



Integrating the IBM Spyre Accelerator

Avery Blanchard, David Grove, Antoni Viros i Martin

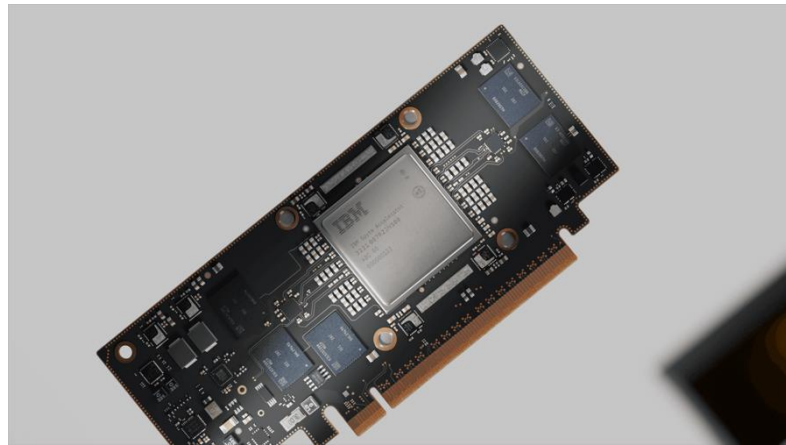
The IBM Spyre chip

The Hardware:

- Dataflow architecture
- 32 cores with dedicated matmul arrays
- 75W of power
- Can run multi-card workloads
- Inference focused

Where it isn't like a GPU:

- Work must be fully scheduled AOT
- Each core has dedicated scratchpad memory
- 128-byte SIMD lanes



Next-gen Spyre coming soon!

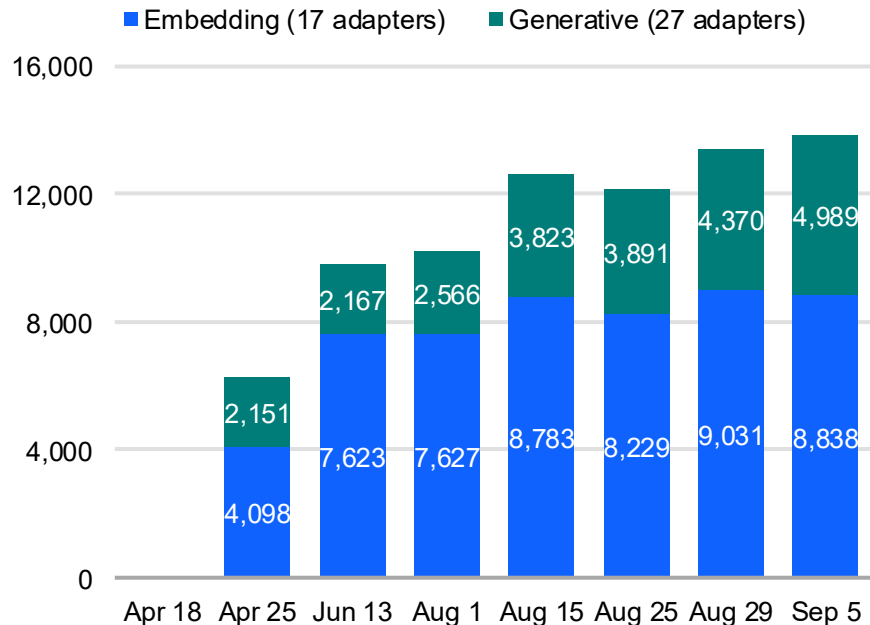
6x more FP8 compute

40x memory bandwidth

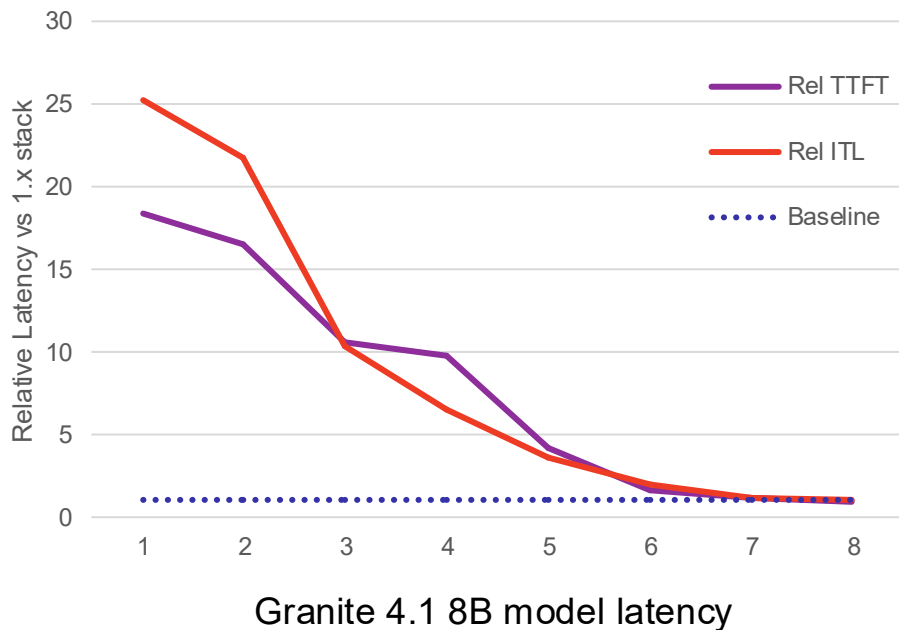
2x faster card interconnect

The torch-spyre project in a nutshell

Model support – Top 10k models in HF



Performance Parity Over Last 8 months

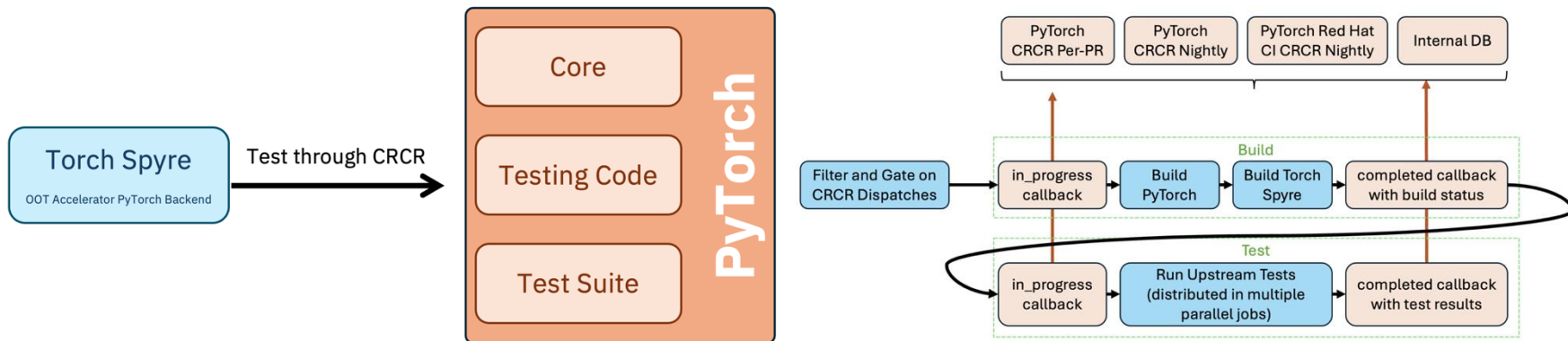


The torch-spyre project in a nutshell

- **torch-spyre** is an out of tree plugin to Pytorch enabling the use of the IBM Spyre accelerator using the PrivateUse1/OpenReg interfaces for **runtime**.
- It is also an integration of **inductor** as the **compiler** for Spyre.
- We have **contributed back** to Pytorch to improve interfaces in both OpenReg and Inductor for OOT devices.

Integrating with Pytorch CI: A recent success story

- PyTorch's Cross-Repository CI Relay (CRCR) gives out-of-tree accelerators a clean, scalable way to plug into upstream CI
- Torch-spyre is now at CRCR's L2 integration level!
 - This means we run Pytorch's CI on Spyre devices and these results show up in CRCR's HUD.

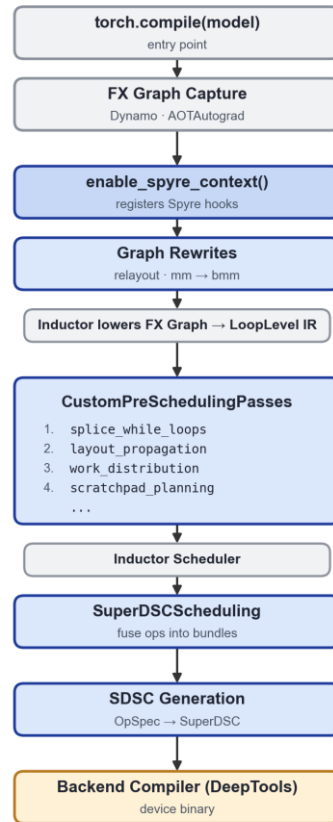


Leveraging Inductor for Spyre

- The rest of the talk will cover how we integrated inductor as our frontend compiler and how we use PyTorch's tools to write efficient kernels for Spyre.
- All this with **7x less code** than our previous compiler stack
(<https://pytorch.org/blog/extending-inductor-for-the-ibm-spyre-accelerator/>)

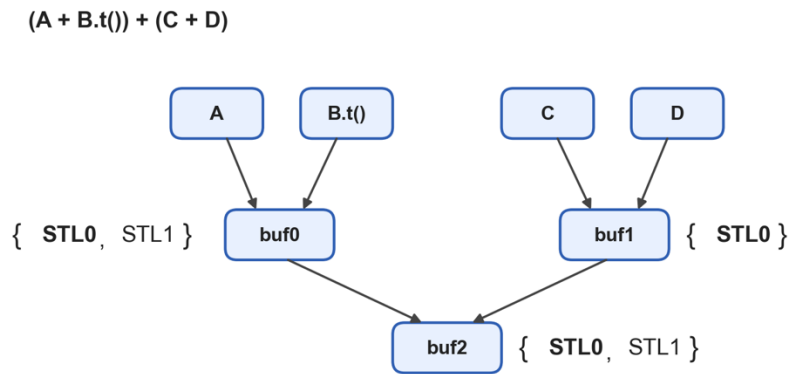
Overall Compiler Architecture

- **Inductor extensions for Spyre and a MLIR-based backend for dataflow hardware.**
- torch-spyre adds device-specific passes and configurations to Inductor.
 - The bulk of the compiler passes happens in CustomPreSchedulingPasses to enable graph rewriting.



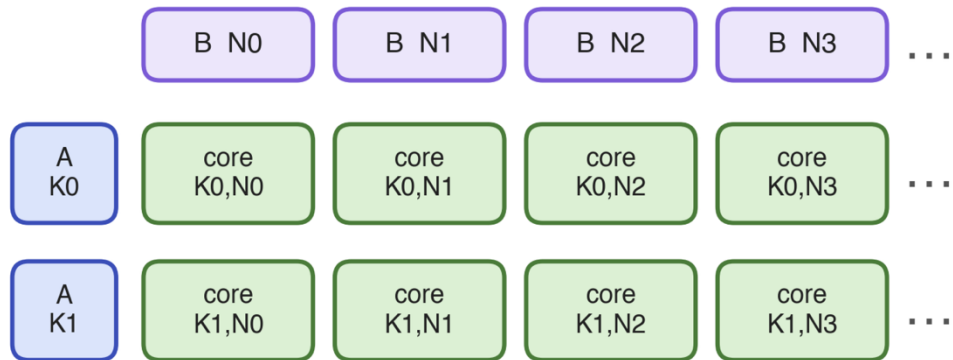
Optimizing Tensor Layout Assignment

- Spyre moves data in **sticks** (128-byte chunks), not individual elements.
- Operations constrain how **sticks** are assembled.
- Minimizing data movement across operations is a global optimization.
- Extend PyTorch layouts with a device-aware specialization.



Dividing Work Across Cores

- Plans division of a single operation's work across cores.
- Decisions account for operation and hardware constraints and cost model for matrix multiply.

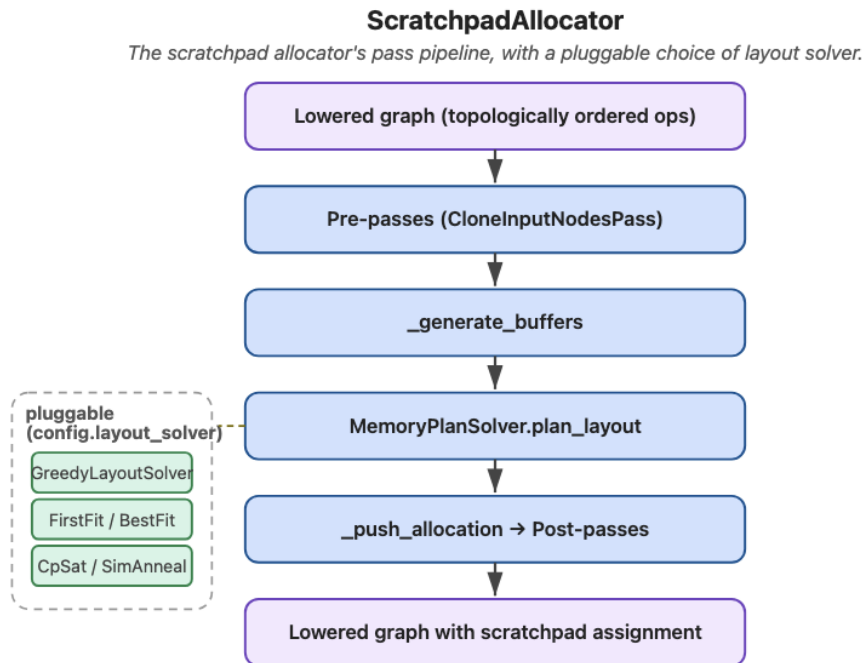


2-way on K $\times$ 16-way on N = 32 cores

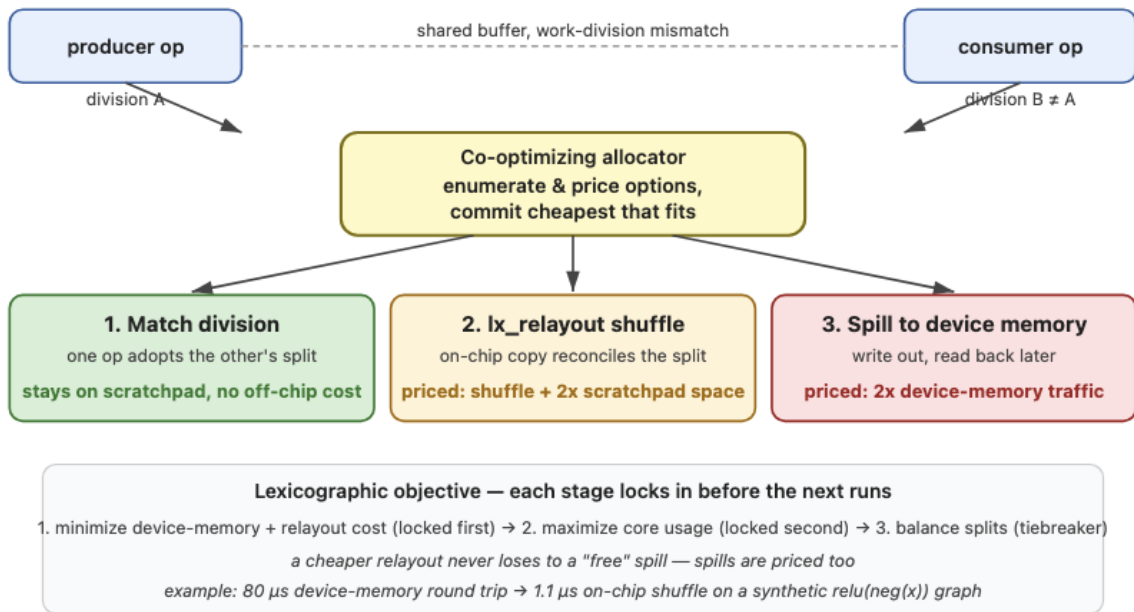
K0 and K1 cores add their partial sums

Planning Scratchpad Allocations

- Plans and optimizes allocation of intermediate tensors in a core's local memory.
- Leverages OR-Tools CPSat to consider the full space of candidates



Co-optimizing Core Division + Scratchpad Allocation

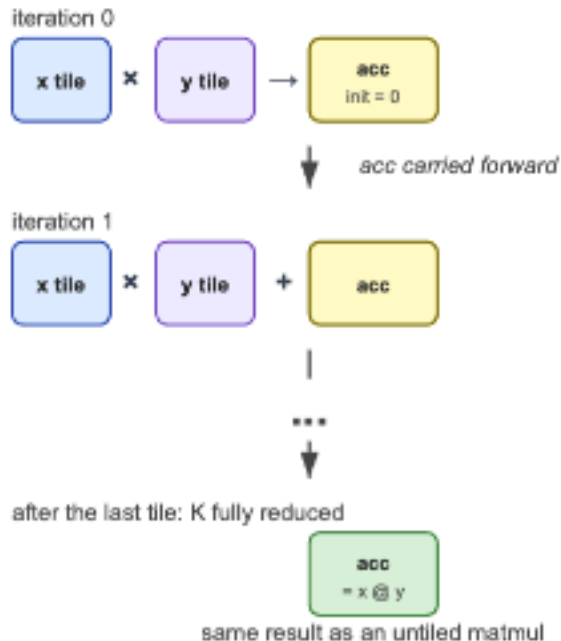


Tiling is an algorithmic change

- Problem: big intermediates spill out of the 2MB per-core scratchpad
- Solution: algorithmic restructuring
 - split-K matmul, online softmax, etc.
- Division of labor
 - Experts (or AI) rewrite the algorithm and create loop structure
 - Compilers and/or auto-tuners pick tile sizes
- `for_each_tile` == scan with tiling made explicit

One loop level, one carry

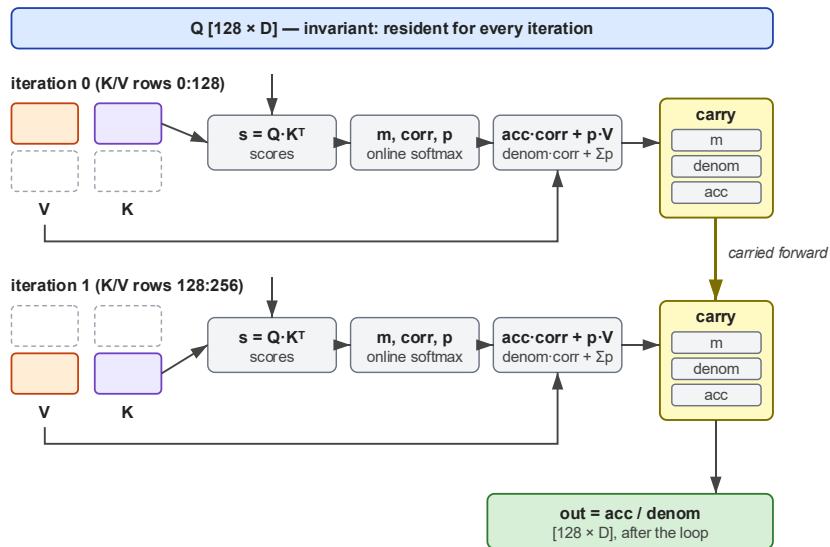
```
def mm_split_K(x, y, *, tile_size):  
    def body(acc, args):  
        x_tile, y_tile = args  
        return acc + x_tile @ y_tile, None  
  
    acc, _ = for_each_tile(  
        body,  
        (x, y),  
        dims=(1, 0),    # K tiled  
        tile_size=tile_size,  
        init=torch.zeros(M, N),  
    )  
    return acc
```



Flash attention: a three value carry

```
def attention(Q, K, V, tile=128):  
    def body(carry, tiles):  
        m, denom, acc = carry  
        k, v = tiles  
        s = Q @ k.transpose(-1, -2)  
        m_new = torch.maximum(m, s.amax(-1, keepdim=True))  
        corr = torch.exp(m - m_new)  
        p = torch.exp(s - m_new)  
        denom = denom * corr + p.sum(-1, keepdim=True)  
        acc = acc * corr + p @ v  
        return (m_new, denom, acc), None
```

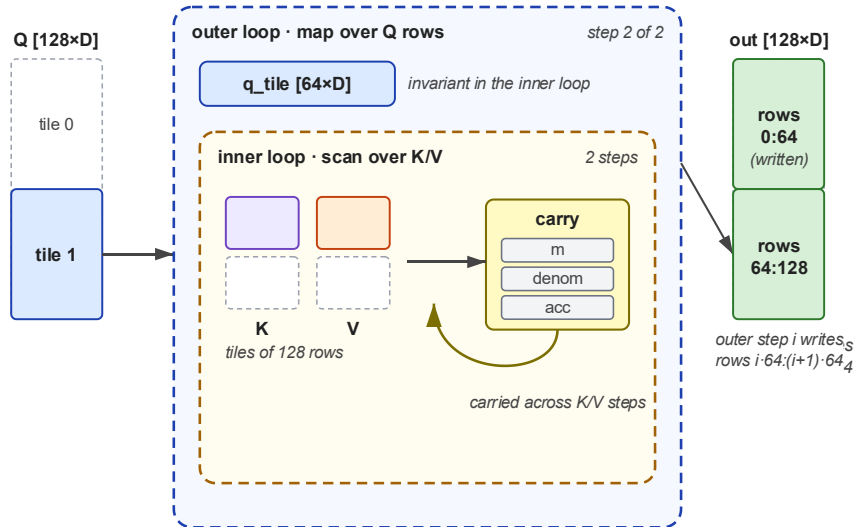
```
(m, denom, acc), _ = for_each_tile(  
    body,  
    (k, v),  
    dims=(0, 0),  
    tile_size=tile,  
    init=(m0, denom0, acc0),    # -inf, 0, 0  
)  
return acc / denom
```



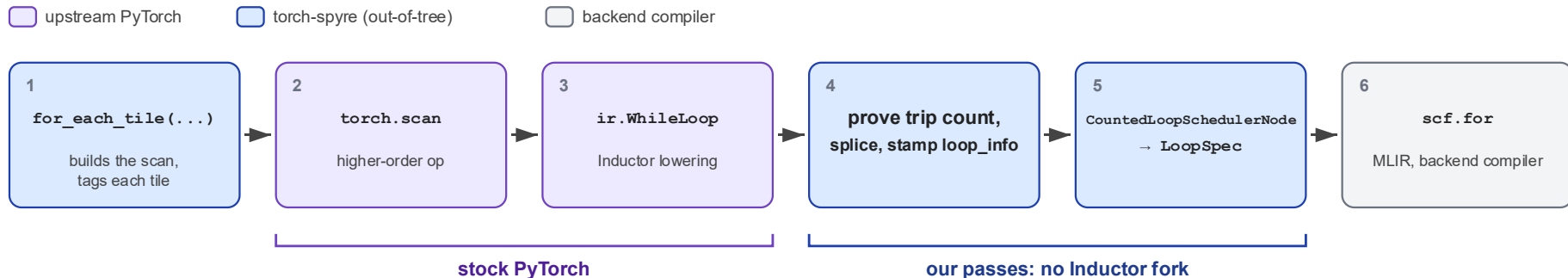
Simplified: production adds masks, GQA, and an `-inf` guard

Nesting: map over queries, scan over keys

```
def nested(Q, K, V):  
    def body(_, tiles):  
        (q_tile,) = tiles  
        return None, attention(q_tile, K, V)  
  
    _, out = for_each_tile(  
        body,  
        (Q,),  
        dims=(0,),  
        tile_size=64,  
        out_dim=0,  
    )  
    return out
```



A thin frontend over scan



What scan is missing for tiling

scan gap (as of PyTorch 2.13)	Our workaround
Single flat tensor output doesn't lower	Carry a step counter alongside
Python float closure becomes SymFloat, rejected	<code>specialize_float=True</code> while tracing
<code>.item()</code> on the induction variable fails under an AOT fake mode	<code>allow_scalar_outputs = True</code>
Stacked output on <code>`dim != 0`</code> forces a copy	Planned: write each tile in place
Caching of graphs with scan disabled by default	Patch config to enable caching
Information loss: “which dimension is being tiled”?	CustomOp to carry this information through HOP expansion to <code>LoopLevelIR</code>

Experience with `for_each_tile`

- The body is ordinary PyTorch: no new DSL
- Change `tile_size`, rerun
- Check against an eager CPU reference
- Current restrictions
 - one output tile in map mode
 - equal-size tiles
 - symbolic tile count implementation is work-in-progress
- Used throughout `torch-spyre`, `spyre-inference`, `hf-adapters`

Takeaways

- PyTorch + Inductor support out-of-tree accelerators
without compromising on performance
- Modest upstream improvements can make a difference
 - Decompositions (merged in PyTorch 2.12)
 - Device memory layouts
 - Tiling construct
- Spyre + torch-spyre is a proof point
we're bringing the lessons back upstream

<https://github.com/torch-spyre/torch-spyre>



Integrating the IBM Spyre Accelerator

Talk feedback

IBM @ PyTorch2026

Presentation schedule,
publications, contacts, careers

