A Crash Course on GPU Optimization – Mark Saroufim
Summary notes

1. Introduction:

- GPUs are expensive but valuable for accelerating machine learning models.
- The talk aims to provide techniques for getting the most out of GPUs (optimizing GPU utilization).
- Marc Saroufim is a PyTorch core developer at Meta and co-founder of the CUDA MODE Discord community.

2. PyTorch and GPU Acceleration:

- PyTorch is a tensor computation language with strong GPU acceleration.
- Using GPUs in PyTorch is as simple as calling `.cuda()` on tensors.
- Under the hood, PyTorch generates kernels for different devices, data types, and tensor layouts.
- The eager execution model in PyTorch (running line by line) can be less performant than a graph mode.

3. Pointwise Operations and CUDA Kernels:

- Pointwise operations (like ReLU) are common in deep learning models.
- PyTorch generates CUDA kernels for these operations, assigning threads to different memory locations.
- Thread scheduling strategies and taking advantage of the memory hierarchy are crucial for performance.

4. Techniques for GPU Optimization:

a. Fusing Operations:

i. Instead of launching multiple small kernels, fusing operations into a single kernel can reduce overhead.
ii. The `torch.compile` function is a fusion compiler that generates fused Triton kernels.

b. Using Tensor Cores:

i. Tensor Cores are special circuitry in NVIDIA GPUs for accelerating matrix operations.
ii. Enabling Tensor Cores in PyTorch is done by setting `torch.set_float32_matmul_precision("high")`.

c. **Reducing Overhead:**
   i. Overhead can come from dispatching kernels and data transfers between CPU and GPU.
   ii. Queueing up multiple kernels in a CUDA graph using `torch.compile(model, mode="reduce-overhead")` can reduce overhead.

d. **Quantization:**
   i. Quantization can help both compute-bound and memory-bound workloads by reducing the data type size.
   ii. Weight-only quantization (int8 weights, bf16 activations) is a technique used in GPT-Fast.
   iii. The quantization space is rapidly evolving, with interest in sub-byte data types like int3, int4, etc.

e. **Custom Kernels:**
   i. For compute-bound problems, custom CUDA/Triton kernels may be required for optimal performance.
   ii. Techniques like Flash Attention and Online Softmax exploit GPU memory hierarchy and efficient normalization.
   iii. Learning CUDA and writing custom kernels can be rewarding but requires effort.

5. Learning Resources:

- The "Programming Massively Parallel Processors" (PMPP) book is recommended for learning CUDA.
- The `torch.utils.cpp_extension.load_inline()` function simplifies writing and testing CUDA kernels within PyTorch.
- The NVIDIA Nsight Compute profiler (ncu) is a valuable tool for analyzing kernel performance.
- The CUDA MODE Discord community and lectures are great resources for learning and collaboration.

6. Concluding Thoughts:

- Performance optimization is important as demand for faster models continues to grow.
- While compilers excel at fusions, kernel authors are still needed for rewriting mathematical operations efficiently.
- Marc is personally focused on quantization and custom kernels in the AO repo and CUDA MODE community.