PyTorch Fuser

The fuser accepts subgraphs wrapped in “fusion nodes” and tries to execute them by just-in-time (JIT) compiling kernels that run all the graph operations.

Code Organization

The fuser is designed hierarchically with device-independent logic eventually deferring to device-specific logic and implementation. The device-specific code is (mostly) found in each devices’ subdirectory. The device-independent logic has six components:

- The Interface (interface.h/cpp) has functions to register and run fusions, interrogate fusion functionality, and perform debugging.
- The Compiler (compiler.h/cpp) performs “upfront” and “runtime” compilation. When fusions are registered, upfront compilation produces fallback code and and performs some shape inference. When a fusion is run, runtime compilation invokes code generation and the device-specific compilation logic.
- The Code Generator (codegen.h/cpp) produces the string to be compiled on the device.
- The Executor (executor.h/cpp) runs requested fusions. It performs shape inference, expands tensors as necessary, determines the device to run on, acquires a cached compiled kernel or requests the Compiler produce a new one, invokes device-specific code to launch the kernel and updates the stack.
- The Fallback (fallback.h/cpp) runs subgraphs that can’t be fused because shape inference didn’t determine a common tensor size or the device the tensors are on doesn’t support fusion.
- The Kernel Specification Cache (kernel_cache.h/cpp) is a thread-safe cache holding the device-independent specifications produced during upfront compilation. These specifications each have their own thread-safe stores of compiled kernels that the Executor checks before requesting runtime compilation.

The device-specific components have logic for compiling and running code in FusedKernelCPU (cpu/fused_kernel.h/cpp) and FusedKernelCUDA (cuda/fused_kernel.h/cpp).