Block Based GPU Programming with Triton – Phil Tillet
Speaker Q&A

Question: What are some of the challenges of building kernels to train models at OpenAI scales that surprised you?
Answer: One of the biggest challenges is ensuring code reliability and agility to avoid bugs that could stall large training jobs. Triton's advantage is that more people can look at the codebase to find bugs. Additionally, being able to quickly (e.g., within a week) write kernels for new research breakthroughs is valuable at OpenAI's scale.

Question: Have you noticed any tricks that are particularly important in Triton to achieving strong performance on consumer GPUs like the RTX 3090 or RTX 4090, which differ substantially from typical hardware you might be running in CI?
Answer: Consumer GPUs have different performance profiles than server GPUs, particularly regarding Tensor Core restrictions. On consumer GPUs, it's important to go out of your way to use 16-bit Tensor Cores, even if it means periodic renormalization, to get the best performance. This is something that hasn't been prioritized for Triton since it's mainly used on server GPUs.

Question: How do you ensure or detect or maintain the consistency of models?
Answer: To ensure consistency when optimizing kernels, the key is to check for bitwise identical output when the optimization should not change the output. For quantized types like FP16, testing with power-of-two inputs and ensuring no rounding errors can help verify bitwise equality with an FP32 reference implementation.

Question: For a company that wants to fine-tune a model or train a model from scratch, what would be the different tools you should start with?
Answer: Triton itself is a language for writing kernels, not a framework. To fine-tune or train a model, you would need a framework on top of Triton that can handle backpropagation and orchestration of GPU work, which is out of the scope of Triton.

Question: Have you seen any surprising kernels that people build on top of Triton?
Answer: One surprising kernel implementation was sorting, which the speaker didn't think was possible in Triton without custom instructions. However, someone found a way to implement it using hypercubes or similar techniques, which surprised the speaker as it demonstrated people becoming better at writing Triton than him.

Question: What did you land on the block base and how did you choose the block size?
Answer: Historically, Triton evolved from a code generator for matrix multiplication that generated blocks of PTX (virtual assembly for NVIDIA GPUs). As obstructions were added, it started resembling a compiler, leading to the development of Triton. To choose

the block size, the speaker typically auto-tunes different block sizes and picks the best one, or selects the largest block size that fits on the GPU without spilling registers.