A Crash Course on GPU Optimization – Mark Saroufim
Speaker Q&A

Question: If I have an open-source model and access to GPUs, what are the first steps I should follow when I want to optimize it? I don't imagine writing custom kernels should be the first step.
Answer: You're right, writing custom kernels shouldn't be the first step. The order you should follow is roughly:
1. Fuse operations using torch.compile
2. Enable tensor cores with torch.set_float32_matmul_precision("high")
3. Reduce overhead with torch.compile(model, mode="reduce-overhead") The first three steps won't take more than 5 minutes and are semantics-preserving. Quantization and custom kernels are more advanced and require more work since they can impact accuracy.

Question: How does PyTorch treat GPUs outside of NVIDIA, like AMD GPUs? Is there any lack of important automatic techniques that haven't been addressed?
Answer: PyTorch can work perfectly fine on AMD devices with just the .cuda() calls. This is mainly thanks to the Triton team, which has provided multiple backends to run the generated Triton kernels on Intel and AMD GPUs. The main weakness in PyTorch currently is around quantization, specifically supporting sub-byte data types due to legacy reasons and the autograd system. This is an area Marc is personally working on.

Question: There's a new framework called ExecutorTorch coming out. How is it different from torch.compile?
Answer: ExecutorTorch is more focused on running PyTorch models on constrained devices like mobile, where power and binary size are critical considerations. However, there is an intersection with torch.compile. Torch.compile has two major subsystems: Dynamo (captures the graph to run) and Inductor (generates Triton kernels). For mobile deployments, you'd capture the graph with Dynamo, but the runtime would be ExecutorTorch instead of Triton.

Question: What do you think about 1-bit quantization?
Answer: There is a working group in CUDA MODE exploring 1-bit quantization. The main challenge is running evaluations at scale because these extremely quantized models tend to be very slow. Marc is optimistic about using torch.compile to fuse bit-packing operations, which could accelerate these models. There has already been a contributor working on this for FP6, and the plan is to extend it to 1-bit quantization.

Question: What are some common pitfalls or mistakes you see people make when they try to optimize on GPUs?
Answer: For eager execution:
- Optimizing the wrong thing, spending too much time on operations that are an insignificant portion of the total runtime without profiling first.

For compiled mode:

- Not enabling the basic techniques like fusion, tensor cores, and overhead reduction, which can provide big gains with little effort.

The main mistake is not profiling and optimizing things arbitrarily instead of focusing on the bottlenecks identified through profiling.

Question: Are there tools to migrate CUDA code to Triton?
Answer: Generically no this is hard - in practice chatgpt is decent

Question: Do you have any suggestions for testing the precision of a custom triton kernel? They usually have some slight diff than pytorch impl. What is the best practice to set rtol and atol in torch.allclose?
Answer: It depends i think you might find some useful baselines in the inductor test utils (https://github.com/pytorch/pytorch/blob/main/test/inductor/test_fused_attention.py#L39)

Question: What about support of torch.compile on Apple silicon?
Answer: Nicest thing for us would be supporting Triton with an Metal backend but not sure what plans do the MPS/MPS Graph folks have. They are pretty secretive about it.

Question: Are there are key paramters you would typically start out using to benchmark performance for a PoC deep learning application in terms of memory and the number of cores / tensor cores? From my research 12GB memory may be a good balance of cost to speed, but curious what you think.
Answer: I wrote this which might help (https://docs.google.com/presentation/d/1IRsttm-FNTV6efX3EcVs8hZTEnTAXWUk/edit)

Question: How do you factor in non-determinism ? I mean, do you use a particular technique to uncover / handle such issues ?
Answer: Determinism has a performance implication At a high level take a simple example like a reduction a + (b + c) != (a + b) + c so you can't parallelize as easily if you insist on determinsim So unless I get a complete accuracy degradation i tend not to worry about it
Question: Anyone here interested, or better looking at MXFP8/6/4 formats?
Answer: We're adding them in ao (https://github.com/pytorch/ao/pull/264)

Question: Can you please elaborate on debugging pytorch tools at system level?
Answer: PyTorch profiler is great

Question: Is there a good framework for understanding what causes graph breaks and what doesn’t on dynamo? My main issues so far has been around conditional logic
Answer: TORCH_LOGS="graph_breaks" and just remove them one by one

Question: Is it theoretically possible to fuse an entire LLM model? How feasible is it?
Answer: Its more out there but there are persistent kernels where you could fuse an entire model I think its possible in principle just not done yet

Question: if we fuse enough small cuda kernels in one and try to send a large kernel to cuda once, what if the kernel is large enough that cpu is taking too much time scattering small kernels and at the same time gpu is staying idle? Would something like speculative execution help?
Answer: Yeah so things will be slower the first time around when you're deciding what to fuse, that's the compilation time