Custom CUDA and Python

Compiled vs Interpreted

- Compiled languages are translated into machine code directly by compilers.
  - C/C++, Java, CUDA
  - executables
- Interpreted languages are read by the interpreter program and translated
  - Python, Javascript
  - scripts

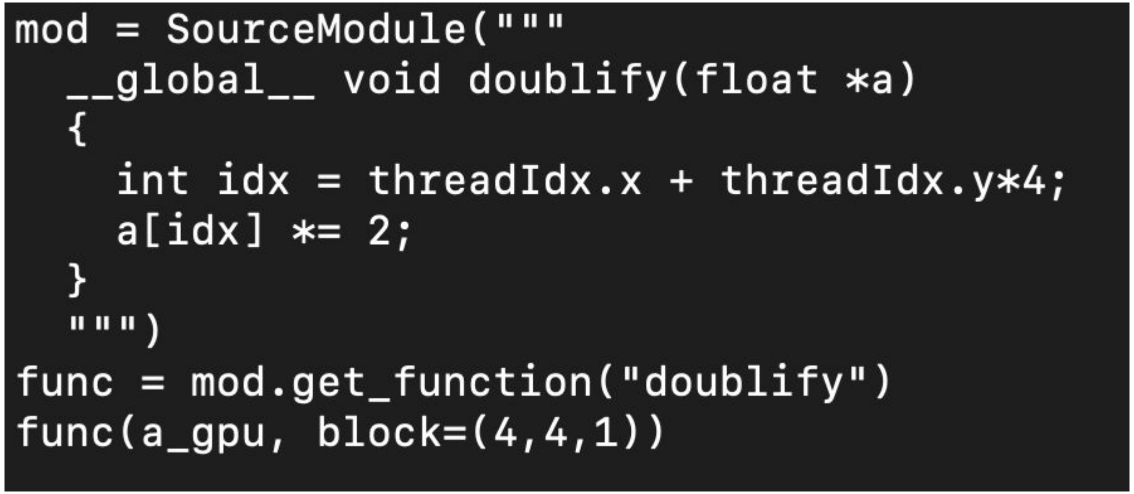
- In order to use compiled code in an interpreted language, we need to give the interpreter program the “translation”.
  - Just-in-time compilation: PyCUDA, PyTorch JIT
  - Python binding: PyBind11, Cython

The AI developers

- Data scientists: (the drivers)
    - The end user.
    - High-level Python code and AI frameworks
- Developers/Researchers: (car mechanics/designers)
    - Writes AI frameworks, builds novel prototype models
    - Python and C++/CUDA
    - Python bindings
- Performance engineers: (engine designers/manufacturers)
    - Writes optimized kernels
    - C++/CUDA

Option: Just-in-time (JIT) compilation. PyCUDA example.

- C/C++/CUDA code is compiled on the fly as the Python code is executed.

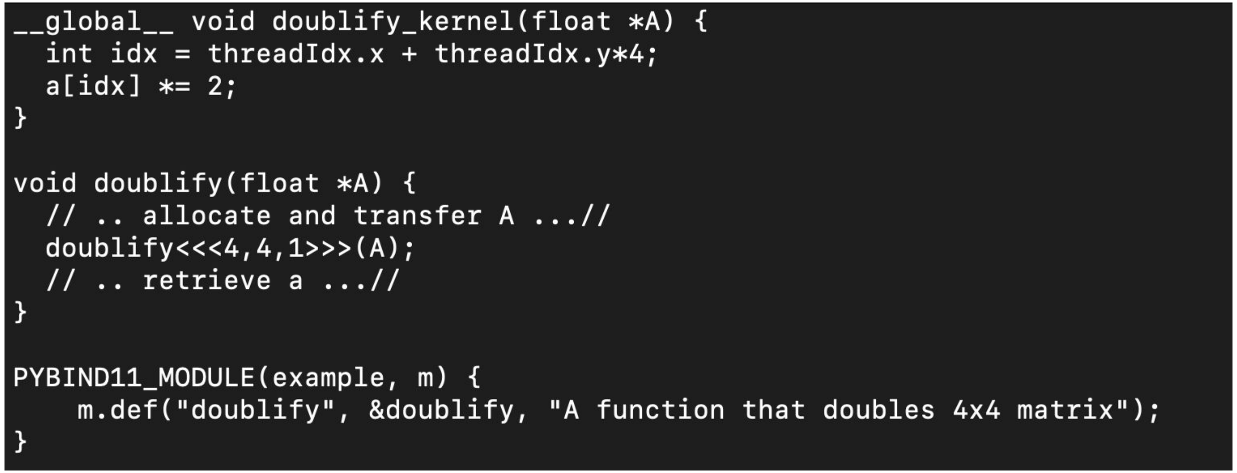


```
mod = SourceModule("""
  __global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
  """)
func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))
```

- User system must compile at runtime.
  - Must have the correct environment and flag sets for compilation.
  - May take some time
- May not always be supported

Option: Python bindings. PyBind11 example.

- C/C++/CUDA code is compiled ahead of time, and bound to a python method.



```
__global__ void doublify_kernel(float *A) {
  int idx = threadIdx.x + threadIdx.y*4;
  a[idx] *= 2;
}

void doublify(float *A) {
  // .. allocate and transfer A ...//
  doublify<<<4,4,1>>>(A);
  // .. retrieve a ...//
}

PYBIND11_MODULE(example, m) {
    m.def("doublify", &doublify, "A function that doubles 4x4 matrix");
}
```

- Less flexible; only supports what the developer compiled.
- Limited support for CUDA (no setuptools support)
  - Couple poorly maintained 3rd party repo, or util as part of frameworks (PyTorch)

Takeaways

- Python/CUDA interfacing needed in AI
  - AI end users prefer Python, whereas CUDA and GPU developers favor C/C++
  - Framework developers and researchers use Python binding or JIT compilation bridge the gap.
- JIT compilation
  - Example: PyCUDA
  - More flexible, but require correct environment, compilation time, and often C/C++ knowledge.
  - Is not always supported.
- Python bindings
  - Example: PyBind11
  - Less flexible, but developer controls the compilation and usually no C/C++ knowledge required.
  - Less official support.