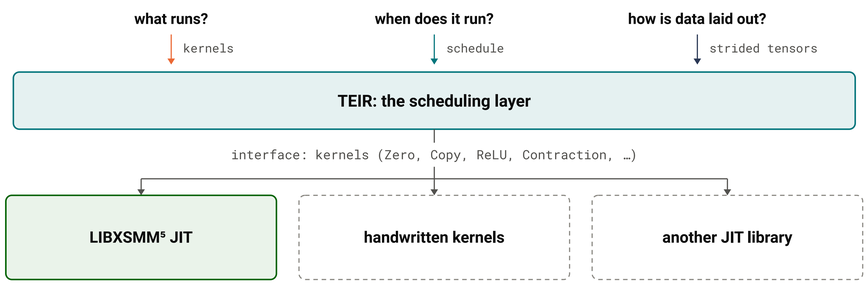
TEIR: A Tiled Execution IR for Tensor Contractions

Felix Lindner and Alex Breuer, Friedrich Schiller University Jena

https://tnzr.org/sc26/teir

1. Motivation: Scheduling Tensor Contractions

- Tensor contractions are core computational kernels in scientific computing and ML
- The best schedule depends on the properties of the contraction and the microarchitecture
- TVM[1] separates the schedule from the computation but finds it by auto-tuning
- BLIS[19] and TBLIS[2] optimize the schedule by library heuristics
- TTGT[20] (e.g., PyTorch[4] einsum) transposes whole tensors around a GEMM



2. TEIR: Kernels, Schedule, and Strided Tensors

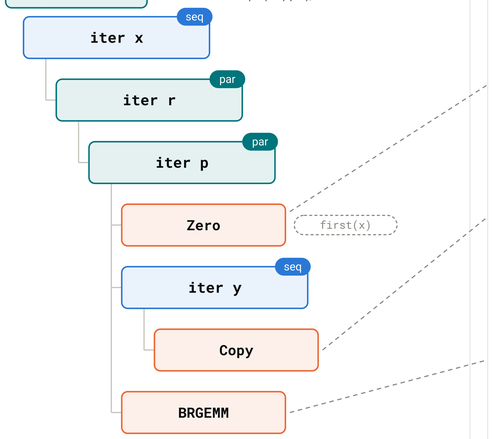
TEIR configuration: axes, schedule, primitives

`trus,pqtu->pqrs |p|=16 |q|=64 |r|=16 |s|=64 |t|=16 |u|=64 FP32` sequential iter · parallel iter · invocation

TEIR-Axes `addr = base + Σ (offset + stride·idx)`

axis	p	q	r	s	x (|t| = |x|·|y|)	y	u
extents	16	64	16	64	2	8	64
tensor					**byte strides (0 = axis not used)**		
xyrus	0	0	16384	4	2097152	262144	256
pqxyu	262144	4096	0	0	2048	256	4
pqrs	262144	4096	256	4	0	0	0
yus (private, per thread)	0	0	0	4	0	16384	256

TEIR-Schedule ordered forest: loops (seq/par), kernel invocations as leaves



TEIR-Primitives `bold = kernel axes`

zero_out
```
Zero {M: [s], N: [q]}
{out = "pqrs"}
```

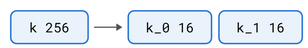
pack_I
```
Copy {M: [s], N: [u]}
{in0 = "xyrus",
out = "yus"}
```

main
```
BRGEMM {M: [s],
N: [q], K: [y, u]}
{in0 = "yus", in1 = "pqxyu",
out = "pqrs"}
```

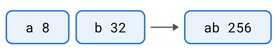
3. Transformations on TEIR

sequential iter · parallel iter · kernel · memory

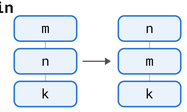
splitLoop



mergeLoops



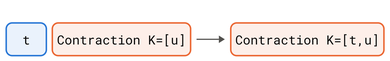
reorderChain



setPolicy



promoteToKernel



applyPacking



4. Compilation Pipeline

axes / memory · kernels · schedule

trus,pqtu->pqrs
`|p|=16 |q|=64 |r|=16 |s|=64 |t|=16 |u|=64`

emitter: einsum → axes & primitives

TEIR (MLIR[6] dialect)
```
teir.optimize_contraction (%A, %B, %C)
<axes, primitives> {12Size, numThreads, …}
```
no schedule yet — only what to compute and the machine model

`--optimize-teir-contraction`

TEIR (MLIR dialect)
```
teir.contraction (%A, %B, %C)
<axes, roots, schedule, primitives>
```

`--convert-teir-to-xsmm`

XSMM
```
%k = einsum_xsmm.brgemm.dispatch [64,64,64,…]
einsum_xsmm.brgemm(%k, %A, %B, %C, %c8)
```

`--convert-xsmm-to-func → LLVM`

Microkernels

generated at runtime by LIBXSMM[5] for AVX-512[7], NEON[8], and SME[9]

1 Normalize axes

drop unit axes, sort by stride, fuse contiguous axes

2 Form the kernel

split M/N/K to target tile sizes, promote to roles
2nd K → BRGEMM

3 First / last kernels (optional)

insert Zero (init) and fused unary kernels
around the contraction kernel

4 Block & parallelize

L2 budget → parallel M/N blocks
L3 budget → sequential K blocks

5 Build the schedule

axis chain → ordered forest with invocation leaves

6 Pack (optional)

pack strided operands into contiguous buffers
(Copy kernel)

5. Evaluation on AVX-512, NEON, and SME

Zen 5[10] (AVX-512[7])	Grace CPU[11] (NEON[8])	M4 Max[12] (SME[9])
1.9×	**1.5×**	**1.5×**
vs. auto-tuned TVM[1]	vs. PyTorch[4]	vs. PyTorch[4]

geomean speedup over the next-best backend

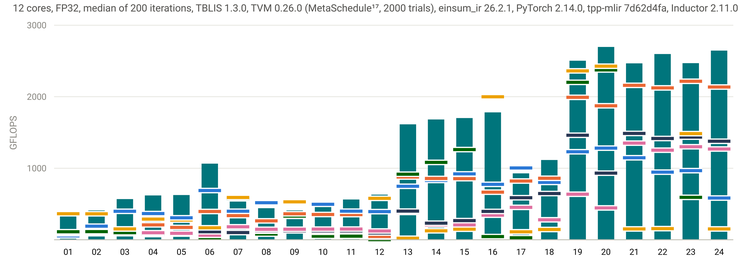
median measured GFLOPS per TCCG[3] case (row-major)

ours · TBLIS[2] · TVM · einsum_ir[13] · PyTorch · tpp-mlir[14] · Inductor[15] · ExecuTorch[16]

bar: ours, dash: other backend (missing dash = timeout or error)

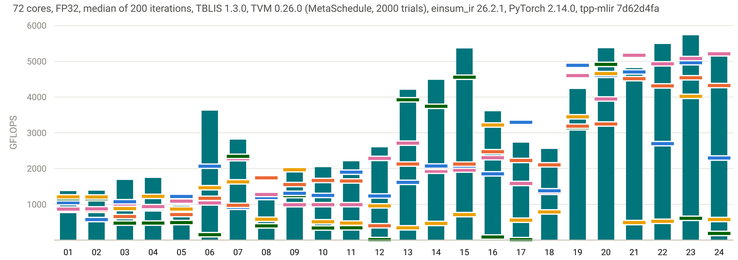
AVX-512 — HP Z2 Mini G1a, AMD Ryzen AI Max PRO 390

12 cores, FP32, median of 200 iterations, TBLIS 1.3.0, TVM 0.26.0 (MetaSchedule[17], 2000 trials), einsum_ir 26.2.1, PyTorch 2.14.0, tpp-mlir 7d62d4fa, Inductor 2.11.0



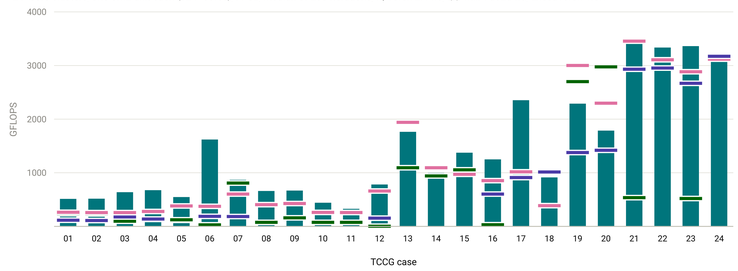
NEON — NVIDIA Grace CPU

72 cores, FP32, median of 200 iterations, TBLIS 1.3.0, TVM 0.26.0 (MetaSchedule, 2000 trials), einsum_ir 26.2.1, PyTorch 2.14.0, tpp-mlir 7d62d4fa



SME — Mac Studio, Apple M4 Max

2 P-core clusters, one SME unit each (12 P-cores), FP32, median of 200 iterations, PyTorch 2.14.0, tpp-mlir 7d62d4fa, ExecuTorch 1.5.1



6. Summary and Future Work

- One intermediate representation from einsum to generated microkernels
- Highest median GFLOPS in 54 of 72 cases

Future work:

- Overlapping compute with data movement, e.g., asynchronous packing
- Einsum trees[13] (contraction path × schedule)

Interested in this work? Reach out!
felix.lindner@uni-jena.de, https://scalable.uni-jena.de

References: [1] TVM: Chen et al., OSDI '18 · [2] TBLIS: Matthews, SIAM J. Sci. Comput. '18 · [3] TCCG: Springer & Bientinesi, ACM TOMS '18 · [4] PyTorch: Paszke et al., NeurIPS '19 · [5] LIBXSMM: Heinecke et al., SC '16 · [6] MLIR: Lattner et al., CGO '21 · [19] BLIS: Van Zee & van de Geijn, ACM TOMS '15
[7] AVX-512: Intel SDM · [8] NEON: Arm ARM DDI0487 · [9] SME: Arm DDI0616 · [10] Zen 5: AMD Ryzen AI Max PRO 390 · [11] Grace CPU: NVIDIA Grace CPU Superchip · [12] M4 Max: Apple Mac Studio (2025) · [20] TTGT: Solomonik et al., JPDC '14
[13] einsum_ir, Einsum Trees: Breuer et al., ASPLOS '25 · [14] tpp-mlir: Golin et al., arXiv '24 · [15] Inductor: Ansel et al., ASPLOS '24 · [16] ExecuTorch: Nachin et al., MLSys '26 · [17] MetaSchedule: Shao et al., NeurIPS '22 · [18] Claude Fable 5: Anthropic, 2026, anthropic.com/news/claude-fable-5-mythos-5

Anthropic Claude Fable 5[18] was used for editing the text and figures of this poster and the accompanying summary. The authors take full responsibility for all content.

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA