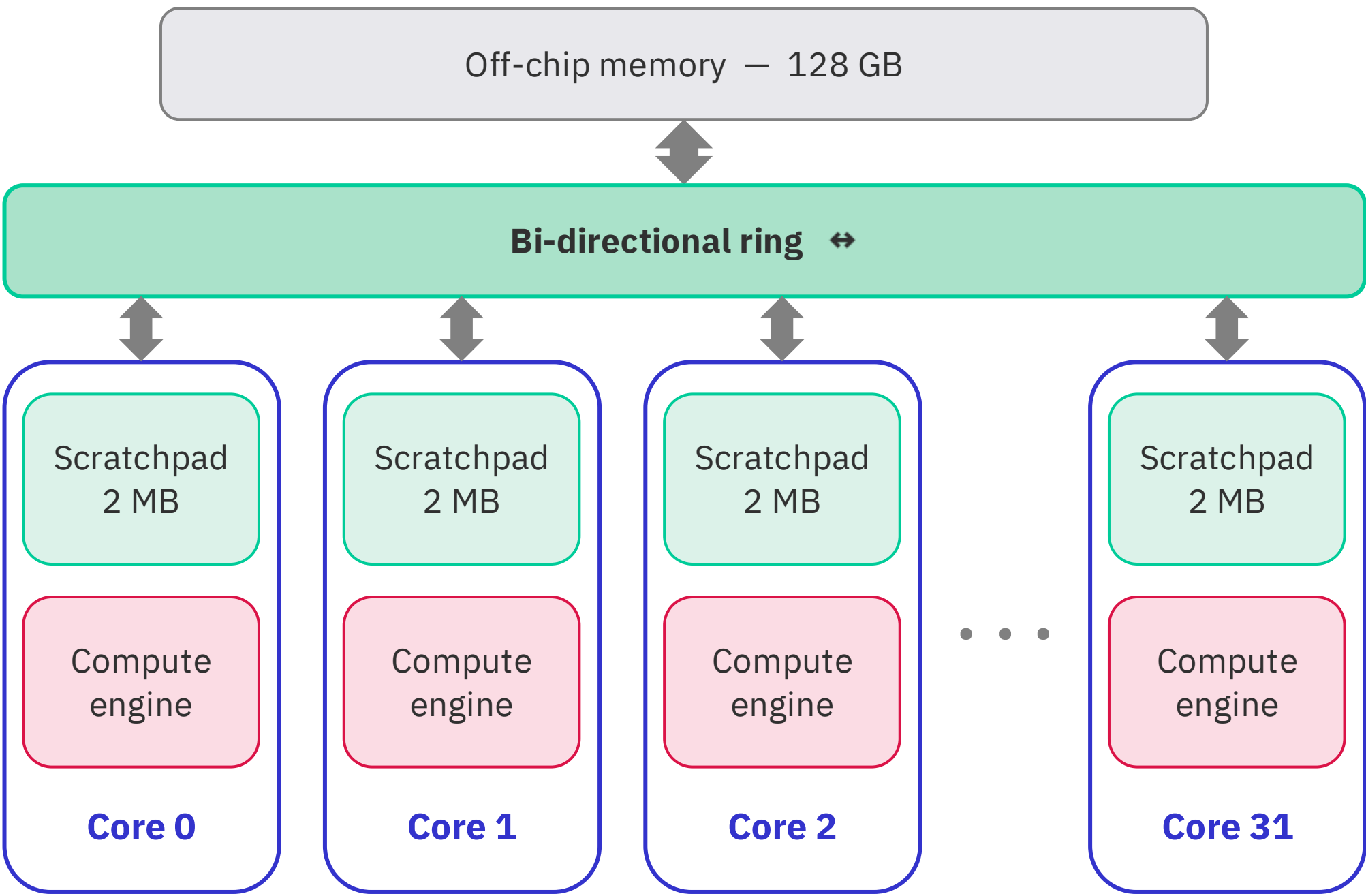
CONFERENCE

—NORTH AMERICA 2026—

Beyond SPMD: Compiling Triton
Kernels for Dataflow AI Architectures

Takuya Nakaike, Yu Chin Fabian Lim, Linsong Chu, Eri Ogawa, Moriyoshi Ohara, Mudhakar Srivatsa · IBM Research

Challenge: Triton extensions to abstract dataflow AI architectures



Spyre, abstracted: 32 AI cores on a bi-directional ring, each with its own private 2 MB scratchpad, over off-chip memory. A scratchpad is accessed in 128 B units, called sticks. A tile lives in a scratchpad, so moving one to another core crosses the ring. [ISSCC 2026, 2.6]

Goal: Rearrange tile layouts within core-local scratchpads, without going through device memory.

- Device-memory bandwidth is the bottleneck of AI accelerators.

Our abstraction: distributed tiles

- Hidden: how dataflow is scheduled.** Dataflow scheduling should be optimized by the backend for a specific hardware instance.
- Exposed: how data is distributed as tiles.** Across memory spaces — device memory **or** core scratchpads that form one **global view**.

Our extensions in Triton

- The memory space for each tile**, such as an on-chip core-local scratchpad or an off-chip device-memory slice.
- A global view of a composite tile** to allow affine indexing. Any core reads any block of it, not just its own memory.

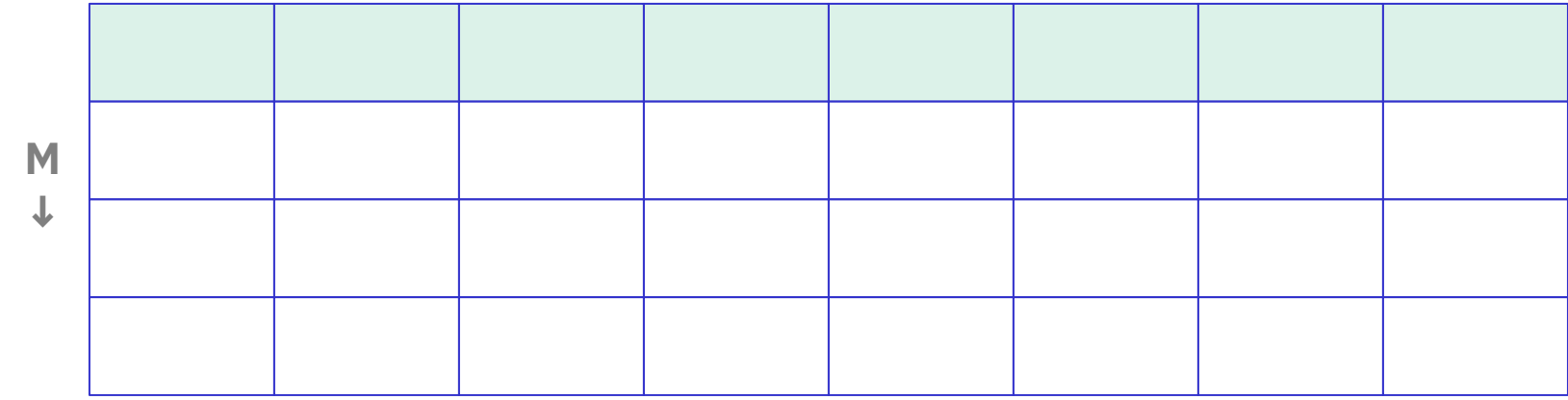
Meta's TLX names cores but has no global view; Gluon has the global view but names no core. **Nobody has both** — and any dataflow accelerator can use the pair.

Triton extensions: memory spaces and a global view of tiles

(a + b) @ w on 32 cores a, b: [M, K] = [64, 512] w: [K, N] = [512, 256] out: [M, N] = [64, 256]

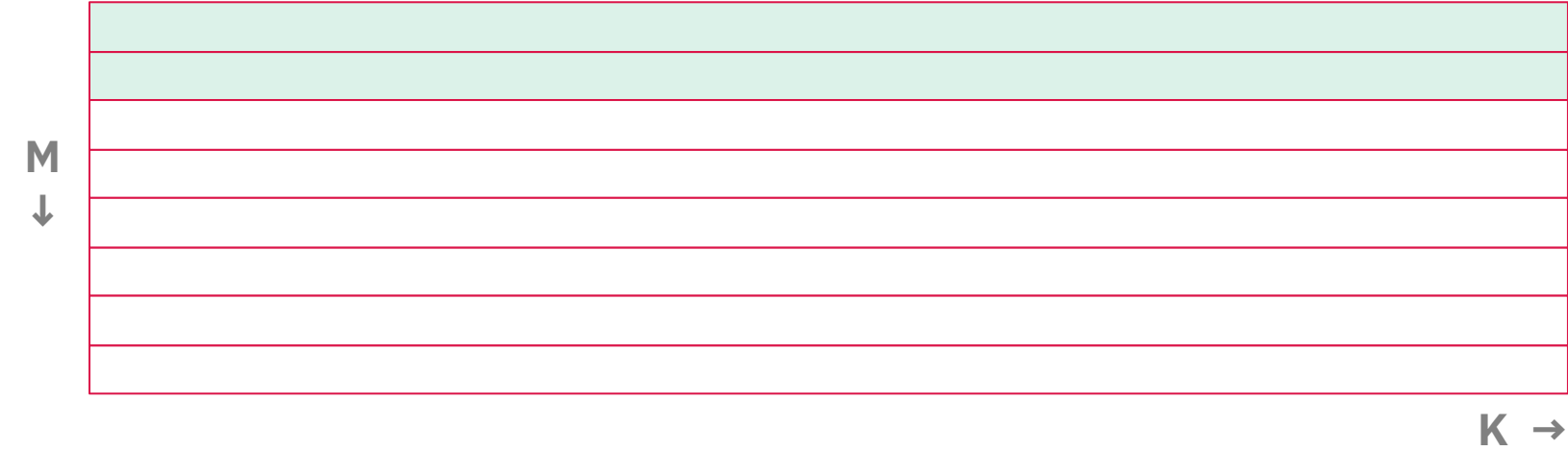
One descriptor composes every tile — no collective primitive added.

add: {M: 4, K: 8} → 32 tiles of [16, 64]



K: 8 tiles → 1
M: 1 tile → 2

matmul: {M: 8, K: 1} → 8 tiles of [8, 512], replicated by 4



```
(a + b) @ w, the matmul's {M: 8, N: 4} re-divides M: [16,64] → [8,512] @ [512,64]

WORK_SLICES = [{ "M":0, "K":0}, {"M":0, "K":1}, ...] # add: {M: 4, K: 8}
MM_SLICES   = [{ "M":0, "N":0}, {"M":0, "N":1}, ...] # matmul: {M: 8, N: 4}
m = tl.wk_slice_coord(WORK_SLICES, "M") * 16
k = tl.wk_slice_coord(WORK_SLICES, "K") * 64
tmp2 = a.load([m, k]) + b.load([m, k])
tl.spyre_pin(tmp2, "ct_local", offset=0) # Specify a memory space

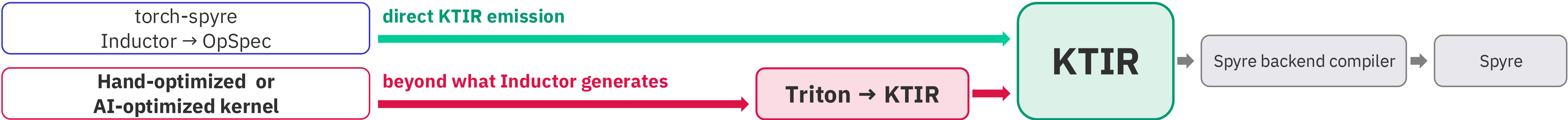
whole = tl.make_distributed_descriptor( # Create a global view
    tmp2, WORK_SLICES, ["M", "K"], [8, 8*64])
mine = whole.load([tl.wk_slice_coord(MM_SLICES, "M") * 8, 0])

tl.spyre_pin(mine, "ct_local", offset=1024) # Specify a memory space
out = tl.dot(mine, w.load([0, tl.wk_slice_coord(MM_SLICES, "N") * 64]))

(tl.wk_slice_coord(TABLE, "M")): this program's coordinate along the M
dimension in TABLE.)
```

Our implementation: compiling Triton kernels through KTIR

KTIR is a tile-based, block-structured MLIR IR for multi-core accelerators, embodying a data-parallel abstraction of the hardware.



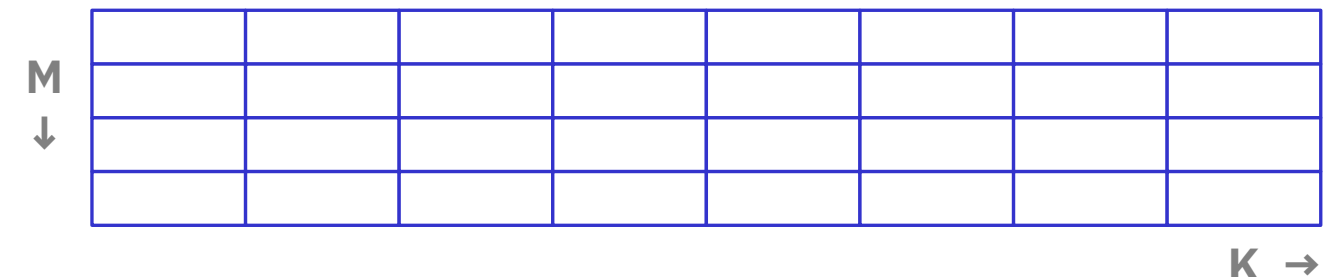
Triton

Represent the tensor tiling by WORK_SLICES.

```
WORK_SLICES = [{ "M":0, "K":0}, {"M":0, "K":1}, ...]

whole = tl.make_distributed_descriptor(
    tmp2, WORK_SLICES, ["M", "K"], [8, 8*64])

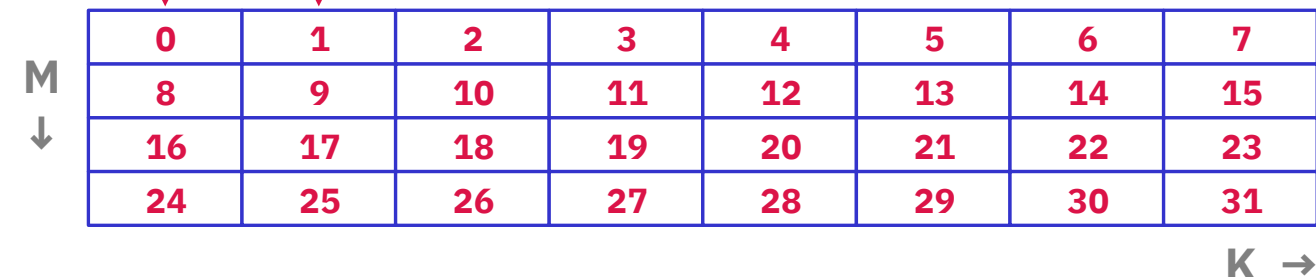
MM_SLICES = [{ "M":0, "N":0}, {"M":0, "N":1}, ...]
mine = whole.load(
    [tl.wk_slice_coord(MM_SLICES, "M") * 8, 0])
```



KTIR

Represent which core holds each tile by ct_id.

```
%28 = ktdp.construct_memory_view %c0, sizes: [16, 64]
      : memref<16x64xf16, #ktdp.memory_space<ct_local, ct_id = 0>>
%29 = ktdp.construct_memory_view %c0, sizes: [16, 64]
      : memref<16x64xf16, #ktdp.memory_space<ct_local, ct_id = 1>>
...
// 32 tiles
%60 = ktdp.construct_distributed_memory_view(%28, %29, ..., %59
      : memref<16x64xf16, #ktdp.memory_space<ct_local, ct_id = 0>>, ...)
      : memref<64x512xf16> // the whole
%62 = ktdp.get_compute_tile_id : index
%63 = affine.apply affine_map<(d0) -> ((d0 floordiv 4) * 8)>(%62)
%64 = ktdp.construct_access_tile %60[%63, %c0]
      : memref<64x512xf16> -> !ktdp.access_tile<8x512xindex>
%65 = ktdp.load %64 : <8x512xindex> -> tensor<8x512xf16>
```



This poster is based on results obtained from the project, “Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems” (JPNP20017), subsidized by the New Energy and Industrial Technology Development Organization (NEDO).