Lecture 1

Wednesday, 21 February 2024 11:28 AM

CUDA is async. So python time module will only measure the time it took to launch the kernel, not the actual runtime.

What is a kernel though? I thought OS had a kernel which is like the main program and then to control the processes, there is shell. Is this the same as shell?

From my understanding, a CUDA kernel is just a function that is run on GPU by multiple threads all at once

Profiling a kernel basically means to collect data w.r.t various processes inside GPU and identifying the bottlenecks

Before actually profiling, always do a warmup because first time cuda context initializes

Then you need to do cuda.synchronize at the end "because CUDA is async"

So what?

↳ Basically a way to tell the CPU that the process is done since it won't know (because the process is async)

Some insights on square kernel profiling:

1) Torch square func" uses the power

2) Power is slightly less optimized than multiplication.

Pytorch profiler gives a json

Memcpy is a CUDA command which is run when you copy tensors on GPU (using .cuda())

How to use a cuda kernel with pytorch? (since cuda is in C/C++)

→ Either create python bindings for C++

no idea what that is

→ Use a load_inline function (pass the func" as a string)

↳ It uses a pybind under the hood

↳ It will also create a build.ninja file. Which is like a build file in C++.

So a pytorch square has a CUDA kernel and a wrapper around it to return a tensor object compatible with C++

Building torch from source takes time because each CUDA kernel needs to be compiled

The load inline method also caches

We can also use numba. You pass two arguments: threads per block and blocks per grid.

CUDA combines threads into blocks to perform computation

Blocks are further organised into grids. There can be 1, 2, or 3 D grids.

Each block performs operations independently

Numba also writes a CUDA kernel under the hood.

We can also run code on GPU using triton.

Triton is a DSL (Domain specific language)

↳ Triton is used in pytorch 2.0 which uses a new compiler called Inductor. This directly converts pytorch code to PTX (Parallel Thread Execution) or PTX which is Nvidia's low level assembly language.

should also look at the generated assembly code to see whats happening.

one trick to write triton kernels is to use torch.compile() with the TORCH_LOGS="output_code" flag.

↳ And then optimise the code from here.

Another important optimization is fusion (to have more and more code in a single kernel)

The best CUDA profiler is the NCU profiler

Nvidia compute Profiler.

some tricks to improve your kernel performance:

→ Tail effect + achieved occupancy 70%. try padding (we can control)

→ long scoreboards stalls: coalesced use shared memory (controlled by triton)

Torch.compile also does fusions automatically in triton kernels