Software Architecture for c10

C10 is currently in development and will be the core library behind PyTorch. It will contain a tensor class and a dispatcher to run tensor operations and dispatch them to different devices. It is intentionally kept minimal and has only very few operations. Most tensor operations will not be part of c10, but will be in extension libraries that can be loaded and registered with c10.

Module Structure

- c10
  - util
  - core
  - backend
    - cpu
    - cuda
    - hip
    - my_accelerator

This is the top level module structure. Each module gets a separate folder, namespace and build target. The build system enforces that there are no dependencies (i.e. linker symbols or header includes) in the wrong direction.

What Goes Where?

- c10/
  - Basically nothing, everything should be in subfolders
- c10/util/
  - Things that are general and not specific to our framework but could in theory be reused in other projects.
  - Features in here should be small and independent. While we're not likely to achieve this 100% (and will not enforce it), the ideal to strive for would be that a header in c10/util is self-contained and doesn't include any other headers, also no others from c10/util.
    - It is important that headers here do **not** include headers from other locations such as c10/core.
  - Examples: intrusive_ptr, ArrayRef, optional
- c10/core/
  - Things that are specific to our framework and can likely not be reused in other projects. Note: Tensor might be reusable in other ML frameworks (well, in theory at least), but not generally in other projects, so it's not a util but belongs to core.
  - These things must be needed in all builds of the framework, i.e. CUDA bindings or additional operators will not be part of core. (note: Some operator schemas will be, because they're offered as methods on Tensor. The kernels, however, will not be in core.)
  - Examples: Tensor, Allocator
  - core is further divided into sub folders, e.g. core/dispatch or core/opschema
- c10/backend/xxx
  - Backend-specific TensorImpls, allocator implementations, and other backend-specific code. Possibly also kernels for the operations whose schemas are in c10/core/.
  - There's modules for backend/cpu, backend/cuda, backend/hip, backend/my_accelerator

Namespaces

Namespace mirror module structure, i.e. there is going to be `c10::core` and `c10::util` namespaces. Each namespace has an `impl` sub-namespace for private code whose identifiers should never appear in third party code. Backends are going to live in `c10::cpu`, `c10::hip`, `c10::my_accelerator`, there will not be a `c10::backend::cpu`.

- c10
  - core
    - impl
  - util
    - impl
  - cpu
    - impl
  - cuda
    - impl
  - hip
    - impl
  - my_accelerator
    - impl

Even though the modules can have further sub-folders (say c10/core/impl/dispatch), these will usually not get a separate namespace but live in the namespace for their module. This is, however, not a strict rule - there might be exceptions to this.

Identifiers that are part of the public API will be imported to the top level `c10` namespace with a using statement, i.e.

```
namespace c10 { using ArrayRef = util::ArrayRef; }
```

This using will happen in the header that declares the file, i.e. in `c10/util/ArrayRef.h`.

Build Targets

- c10-util: everything in c10/util/...
- c10-core: everything in c10/core/...
- c10-cpu: everything in c10/backend/cpu/...
- c10-cuda, c10-hip, c10-my_accelerator accordingly

Even though the modules can have further sub-folders (say c10/core/dispatch), these will not be in a separate build target, but be built together with the rest of their module.

If it turns out to be too hard to set up CMake for these build targets (especially in regards to the transitively-linked symbol issue for dynamic libraries), we might reconsider and merge build targets together. c10-cuda will still have to stay separate.

Detailed Folder Structure

Each module has an `impl` subfolder for the code in the impl namespace (see namespace section for definition of this). We unfortunately will need to allow public headers to include private headers, because we can't cleanly split that yet, but private code means any symbols or identifiers defined in here should never appear in third party code.

Each module has a `test` subfolder for unit tests. The folder structure inside the test folder will mirror the folder structure of the source, i.e. `c10/core/impl/TensorImpl.h` is tested in `c10/core/test/impl/TensorImpl.h`.

Further subfolders inside a module are encouraged if there's a set of files that together builds a functional unit, for example c10/core/impl/dispatch. These subfolder will, however, not be a separate build target and in most cases also not be in a separate namespace from the rest of the module.

There will be a top level `c10.h` file that includes the whole public API. Third parties (i.e. non-fb developers) should only ever include this and no other headers, allowing us to move things around.

So the actual, detailed folder structure will look more like this:

- c10
  - c10.h file (third parties should **only** include this)
  - util
    - impl
    - test
      - impl
  - core
    - impl
      - dispatch
      - opschema
    - test
      - impl
        - dispatch
        - opschema
  - backend
    - cpu
      - impl
      - test
        - impl
    - cuda
      - impl
      - test
        - impl
    - hip
      - impl
      - test
        - impl
    - my_accelerator
      - impl
      - test
        - impl