02:00:18 flash attention, 96ms

now torch compile is amazing but there are operations torch compile will not find and an amazing example of that is Flash attention to which we turn next so flash attention comes from this paper from uh Stanford in 2022 and it's this incredible algorithm for performing attention so um and running it a lot faster so flash attention will come here and we will take out these four lines and Flash attention implements these four lines really really quickly and how does it do that well flash attention is a kernel Fusion operation so you see here we have um in this diagram they're showing P torch and you have these four operations uh they're including Dropout but we are not using Dropout here so we just have these four lines of code here and instead of those we are fusing them into a single fused kernel of flash attention so it's an it's a it's a kernel

Fusion algorithm but it's a kernel Fusion that torch compile cannot find and the reason that it cannot find it is that it um requires an algorithmic rewrite of how attention is actually implemented here in this case and what's remarkable about it is that uh flash attention actually if you just count the number of flops flash attention does more flops than this attention here but flash attention is actually significantly faster in fact they site 7. six times faster potentially and that's because it is very mindful of the memory hierarchy as I described it just now and so it's very mindful about what's in high bandwidth memory what's in the shared memory and it is very careful with how it orchestrates the computation such that we have fewer reads and writes to the high bandwidth memory and so even though we're doing more flops the expensive part is they load and store into hbm and that's what they avoid and so in particular they do not

ever materialize this end byend attention Matrix this ATT here a flash attention is designed such that this Matrix never gets materialized at any point and it never gets read or written to the hbm and this is a very large Matrix right so um because this is where all the queries and keys interact and we're sort of getting um for each head for each batch element we're getting a t BYT Matrix of attention which is a Million numbers even for a single head at a single batch index at like so so basically this is a ton of memory and and this is never materialized and the way that this is achieved is that basically the fundamental algorithmic rewrite here relies on this online softmax trick which was proposed previously and I'll show you the paper in a bit and the online softmax trick coming from a previous paper um shows how you can incrementally evaluate a soft Max without having to sort of realize all of the inputs to

the softmax to do the normalization and you do that by having these intermediate variables M and L and there's an update to them that allows you to evaluate the softmax in an online manner um now flash attention actually so recently flash attention 2 came out as well so I have that paper up here as well uh that has additional gains to how it calculates flash attention and the original paper that this is based on basically is this online normalizer calculation for softmax and remarkably it came out of Nvidia and it came out of it like really early 2018 so this is 4 years before flash attention and this paper says that we propose a way to compute the classical softmax with fewer memory accesses and hypothesize that this reduction in memory accesses should improve softmax performance on actual hardware and so they are extremely correct in this hypothesis but it's really fascinating to me that they're from Nvidia and that they

had this realization but they didn't actually take it to the actual flash attention that had to come four years later from Stanford so I don't fully understand the historical how this happened historically um but they do basically propose this online update to the softmax uh right here and this is fundamentally what they reuse here to calculate the softmax in a streaming Manner and then they realize they can actually fuse all the other operations with the online sofx calculation into a single fused kernel flash attention and that's what we are about to use so great example I think of being aware of um memory hierarchy the fact that flops don't matter uh the entire memory access pattern matters and that torch compile is amazing but there are many optimizations that are still available to us that potentially torch compile cannot find maybe maybe one day it could but right now it seems like a lot to ask so

here's what we're going to do we're going to use Flash attention and the way to do that basically in pytorch is we are going to comment out these four lines and we're going to replace them with a single line and here we are calling this compound operation in pytorch called scale that product attention and uh pytorch will call flash attention when you use it in this way I'm not actually 100% sure why torch compile doesn't realize that these four lines should just call flash attention in this exact way we have to do it again for it which in my opinion is a little bit odd but um here we are so you have to use this compound up and uh let's wait for a few moments before torch comp compile gets around to it and then let's remember that we achieved 6.05 661 I have it here that's the loss we were expecting to see and we took 130 milliseconds uh before this change so we're expecting to see the exact same result by

iteration 49 but we expect to see faster runtime because Flash attention is just a an algorithmic rewrite and it's a faster kernel but it doesn't actually change any of the computation and we should have the exact same optimization so okay so we're a lot faster we're at about 95 milliseconds and we achiev 6.58 okay so they're basically identical up to a floating Point fudge Factor so it's the identical computation but it's significantly faster going from 130 to roughly 90 96 and so this is um 96 divide 130ish so this is maybe 27 is% Improvement um so uh really interesting and that is Flash retention okay we are