Tensor Compilers for AI and HPC

a holistic overview of the system design and inner workings of tensor and machine learning compilers

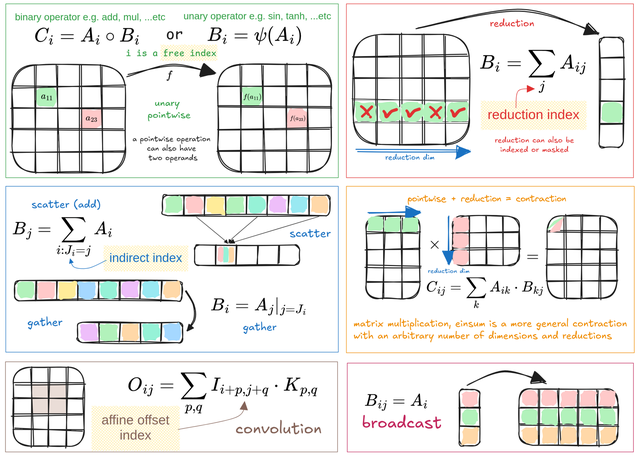
Paul Mifsud Stefan Kesselheim
Juelich Supercomputing Centre, Forschungszentrum Juelich

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

0 Motivation

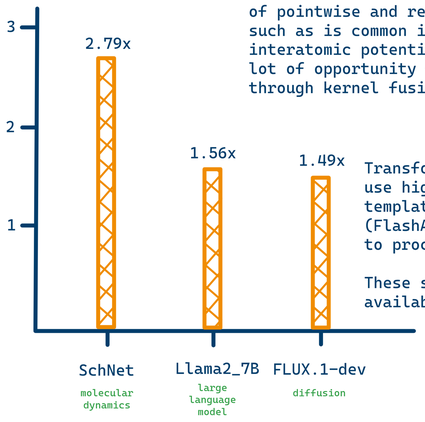
1. Why are tensor compilers necessary?
2. How do tensor compilers work?
   - What principles are they based on?
   - How do they achieve their speedups?
3. What are the limitations of tensor compilers?

1 Taxonomy of Operators



Tensor Programs are expressed in the form of a directed acyclical graph (DAG) of operations on tensors (n-dim arrays).
These operators can be categorized into classes. In this work, we choose a classification based on indexing and memory access patterns. There are four types of indices: free indices, reduction indices, indirect indices and affine offset indices.

6 Evaluation

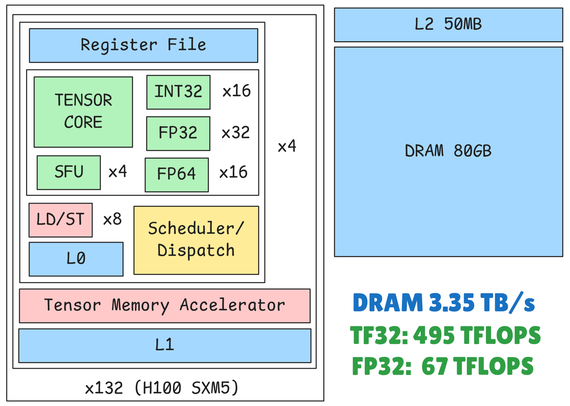


Model architectures which include a lot of pointwise and reduction operations, such as is common in machine learned interatomic potential models provide a lot of opportunity for optimization through kernel fusion.

Transformer based models, use highly optimized templated kernels (FlashAttention) to process self-attention.

These same kernels are also available in eager mode.

2 GPU Principles



DRAM 3.35 TB/s
TF32: 495 TFLOPS
FP32: 67 TFLOPS

NVIDIA H100 SXM Architecture

Principle 1: Optimize Global Memory Traffic.

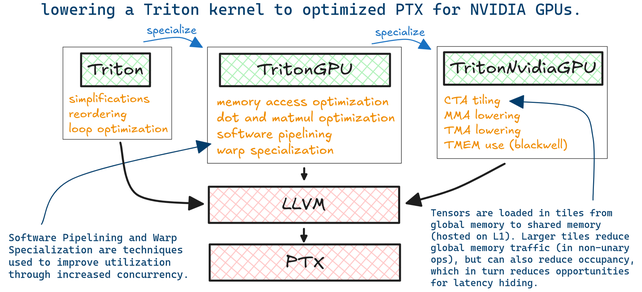
SM load/store instructions can easily saturate global memory bandwidth. Because of this, most kernels end up being memory bound. Various optimization techniques aim to optimize global memory traffic: kernel fusion, tiling and caching in shared memory, recomputation, global memory coalescing, vectorized loads and stores, ...etc.

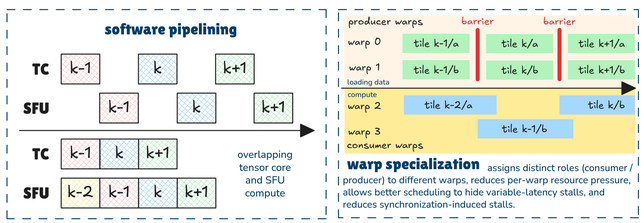
Principle 2: Optimize Latency Hiding.

An SM typically holds more resident warps than it can issue from, in one clock cycle, but many warps are stalled waiting on memory, dependencies, or execution resources. More resident warps means a higher probability that each of the schedulers can find an eligible warp to issue an instruction from, during every cycle. Techniques to improve latency hiding include: finding the optimum amount of resident warps*, and warp specialization.
*occupancy is constrained by other factors

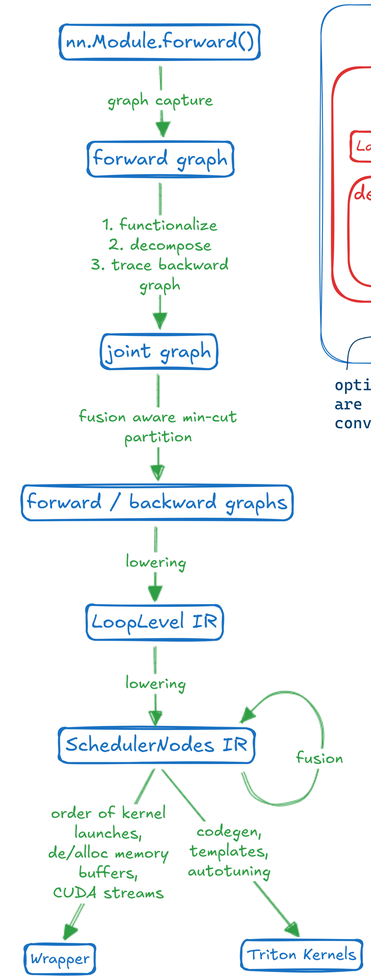
5 Triton Compilation

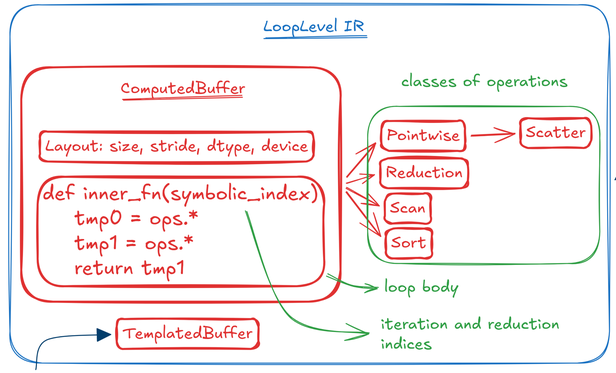
Triton includes three IR dialects which are responsible for lowering a Triton kernel to optimized PTX for NVIDIA GPUs.



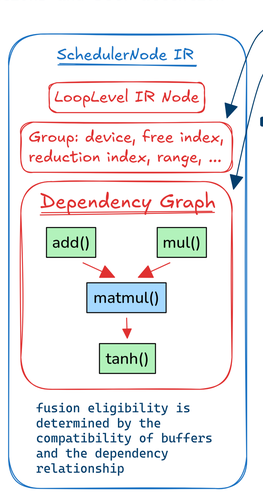


3 PyTorch 2.0 Compiler





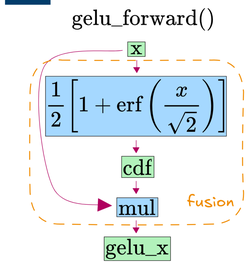
optimized handwritten kernel templates are provided for matrix multiplications, convolutions and self-attention

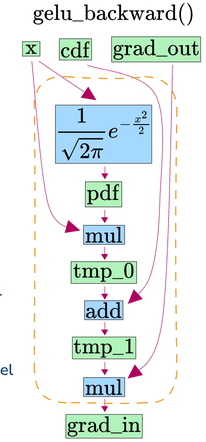


TorchInductor, the PyTorch 2.0 compiler backend includes two intermediate representations (IR). The LoopLevel IR describes how to map any operator to any parallel processor architecture. Operators are represented as nested loops. The indexing describes the access pattern, and the per-thread operations are represented using a set of virtual operators (ops.*), which can be later overloaded to cater to the task at hand e.g. code generation or analysis.

LoopLevel IR is lowered further into SchedulerNodes to determine scheduling. This primarily involves, first kernel fusion and afterwards determining the memory buffer handling and kernel execution strategy. Intra-kernel scheduling is handled by Triton during code generation.

4 Kernel Fusion





Kernel Fusion is the most consequential tensor compiler optimization, operating based on the first principle of GPU programming. Fusing kernels together allows the output of the preceding kernel to be directly used in the succeeding kernel without a round trip to global memory.

GELU	Separate	Fused
without recomputation	10 reads 6 writes	4 reads 3 writes
with recomputation	11 reads 7 writes	3 reads 2 writes

There is a symbiotic relationship between gradient checkpointing (recomputation) and operator fusion. Recomputation reduces both peak memory usage and also global memory bandwidth!

7 References

[1] Ansel, Jason, et al. "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation." Proceedings of the 29th ACM international conference on architectural support for programming languages and operating systems, volume 2. 2024.

[2] He, Horace, and Shangdi Yu. "Transcending runtime-memory tradeoffs in checkpointing by being fusion aware." Proceedings of Machine Learning and Systems 5 (2023): 414-427.

[3] Vasilache, Nicolas, et al. "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions." arXiv preprint arXiv:1802.04730 (2018).

[4] Kjolstad, Fredrik, et al. "The tensor algebra compiler." Proceedings of the ACM on Programming Languages 1.OOPSLA (2017): 1-29.

[5] Paul, Sayak, Animesh Jain, and Benjamin Bossan. "torch.compile and Diffusers: A Hands-On Guide to Peak Performance." PyTorch Blog, July 17, 2025

[6] Vellaisamy, Prabhu, et al. "Characterizing and optimizing llm inference workloads on cpu-gpu coupled architectures." 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2025.

Contact: p.mifsud@fz-juelich.de