

Interfacing C++/CUDA with Python

Casper O. da Costa-Luis [github/casperdcl](#)

2025-04-07 [video](#) | [html](#) | [pdf](#)



announcement

website



chat

discord

History

Context

ctypes

<https://docs.python.org/3/library/ctypes.html>

- ▶ *Foreign Function Interface* (FFI)

ctypes

<https://docs.python.org/3/library/ctypes.html>

- ▶ *Foreign Function Interface* (FFI)
- ▶ Hack to call C/C++ functions from Python

ctypes

<https://docs.python.org/3/library/ctypes.html>

- ▶ *Foreign Function Interface* (FFI)
- ▶ Hack to call C/C++ functions from Python
- ▶ e.g. https://github.com/gschramm/square_array

ctypes example

```
import ctypes
lib = ctypes.CDLL("./libmy_experiment.so")

# (re)define function signature for `void myfunc(float*, size_t)`
lib.myfunc.argtypes = [ctypes.c_void_p, ctypes.c_size_t]
lib.myfunc.restype = None

input_array = np.array([1, 2, 3], dtype=np.float32)
# call function
lib.myfunc(
    input_array.ctypes.data_as(ctypes.POINTER(ctypes.c_float)),
    ctypes.c_size_t(len(input_array)))
```

CPython API

<https://docs.python.org/3/c-api/intro.html>

- ▶ “correct” way to call C/C++ functions from Python

CPython API example

```
#include <Python.h>
#include <numpy/arrayobject.h>

static PyObject *myfunc(PyObject *self, PyObject *args) {
    PyObject *arr = NULL;
    if (!PyArg_ParseTuple(args, "O", &arr)) return NULL;
    PyArrayObject *np_arr = PyArray_FROM_OTF(
        arr, NPY_FLOAT32, NPY_ARRAY_INOUT_ARRAY);
    float *arr_ptr = PyArray_DATA(np_arr);
    npy_intp *size = PyArray_SHAPE(np_arr);
    for (size_t i = 0; i < size[0]; ++i) arr_ptr[i] *= 2;
}
```

```
static struct PyModuleDef my_module = {
    PyModuleDef_HEAD_INIT,
    .m_name = "my_experiment"
};

static PyMethodDef my_methods[] = {
    {"myfunc", myfunc, METH_VARARGS, "In-place modifies ndarray[float]"},
    {NULL, NULL, 0, NULL} // Sentinel
};

PyMODINIT_FUNC PyInit_spam(void) {
    import_array();
    return PyModule_Create(&my_module);
}
```

with my_experiment.{so,dll} in PYTHONPATH:

```
>>> import my_experiment
```

```
>>> help(my_experiment.myfunc)
```

Help on built-in function myfunc in my_experiment:

```
my_experiment.myfunc = myfunc(...)
```

```
    In-place modifies ndarray[float]
```

Buffer protocol

<https://docs.python.org/3/c-api/buffer.html>

- ▶ Subset of CPython API

Buffer protocol

<https://docs.python.org/3/c-api/buffer.html>

- ▶ Subset of CPython API
- ▶ Standard struct to expose arrays

```
float *data
```

```
size_t ndim
```

```
size_t shape[]
```

[https:](https://github.com/AMYPAD/CuVec/blob/main/cuvec/include/cuvec_cpython.cuh#L128)

[//github.com/AMYPAD/CuVec/blob/main/cuvec/include/cuvec_cpython.cuh#L128](https://github.com/AMYPAD/CuVec/blob/main/cuvec/include/cuvec_cpython.cuh#L128)

```
int my_buffer(PyObject *obj, Py_buffer *view, int flags) {  
    view->buf = (void *)MY_GET_ARR_PTR(obj);  
    view->obj = obj;  
    view->len = MY_GET_ARR_SIZE(obj) * sizeof(float);  
    view->readonly = 0;  
    view->itemsize = sizeof(float);  
    view->format = "f";  
    view->ndim = MY_GET_ARR_NDIM(obj);  
    view->shape = MY_GET_ARR_SHAPE(obj);  
    view->strides = MY_GET_ARR_STRIDES(obj);  
    view->suboffsets = NULL; view->internal = NULL;  
    Py_INCREF(view->obj);  
    return 0;  
}
```

```
static PyObject *myfunc(PyObject *self, PyObject *args) {  
    Py_buffer *view = NULL;  
    if (!PyArg_ParseTuple(args, "w*", &view)) return NULL;  
    float *arr_ptr = view->buf;  
    for (size_t i = 0; i < view->shape[0]; ++i) arr_ptr[i] *= 2;  
    PyBuffer_Release(view);  
}
```

C++ wrappers (pybind11, etc)

<https://pybind11.readthedocs.io/en/stable/>

- ▶ C++ templates reduce boilerplate

```
#include <pybind11/pybind11.h>

void myfunc(pybind11::buffer view) {
    pybind11::buffer_info arr = view.request();
    float *ptr = arr.ptr;
    if (arr.ndim != 1) throw std::runtime_error("expected 1D array");
    for (size_t i = 0; i < arr.size; ++i) ptr[i] *= 2;
}

using namespace pybind11::literals;
PYBIND11_MODULE(my_examples, m){
    m.def("myfunc", &myfunc, "input_array"_a,
          "In-place modifies ndarray[float]");
}
```

Python interfaces

`__cuda_array_interface__`

https://numba.readthedocs.io/en/stable/cuda/cuda_array_interface.html

- ▶ Python-level equivalent of C buffer protocol

Python interfaces

`__cuda_array_interface__`

https://numba.readthedocs.io/en/stable/cuda/cuda_array_interface.html

- ▶ Python-level equivalent of C buffer protocol
- ▶ IMO a bad idea

```
@property
```

```
def __cuda_array_interface__(self) -> Dict[str, Any]:
```

```
    return {'shape': self.shape, 'typestr': numpy.dtype(self.typechar).str  
            'data': (self.get_memory_address(), self.read_only), 'version'
```

`__dlpack__`

[https://data-apis.org/array-](https://data-apis.org/array-api/latest/API_specification/generated/array_api.array.__dlpack__.html)

[api/latest/API_specification/generated/array_api.array.__dlpack__.html](https://data-apis.org/array-api/latest/API_specification/generated/array_api.array.__dlpack__.html)

https://dmlc.github.io/dlpack/latest/python_spec.html#reference-implementations

- ▶ Similar to `__cuda_array_interface__` but also handles ownership & multiple devices/streams

```
@property
def __dlpack__(self, copy=False, max_version: tuple[int]=None,
               stream: int=None, dl_device: tuple[int]=None
):
    dl_tensor = DLTensor(data=self.addr, device=dl_device, ndim=self.ndim,
                        dtype=DLDataType.from_dtype(self.dtype),
                        shape=ctypes.cast(self.shape, ctypes.POINTER(ctypes.c_int64)),
                        strides=None, byte_offset=0)

    managed_tensor = DLManagedTensor(dl_tensor=dl_tensor, manager_ctx=0,
                                       deleter=DLTensorDeleter(lambda addr: None))
    return pythonapi.PyCapsule_New(ctypes.byref(managed_tensor), b'dltensor')
```

CUDA Unified Memory

Host (CPU) vs Device (GPU)

```
#include <cuda_runtime.h>
int N = ...;
float cpu_data[N];
for (int i = 0; i < N; ++i) data[i] = ...;

float *gpu_data;
cudaMalloc(&gpu_data, sizeof(float) * N);
cudaMemcpy(gpu_data, cpu_data, sizeof(float) * N, cudaMemcpyHostToDevice);
mykernel<<<1, N>>>(gpu_data, N);
cudaDeviceSynchronize();
cudaMemcpy(cpu_data, gpu_data, sizeof(float) * N, cudaMemcpyDeviceToHost);
cudaDeviceSynchronize();
cudaFree(gpu_data);
```

Unified Memory

```
#include <cuda_runtime.h>
int N = ...;
float *data;
cudaMallocManaged(&data, sizeof(float) * N);
for (int i = 0; i < N; ++i) data[i] = ...;

mykernel<<<1, N>>>(data, N);
cudaDeviceSynchronize();
```

CuVec

<https://amypad.github.io/CuVec/>

- ▶ `std::vector<T, {malloc, free}> → std::vector<T, {cudaMallocManaged, cudaFree}>`

CuVec

<https://amypad.github.io/CuVec/>

- ▶ `std::vector<T, {malloc, free}> → std::vector<T, {cudaMallocManaged, cudaFree}>`
 - ▶ `std::vector<T>::data()`

CuVec

<https://amypad.github.io/CuVec/>

- ▶ `std::vector<T, {malloc, free}> → std::vector<T, {cudaMallocManaged, cudaFree}>`
 - ▶ `std::vector<T>::data()`
 - ▶ `std::vector<T>::size()`

CuVec

<https://amypad.github.io/CuVec/>

- ▶ `std::vector<T, {malloc, free}> → std::vector<T, {cudaMallocManaged, cudaFree}>`
 - ▶ `std::vector<T>::data()`
 - ▶ `std::vector<T>::size()`
 - ▶ `std::vector<T>::resize()`

CuVec in Python

- ▶ Buffer protocol to expose to Python

CuVec in Python

- ▶ Buffer protocol to expose to Python
- ▶ Inherit from `numpy.ndarray`

NiftyPET

Case Study

- ▶ NumPy arrays

Case Study

- ▶ NumPy arrays
 - ▶ CPython API

Case Study

- ▶ NumPy arrays
 - ▶ CPython API
 - ▶ CUDA: cudaMalloc, memcpyH2D, kernel, memcpyD2H, cudaFree

CuVec

- ▶ CuVec arrays
 - ▶ CPython API
 - ▶ CUDA: kernel, sync

NumCu

<https://amypad.github.io/NumCu/>

Minimal Python/C++/CUDA library using `CuVec`'s CPython `buffer protocol`.

Bonus

Packaging

► `pip install`

[github/casperdcl](#)

Packaging

- ▶ `pip install`
- ▶ `pyproject.toml::build-system`

[github/casperdcl](#)

Packaging

- ▶ `pip install`
- ▶ `pyproject.toml::build-system`
- ▶ `requires = [cmake, scikit-build-core, pybind11, ...]`

[github/casperdcl](https://github.com/casperdcl)

Packaging

- ▶ `pip install`
- ▶ `pyproject.toml::build-system`
- ▶ `requires = [cmake, scikit-build-core, pybind11, ...]`
- ▶ `cmake`

[github/casperdcl](https://github.com/casperdcl)

Packaging

- ▶ `pip install`
- ▶ `pyproject.toml::build-system`
- ▶ `requires = [cmake, scikit-build-core, pybind11, ...]`
- ▶ `cmake`
- ▶ `install *.{py,so,dll}`

[github/casperdcl](https://github.com/casperdcl)