Tensor Contractions on AWS Trainium2

Max Engel
Friedrich Schiller University Jena
Jena, Germany
max.engel@uni-jena.de

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

***Abstract*—Tensor contractions are a fundamental operation across computational science and machine learning, yet reaching high performance for them on domain-specific accelerators built around large systolic arrays, such as AWS Trainium, remains challenging. We present a compiler that optimizes a binary tensor contraction and lowers it into an efficient Neuron Kernel Interface (NKI) kernel. It translates the contraction to a compact intermediate representation, explores kernel tiling and SBUF tiling choices, and emits NKI code. On Trainium2 the generated kernels reach near-peak throughput on large matrix multiplications and large tensor contractions.**

I. Motivation

Tensor contractions arise in simulating quantum circuits [1], in many-body and coupled-cluster electronic-structure methods [2], and throughout deep learning [3]. As these workloads grow, so does the need for implementations that handle the diverse range of tensor shapes and sizes on modern hardware architectures.

Domain-specific accelerators built around large systolic arrays, such as Google's TPU and AWS Trainium, deliver far higher throughput and energy efficiency than general-purpose hardware [4], but are tuned for large, regular matrix-matrix multiplications. A general contraction rarely looks like that. For the contraction `eafd,fbec→abcd`, a case from the TCCG benchmark [5], every free and reduction axis size is between 84 and 96, well below 128. Consequently, two axes alone cannot fully utilize Trainium's 128×128 array of processing elements.

Libraries commonly reduce contractions to GEMM with the Transpose-Transpose-GEMM-Transpose (TTGT) approach [6]. It transposes the operands into a GEMM-compatible layout, calls one large matrix multiply, and transposes the resulting matrix to the layout of the output tensor. The transpositions, however, are an overhead. They only rearrange data in memory and perform no computation. On CPUs, GETT [5] and TBLIS [7] avoid explicit transposes by packing, effectively fusing the transpositions with the computation.

For the Trainium2 architecture shown in Figure 1, a packing-based approach requires combining several axes onto each engine dimension, tiling, caching in on-chip SBUF, double-buffering, and distributing across cores, while keeping the data movements fast enough to feed the compute engines.

We automate these coupled decisions for the Neuron Kernel Interface (NKI) programming model of Trainium. We search the space of possibilities, prune weak candidates, and rank the best with a measurement-based performance model.

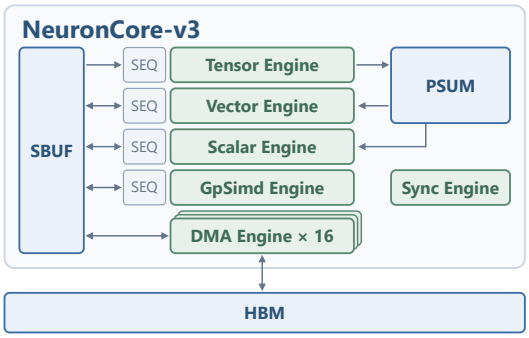


Fig. 1. One NeuronCore of Trainium2. The compute engines exchange data with SBUF, the software-managed on-chip scratchpad, while the DMA engines move data between SBUF and off-chip HBM. Redrawn after the AWS Neuron SDK documentation [8].

II. Approach

Our toolchain compiles a binary tensor contraction, given as an einsum equation and its axis sizes, into an efficient NKI kernel. The compilation proceeds in three stages.

(1) Lower to TEIR builds an initial Tiled Execution Intermediate Representation (TEIR) tree with scalar operations.

(2) Optimize TEIR rewrites the TEIR tree through a pipeline of three passes: *normalize contraction*, *tiling search*, and *multicore parallelization*. The normalize contraction pass chooses a canonical order for the iterations and fuses them. The tiling search pass optimizes the contraction for a single core. It enumerates kernel tiling candidates (which axes combine onto the tensor engine's dimensions) crossed with SBUF tiling candidates (the number of elements per axis held in the on-chip scratchpad SBUF). Because this space is large, most candidates are pruned cheaply. Kernel tilings must keep a unit-stride axis as an innermost dimension of the kernel, and SBUF tilings are reduced to a Pareto front of maximal-reuse choices. Only the survivors are realized into full schedules (with their memory allocations, data movements, and tensor-engine operations) and scored by a measurement-based performance model. From the realized schedule, the model estimates the time on each engine (the tensor, vector, scalar, and DMA engines) and accounts for their overlap, as well as pipeline startup and cooldown terms. Finally, the multicore parallelization pass distributes the best schedule across cores.

(3) Generate NKI Code walks the optimized TEIR and emits an NKI kernel.

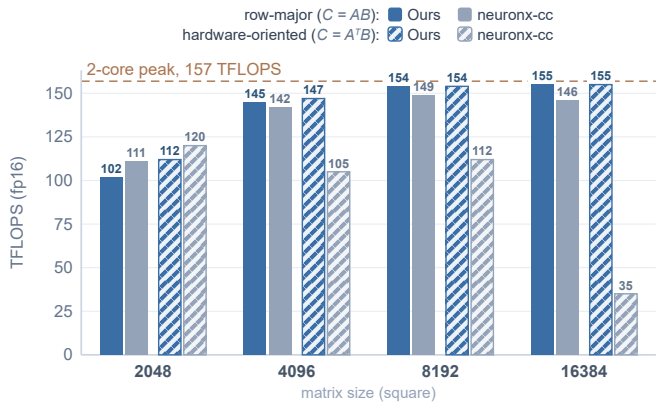


Fig. 2. Square matrix multiplication on two NeuronCores of Trainium2, with the row-major layout $C = AB$ and the hardware-oriented layout $C = A^T B$.

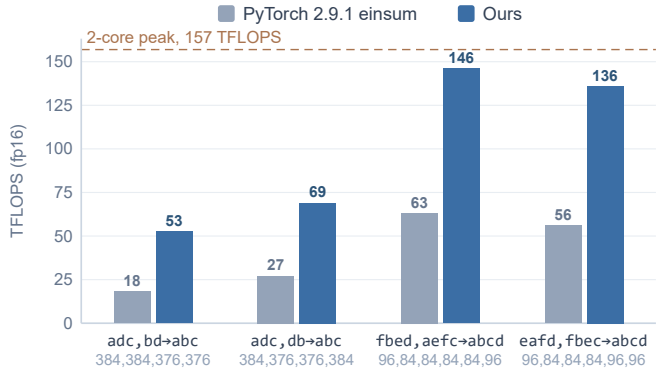


Fig. 3. Four tensor contractions from the TCCG benchmark [5] in row-major data layout on two NeuronCores of Trainium2. Axis sizes are listed in alphabetical index order.

III. Results

All measurements were performed on a trn2.3xlarge instance with one Trainium2 device using two NeuronCores (LNC=2). We compare against the vendor compiler neuronx-cc 2.26.6360.0 and PyTorch 2.9.1.

Figure 2 shows our measurements of square matrix multiplication for sizes 2048 to 16384. We achieve high performance for both the row-major data layout $C = AB$ and the hardware-oriented data layout $C = A^T B$, which matches the tensor engine's native primitive. From a matrix size of 4096 upward we outperform the vendor compiler neuronx-cc in both layouts and reach up to 155 TFLOPS. This is **98.7%** of the 157 TFLOPS two-core fp16 peak.

Figure 3 shows four cases from the TCCG benchmark [5] in row-major data layout. The two smallest are DMA-bound. SBUF can hold more than 4.5 MB per tensor even with double-buffering, but their second inputs are less than 300 KB. Because of the limited reuse, our kernels do not hide the DMA transfers completely. The larger cases offer better reuse, reaching 146 TFLOPS on `fbed,aefc→abcd`. Across all four we are **2.3× to 2.9× faster** than `torch.einsum`.

Acknowledgment

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

We used Claude Code (Fable 5, Anthropic) for editing this work and take complete accountability for the contents.

References

[1] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1137/050644756

[2] A. A. Auer, G. Baumgartner, D. E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. Harrison, S. Krishnamoorthy, S. Krishnan, C.-C. Lam, Q. Lu, M. Nooijen, R. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov, "Automatic code generation for many-body electronic structure methods: The tensor contraction engine," *Molecular Physics*, vol. 104, no. 2, pp. 211–228, Jan. 2006.

[3] J. Kossaifi, Z. C. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar, "Tensor regression networks," *J. Mach. Learn. Res.*, vol. 21, no. 123, pp. 1–21, Jan. 2020.

[4] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[5] P. Springer and P. Bientinesi, "Design of a high-performance GEMM-like tensor–tensor multiplication," *ACM Trans. Math. Softw.*, vol. 44, no. 3, Jan. 2018. [Online]. Available: https://doi.org/10.1145/3157733

[6] S. Hirata, "Tensor contraction engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories," *The Journal of Physical Chemistry A*, vol. 107, no. 46, pp. 9887–9897, 2003. [Online]. Available: https://doi.org/10.1021/jp034596z

[7] D. A. Matthews, "High-performance tensor contraction without transposition," *SIAM Journal on Scientific Computing*, vol. 40, no. 1, pp. C1–C24, 2018. [Online]. Available: https://doi.org/10.1137/16M108968X

[8] Amazon Web Services, "Trainium2 architecture guide for NKI," 2026. [Online]. Available: https://awsdocs-neuron.readthedocs-hosted.com/en/v2.31.0/nki/guides/architecture/trainium2_arch.html