TEIR: A Tiled Execution IR for Tensor Contractions

Felix Lindner
Friedrich Schiller University Jena
Jena, Germany
felix.lindner@uni-jena.de

Alex Breuer
Friedrich Schiller University Jena
Jena, Germany
alex.breuer@uni-jena.de

I. Problem and Motivation

Tensor contractions are core computational kernels in scientific computing [1], [2] and machine learning [3] workloads. Efficiently executing a tensor contraction requires hardware-aware tiling and scheduling tile computations across processing resources (parallelization) and over time (loop ordering). This yields a high-dimensional schedule space, and the best schedule depends on the properties of the contraction and on the microarchitecture. Existing systems typically search this space with expensive auto-tuning (TVM's MetaSchedule [4], [5]) or optimize it with library heuristics (BLIS [6], TBLIS [2]). Alternatively, transpose-transpose-GEMM-transpose (TTGT) reformulations [7] transpose whole tensors before and after the GEMM. Recent systems expose tiles as explicit units of computation, from domain-specific programming environments such as cuTile [8] and Triton-CPU [9] to optimized tile-based primitives such as KleidiAI [10], CUTLASS [11], and LIBXSMM [12]. We introduce the Tiled Execution Intermediate Representation (TEIR), which explicitly represents tiling and scheduling decisions: it separates what runs (kernels on tiles), when it runs (an execution schedule), and how data is laid out (axes with per-tensor byte strides and offsets). We implement TEIR as an MLIR dialect whose optimization passes schedule a contraction and lower it to LIBXSMM kernels. Across 24 contractions on three CPU platforms using AVX-512, NEON, and SME, TEIR achieves 1.5–1.9× the geometric-mean throughput of the next-best backend on each platform and leads in 54 of 72 cases against all measured backends.

II. The Intermediate Representation

As shown in Fig. 1, TEIR describes a tensor operation with three components. A TEIR-Axis carries an extent and per-tensor byte strides and offsets. A TEIR-Primitive describes an operation on low-dimensional tiles, such as Zero, Copy, ReLU, or (BR)GEMM, and the axes it consumes. A TEIR-Schedule combines axes and primitive configuration into an execution forest containing loop nodes (sequential and parallel) and leaf nodes that invoke kernels. As shown in Fig. 2, our compiler parses an einsum string and the extents of the axes into a `teir.optimize_contraction` operation. Next the compiler applies TEIR passes to rewrite the program into a scheduled `teir.contraction` operation. Last it lowers to executable code using the XSMM dialect. There are six transformations on TEIR. `splitLoop` splits one axis into two and partitions its extent. `mergeLoops` merges two axes into one. `reorderChain` reorders loop nodes in a schedule. `setPolicy` sets the policy of a loop node.

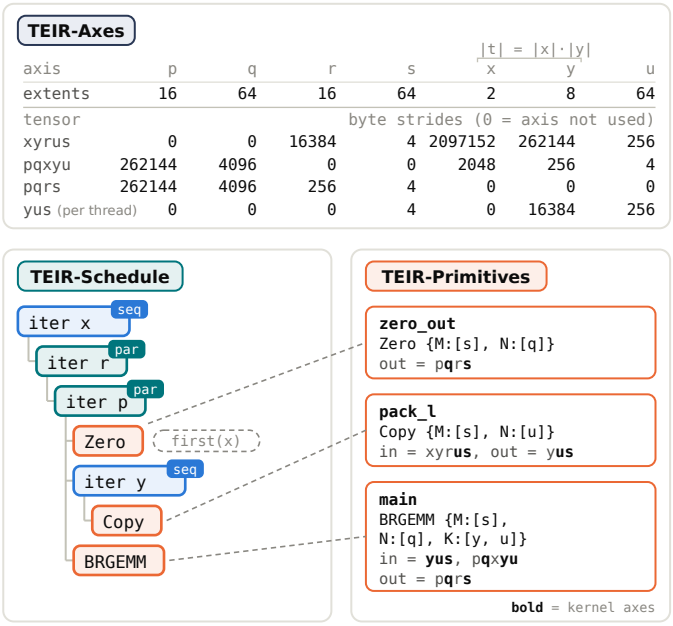


Fig. 1. A program containing TEIR-Axes, TEIR-Schedule, and TEIR-Primitives of the running example `trus,pqtu->pqrs` (FP32) after splitting `t` into `x` and `y`.

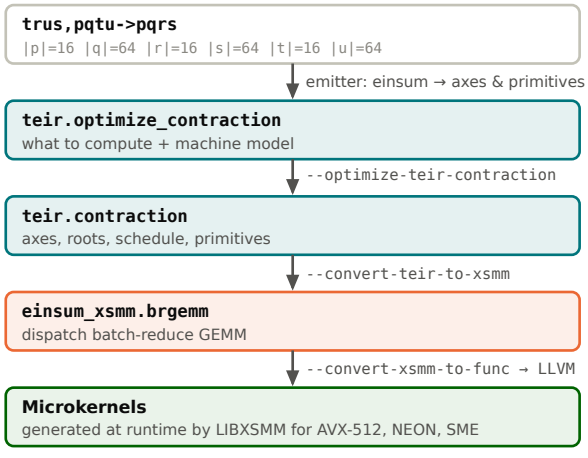


Fig. 2. Compilation pipeline using the TEIR and XSMM MLIR dialects.

`promoteToKernel` moves an axis from the schedule into a primitive. `applyPacking` adds a Copy node to the schedule that packs a strided subtensor into a contiguous per-thread buffer. A deterministic heuristic composes these transformations: it derives optimized kernel tile sizes, cache blocking, and parallelization from cache sizes and core count.

III. Implementation and Evaluation

We implemented TEIR as a dialect in MLIR [13] (Fig. 2): the optimizer rewrites `teir.optimize_contraction` into `teir.contraction`, and two conversion passes lower it through the XSMM dialect to loops over function calls to LIBXSMM kernels [12]. Standard MLIR and LLVM passes then produce LLVM IR.

We benchmarked the TEIR toolchain on an HP Z2 Mini G1a with an AMD Ryzen AI Max PRO 390 SoC [14] using AVX-512, an NVIDIA Grace CPU [15] using NEON, and a Mac Studio with an Apple M4 Max SoC (16-core CPU) [16] using SME [17], [18]. As a benchmark, we used the 24 TCCG contractions reported by Springer and Bientinesi [1]. TCCG stores tensors column-major. We kept its index strings but used row-major storage, which reverses the index order of every tensor. All measurements used FP32, 10 warm-ups, and 200 iterations per contraction. We used all 12 cores of the Ryzen system, the two SME units of both P-Core clusters of the M4 Max, and all 72 cores of one Grace CPU. Fig. 3 presents the geometric mean of the median performance on each system. We compare the performance on the Ryzen system with TBLIS 1.3.0 [2], TVM 0.26.0 [4], TPP-MLIR (7d62d4f) [19], PyTorch 2.14.0 [20], TorchInductor 2.11.0 [21], and einsum_ir 26.2.1 [22]. On the Grace CPU, we compare with TBLIS, TVM, TPP-MLIR, PyTorch, and einsum_ir; on the M4 Max, with ExecuTorch 1.5.1 [23], PyTorch, and TPP-MLIR. TVM auto-tuned each contraction with MetaSchedule [5] for 2000 trials. A backend-contraction pair counts as successful only if compilation, warm-up, and 200 timed iterations finished without error and in time. TEIR reached a geometric-mean median performance of 1022, 2833, and 1115 FP32 GFLOPS, respectively: $1.9\times$, $1.5\times$, and $1.5\times$ over the next-best backend (TVM, PyTorch, PyTorch).

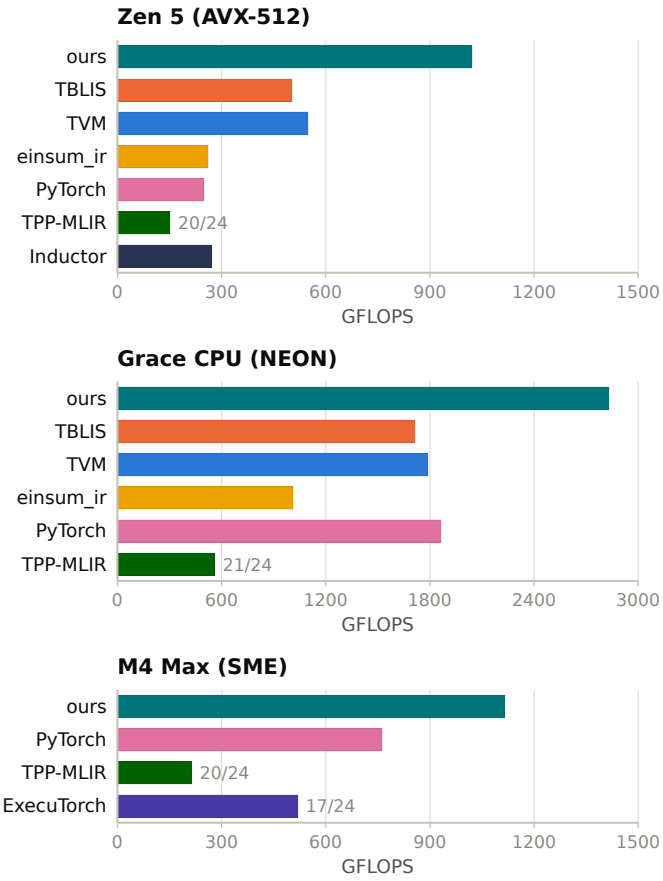


Fig. 3. Geometric mean of the median performance over the evaluated TCCG contractions. Gray labels: successful contractions out of 24.

Acknowledgment

This work was supported by the Carl Zeiss Foundation (project "Interactive Inference").

Anthropic Claude Fable 5 [24] was used for editing the text and figures of this summary and the accompanying poster. The authors take full responsibility for all content.

References

[1] P. Springer and P. Bientinesi, "Design of a High-Performance GEMM-like Tensor–Tensor Multiplication," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 3, pp. 28:1–28:29, Jan. 2018.

[2] D. A. Matthews, "High-Performance Tensor Contraction without Transposition," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. C1–C24, Jan. 2018.

[3] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions," Feb. 2018, arXiv:1802.04730.

[4] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594.

[5] J. Shao, X. Zhou, S. Feng, B. Hou, R. Lai, H. Jin, W. Lin, M. Masuda, C. H. Yu, and T. Chen, "Tensor Program Optimization with Probabilistic Programs," in *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

[6] F. G. Van Zee and R. A. van de Geijn, "BLIS: A Framework for Rapidly Instantiating BLAS Functionality," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 3, pp. 14:1–14:33, Jun. 2015.

[7] E. Solomonik, D. Matthews, J. R. Hammond, J. F. Stanton, and J. Demmel, "A massively parallel tensor contraction framework for coupled-cluster computations," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3176–3190, 2014.

[8] NVIDIA Corporation, "NVIDIA CUDA Tile," https://developer.nvidia.com/cuda/tile, accessed: 2026-08-08.

[9] Triton Developers, "Triton-CPU: An Experimental CPU Backend for Triton," https://github.com/triton-lang/triton-cpu, accessed: 2026-10-01.

[10] Arm Ltd., "KleidiAI," https://gitlab.arm.com/kleidi/kleidiai, accessed: 2026-10-01.

[11] NVIDIA Corporation, "CUTLASS: CUDA Templates for Linear Algebra Subroutines," https://github.com/NVIDIA/cutlass, accessed: 2026-08-08.

[12] A. Heinecke, G. Henry, M. Hutchinson, and H. Pabst, "LIBXSMM: Ac-

celerating Small Matrix Multiplications by Runtime Code Generation," in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2016, pp. 981–991.

[13] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling Compiler Infrastructure for Domain Specific Computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Feb. 2021, pp. 2–14.

[14] Advanced Micro Devices, Inc., "AMD Ryzen AI Max PRO 390," https://www.amd.com/en/products/processors/laptop/ryzen-pro/ai-max-pro-300-series/amd-ryzen-ai-max-pro-390.html, accessed: 2026-08-08.

[15] NVIDIA Corporation, "NVIDIA Grace CPU Superchip Architecture In-Depth," https://developer.nvidia.com/blog/nvidia-grace-cpu-superchip-architecture-in-depth/, accessed: 2026-08-08.

[16] Apple Inc., "Apple unveils new Mac Studio, the most powerful Mac ever," https://www.apple.com/newsroom/2025/03/apple-unveils-new-mac-studio-the-most-powerful-mac-ever/, Mar. 2025, press release. Accessed: 2026-08-08.

[17] Arm Ltd., "The Scalable Matrix Extension (SME), for Armv9-A," https://support.arm.com/documentation/ddi0616/latest, document DDI0616. Accessed: 2026-07-29.

[18] S. Remke and A. Breuer, "Hello SME! Generating Fast Matrix Multiplication Kernels Using the Scalable Matrix Extension," in *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024.

[19] R. Golin, L. Chelini, A. Siemieniuk, K. Madhu, N. Hasabnis, H. Pabst, E. Georganas, and A. Heinecke, "Towards a high-performance AI compiler with upstream MLIR," Apr. 2024, arXiv:2404.15204.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, pp. 8024–8035.

[21] J. Ansel, E. Yang, H. He *et al.*, "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, 2024, pp. 929–947.

[22] A. Breuer, M. Blacher, M. Engel, J. Giesen, A. Heinecke, J. Klaus, and S. Remke, "Einsum Trees: An Abstraction for Optimizing the Execution of Tensor Expressions," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25)*, 2025, pp. 275–292.

[23] M. Nachin, D. Desai, S. S. Jia, C. Lai, M. Liu, J. Szwejbka, R. Alvarez, R. J. Ascani, D. Bort, M. Candales, A. Caples, Y. Cao, Z. Chen, S. Chintala, G. Comer, T. Islam, S. Jia, T. Karuturi, J. Khuu, A. Kukkadapu, T. Manlaibaatar, A. Or, K. Patel, S. Pothapragada, L. Qiu, S. Rao, O. Reblitz-Richardson, M. Ren, S. Roy, A. Shoumikhin, S. Wolchok, G. Yang, A. Yi, M. Yuan, H. Zhang, J. Zhang, J. Zhang, S. Zhang, and C. C. Bilgin, "ExecuTorch – A Unified PyTorch Solution to Run ML Models On-Device," in *Proceedings of Machine Learning and Systems*, vol. 8, 2026, MLSys 2026.

[24] Anthropic, "Claude Fable 5," https://www.anthropic.com/news/claude-fable-5-mythos-5, 2026, large language model, announced 2026-06-09. Accessed: 2026-10-01.