Fused Gelu

Gelu(Gaussian Error Linear Unit) is an activation function commonly used in Transformers. It is based on the error function (erf) and can be approximated with a fast polynomial or tanh-based formula.

We fused Gelu with GEMM to test fused kernels, where the matrix multiplication and activation are computed in a single GPU kernel, keeping the data local. This approach reduces memory traffic and kernel launch overhead, improving performance compared to executing GEMM and GELU separately.

To evaluate performance, we measure TFLOPs. Since Gemm dominates computation, TFLOPs are dictated by matrix multiplication. The fused GELU adds negligible memory overhead, so it does not limit compute performance.

See the code here.

Triton (Tutorial Version):

We modified the triton MatMul tutorial to use Gelu as the activation. The main changes are:

1. Bias addition
2. Gelu activation

The operation computed is:

$$Output = GELU(AxB + bias)$$

The kernel implementation : here

TorchInductor:

Just like with Matmul, and layernorm we used torch.compile for fused gelu- wrapping the pytorch function in a compiled version so we could benchmark it in default and max-autotune modes.

The kernels implementation can be found here.

Kernel LLM

We used this prompt, with the same flags as in Matmul. The same issues occurred: grid specification and tensor data type, which we fixed by explicitly setting the grid and using bfloat16. We generalized the kernels for different shapes but did not attempt autotuning.

The finalized kernel we used can be found here.

Makora Generate

We reused the same prompt from KernelLLM, again Mako handled prompt validation, kernel generation, compilation and benchmarking automatically, making it very easy to use. However, the generated kernel used the tanh approximation of GELU. By explicitly specifying $torch.nn.GELU(approximate = 'none')$ in the prompt solved the issue, ensuring that the generated kernel used the erf function instead of tanh.

The finalized kernel we used can be found here.

Helion

For the fused GEMM+Bias+GELU kernel, Helion is a natural fit because the code inside the tiled part becomes a single device kernel, so bias-add and GELU stay fused with the matmul tile accumulation. In our gemm_gelu() kernel, we do addmm into an FP32 accumulator, then apply bias and F.gelu(...) before writing the output tile.[x]