# CuPy and CuPy-Xarray:

Presenter: Giacomo Capodaglio
Oct 14th 2025:
AMD @ CASTIEL

AMD
together we advance_

# What is CuPy

- NumPy is a Python interface to optimized routines written in C that provide arrays, multi-dimensional arrays and common numerical operations on them. These are much faster than operating on Python lists

- SciPy provides fundamental algorithms common in scientific and numerical computing. The underlying code is a mixture of Fortran, C and C++

- CuPy is a NumPy/SciPy-compatible array library for GPU-accelerated computing with Python. It is not an Nvidia product, despite the Nvidia sounding name.

- CuPy acts as a drop-in replacement to run existing NumPy/SciPy code on Nvidia CUDA or AMD ROCm™ platforms

- CuPy provides the N-dimensional array (`ndarray`), sparse matrices, and the associated routines for GPU devices, most having the same API as NumPy and SciPy

- CuPy provides interfaces to GPU optimized libraries such as rocBLAS, rocSPARSE, rocFFT, and RCCL

source: [cupy documentation]

# CuPy and HIP

- CuPy uses HIP as backhand to run on AMD GPUs
- **HIP**: **H**eterogeneous-compute **I**nterface for **P**ortability
  - C++ runtime API and kernel language
  - Works on AMD and Nvidia GPUs
- The CPU is often referred to as the host, and the GPU as the device
- In HIP, launching a kernel is **non-blocking for the host**
  - After sending instructions/data, the host continues to do more work while the device executes the kernel. This means GPU execution and CPU activity can overlap
- What it means for CuPy: appropriate synchronization calls have to be made after a kernel call:
  - `cupy.cuda.Device(0).synchronize()`
  - `cupy.cuda.Stream.synchronize()`
- In HIP, memory copies such as `hipMemcpy` is **blocking for the host**
  - All activity on the host stops until the copy has completed.
- What it means for CuPy: no need to sync if calling a memory copy right after a kernel.

[Public]

click **here** for differences between CuPy and NumPy

# CuPy functions

CuPy vs NumPy API

## Comparison Table #

Here is a list of NumPy / SciPy APIs and its corresponding CuPy implementations.

`-` in CuPy column denotes that CuPy implementation is not provided yet. We welcome contributions for these functions.

### NumPy / CuPy APIs

#### Module-Level

| NumPy | CuPy |
|---|---|
| `numpy.DataSource` | `cupy.DataSource` (*alias of* `numpy.DataSource`) |
| `numpy.ScalarType` | - |
| `numpy.abs` | `cupy.abs` |
| `numpy.absolute` | `cupy.absolute` |
| `numpy.add` | `cupy.add` |
| `numpy.all` | `cupy.all` |
| `numpy.allclose` | `cupy.allclose` |

full list here: cupy API vs numpy API

CuPy-specific functions

## CuPy-specific functions

CuPy-specific functions are placed under `cupyx` namespace.

| | |
|---|---|
| `cupyx.rsqrt` | Returns the reciprocal square root. |
| `cupyx.scatter_add`(a, slices, value) | Adds given values to specified elements of an array. |
| `cupyx.scatter_max`(a, slices, value) | Stores a maximum value of elements specified by indices to an array. |
| `cupyx.scatter_min`(a, slices, value) | Stores a minimum value of elements specified by indices to an array. |
| `cupyx.empty_pinned`(shape[, dtype, order]) | Returns a new, uninitialized NumPy array with the given shape and dtype. |
| `cupyx.empty_like_pinned`(a[, dtype, order, ...]) | Returns a new, uninitialized NumPy array with the same shape and dtype as those of the given array. |

full list here: cupy documentation

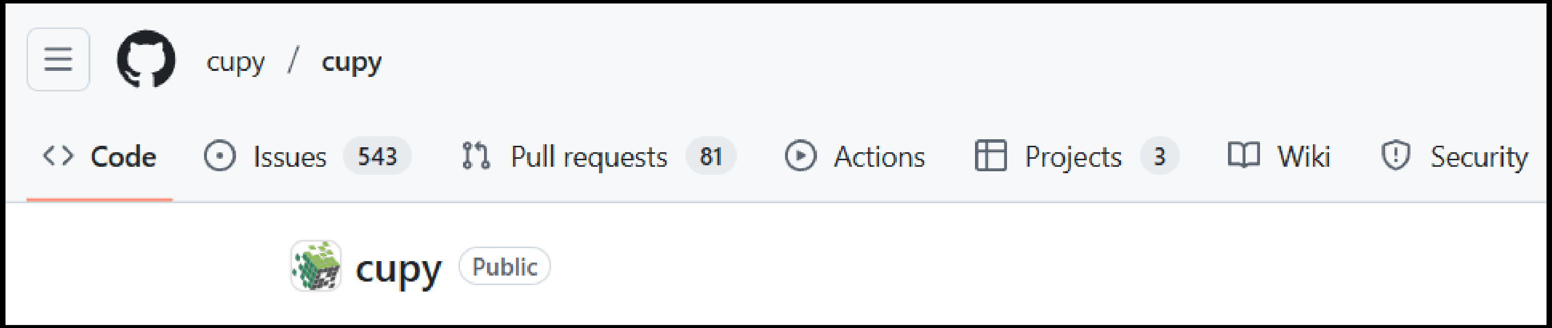


AMD
together we advance_

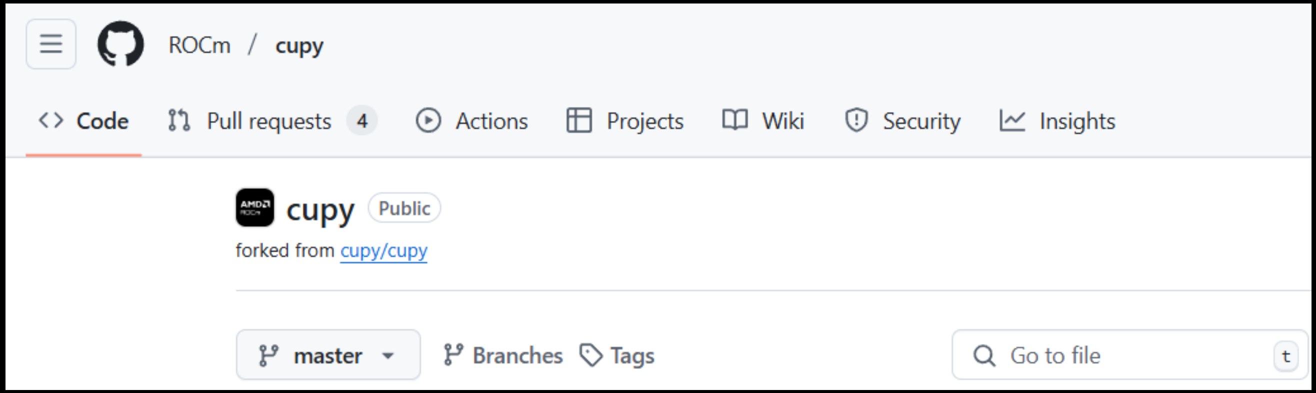
# CuPy Installation – GitHub Repos

- There are two GitHub repos to take the CuPy source code from to run on AMD GPUs

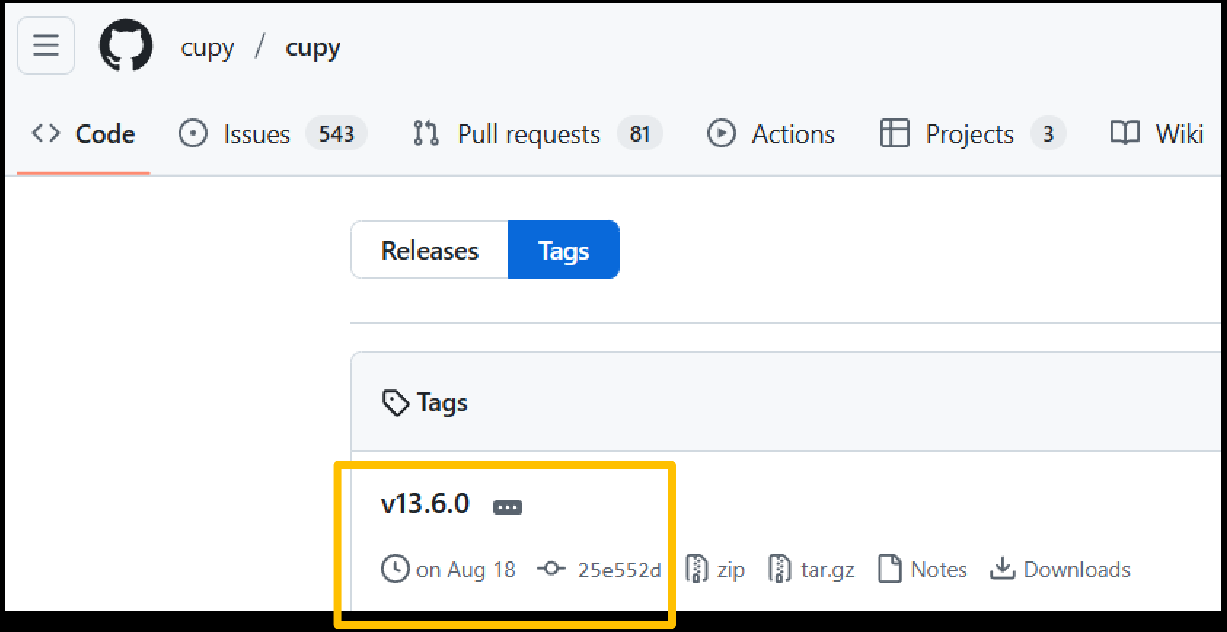
- We are using the upstream CuPy repository: https://github.com/cupy/cupy

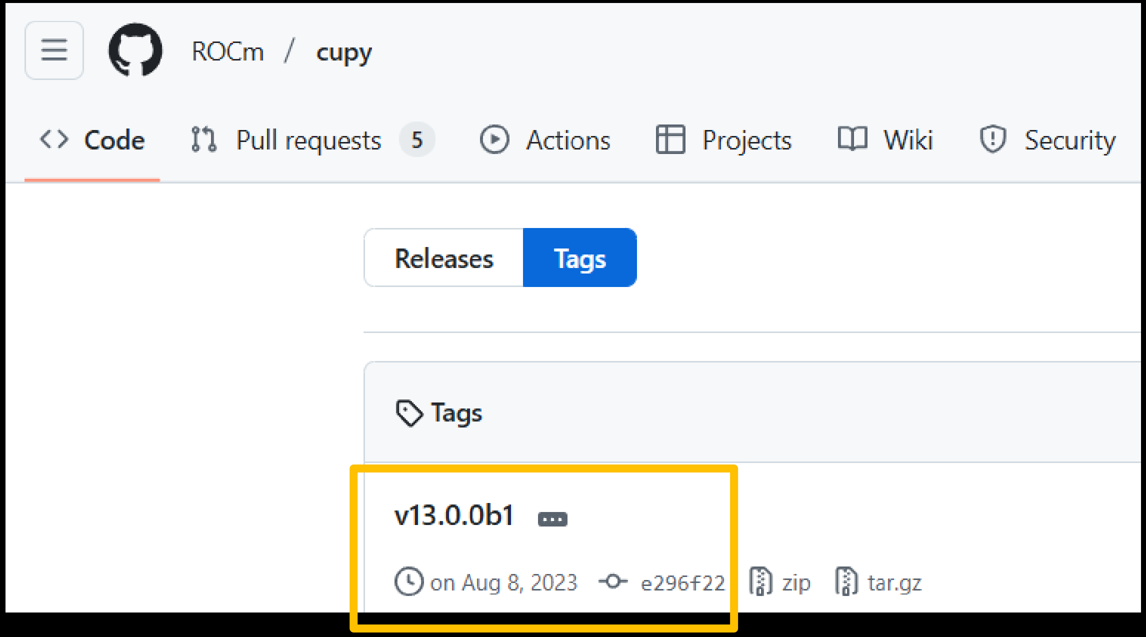



- There is also a fork of the CuPy upstream repository in the ROCm github: https://github.com/rocm/cupy



# CuPy Installation – Versions




Upstream versions are more recent

The one above is the one we have installed

As of September 23rd 2025

ROCm repo versions tend to be behind

There is work from AMD to get changes pushed directly to the upstream repo

The `ROCm/cupy` will soon be updated too



AMD
together we advance_

# Current AMD work to be integrated into upstream repo

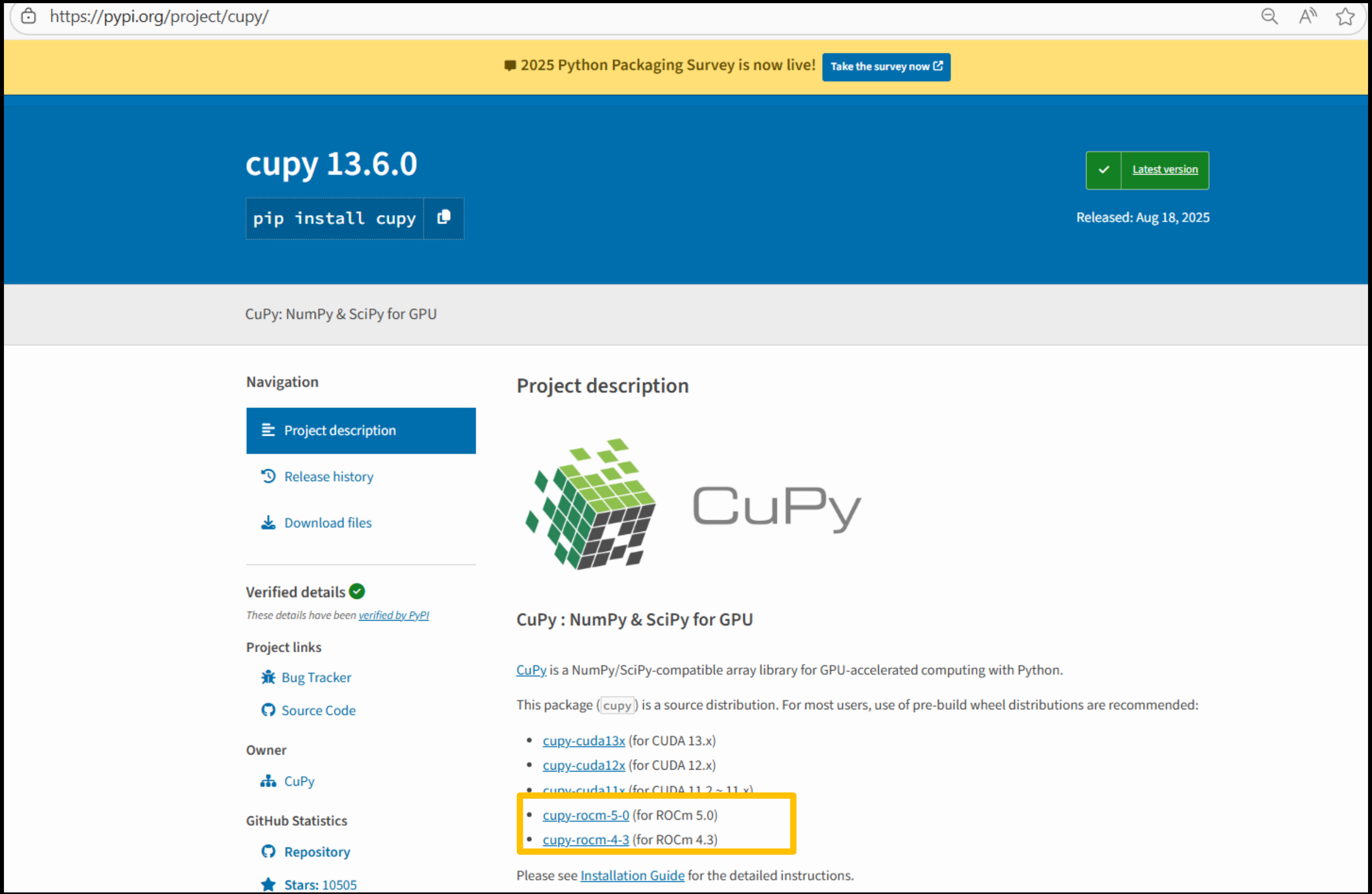
rocm

Open 24 Closed 441 | Author | Labels | Projects | Milestones

- **Enable rocm7** `cat:enhancement` `prio:high`
  cupy/cupy#9398 · gpinkert opened last week · Review required · v14.0.0a2 · ✕ 5/9
- **Adding ROCm 7.0.0 to the CI matrices.** `cat:test` `prio:high` `skip-ci`
  cupy/cupy#9397 · by gpinkert was closed last week · Approved · v14.0.0a2 · ✓ 5/14
- **Hipblas v2 api**
  cupy/cupy#9389 · by amd-nicknick was closed last week · Review required · ✓ 1/3
- **feat(hipCUB) Use c++17 for hipCUB** `cat:enhancement` `prio:medium`
  cupy/cupy#9384 · gpinkert opened 2 weeks ago · Review required · v14.0.0a2 · ✓ 5/9
- **feat(kernels) fix HIPRTC build error ROCm 6/7** `cat:enhancement` `prio:medium`
  cupy/cupy#9382 · by gpinkert was closed last week · Approved · v14.0.0a2 · ✕ 6/12
- **feat(CI) ROCm 6.4.3** `cat:test` `prio:medium` `skip-ci`
  cupy/cupy#9380 · by gpinkert was closed last week · Review required · v14.0.0a2 · ✓ 5/10
- **feat(import) Raise import error** `cat:install` `no-compat` `prio:medium`
  cupy/cupy#9351 · gpinkert opened 3 weeks ago · Review required · v14 · ✓ 4/10
- **Support CUDA Array Inferface in ROCm build** `cat:enhancement` `prio:medium`
  cupy/cupy#9340 · by gpinkert was closed last week · Review required · v14.0.0a2 · ✓ 7/9

As of September 23rd 2025

# CuPy – Installation with `pip3` (pre-built wheel for Linux x86_64)




As of September 23rd 2025

```
pip3 install cupy-rocm-5-0

pip3 install cupy-rocm-4-3
```

Only old versions of ROCm currently available as pre-built wheels

Wheels for ROCm 6.4 and 7 will soon be available



AMD together we advance_

# CuPy – Simple Installation from Source (latest version)

```
export CUPY_INSTALL_USE_HIP=1
export ROCM_HOME=${ROCM_PATH}
export HIPCC=${ROCM_HOME}/bin/hipcc
export HCC_AMDGPU_ARCH=${AMDGPU_GFXMODEL}

pip3 install cupy --target=$CUPY_PATH
```

NOTE: it will **not** work on Ubuntu 24.04
need to use a virtual environment

```
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.
```

> **Note**
>
> If you don't specify the `HCC_AMDGPU_TARGET` environment variable, CuPy will be built for the GPU architectures available on the build host. This behavior is specific to ROCm builds; when building CuPy for NVIDIA CUDA, the build result is not affected by the host configuration.

source: cupy docs

# CuPy – Robust Installation from Source

Installation from source script available in our model installation repository:
https://github.com/amd/HPCTrainingDock/blob/main/extras/scripts/cupy_setup.sh

also installs **numpy-allocator**
to leverage unified shared memory

and **cupy-xarray**

… most relevant part reported below…

```
export CUPY_INSTALL_USE_HIP=1
export ROCM_HOME=${ROCM_PATH}
export HIPCC=${ROCM_HOME}/bin/hipcc
export HCC_AMDGPU_ARCH=${AMDGPU_GFXMODEL}
python3 -m venv cupy_build
source cupy_build/bin/activate
pip3 install -v --target=$CUPY_PATH pytest mock xarray[complete] dask build numpy-allocator --no-cache
export PYTHONPATH=$PYTHONPATH:$CUPY_PATH
# Get source from the upstream repository of CuPy.
git clone -q --depth 1 -b v$CUPY_VERSION --recursive https://github.com/cupy/cupy.git
cd cupy
python3 -m build --wheel
pip3 install -v --upgrade --target=$CUPY_PATH dist/*.whl
pip3 install -v --target=$CUPY_PATH cupy-xarray --no-deps
deactivate
```

# Basics of CuPy

- Must import the CuPy Python™ module in your Python code: `import cupy as cp`
- To create an array on the device use `cp.array`: `gpu_array = cp.array(cpu_array)`
- To copy data from GPU to CPU, use `cp.asnumpy`: `cpu_array = cp.asnumpy(gpu_array)`
- To copy back from CPU to GPU use `cp.asarray`: `gpu_array2 = cp.asarray(cpu_array)`
- Operations between GPU arrays will be done on the GPU: `result_gpu = gpu_array + gpu_array2`
- CuPy has the concept of a current device – usually GPU device 0: `gpu_array.device`
- Note that the device will be called `<CUDA Device 0>` even if you are on AMD GPUs.

[Public]

# NumPy – CuPy Interoperability

- CuPy implements a subset of the NumPy interface by implementing `cupy.ndarray`, a counterpart to NumPy `ndarrays`

- The `cupy.ndarray` object implements the `__array_ufunc__` interface. This enables NumPy universal functions (ufunc) to be applied to CuPy arrays. Note that the return type of these operations **is still consistent** with the **initial type**.

```
>>> import cupy as cp
>>> import numpy as np
>>> gpu_arr = cp.random.randn(1, 2, 3, 4).astype(cp.float32)
>>> result = np.sum(gpu_arr)
>>> print(type(result))
<class 'cupy._core.core.ndarray'>
```

- `cupy.ndarray` also implements the `__array_function__` interface, meaning it is possible to do operations such as

```
a = np.random.randn(100, 100)
a_gpu = cp.asarray(a)
qr_gpu = np.linalg.qr(a_gpu)
```

source: numpy-documentation

# Simple CuPy code example

- First get the example to run from the training examples repository

```
git clone https://github.com/amd/HPCTrainingExamples
cd HPCTrainingExamples/Python/cupy
```

- Set up the environment: note the "module" below is not the Python™ module

```
module load cupy
```

- Run the example

```
python3 cupy_array_sum.py
```

- Output should be:

```
CuPy Array: [1 2 3 4 5]
Squared CuPy Array: [ 1  4  9 16 25]
NumPy Array: [5 6 7 8 9]
CuPy Array from NumPy: [5 6 7 8 9]
Addition Result on GPU: [ 6  8 10 12 14]
Result on CPU: [ 6  8 10 12 14]
```

# Simple CuPy code example: a closer look

```
import cupy as cp
import numpy as np

# Create a CuPy array
gpu_array = cp.array([1, 2, 3, 4, 5])
print("CuPy Array:", gpu_array)

# Perform operations on the GPU
gpu_array_squared = gpu_array ** 2
print("Squared CuPy Array:", gpu_array_squared)

# Create a NumPy array
cpu_array = np.array([5, 6, 7, 8, 9])
print("NumPy Array:", cpu_array)

# Transfer NumPy array to GPU
gpu_array_from_cpu = cp.asarray(cpu_array)
print("CuPy Array from NumPy:",
gpu_array_from_cpu)

# Perform element-wise addition
result_gpu = gpu_array + gpu_array_from_cpu
print("Addition Result on GPU:", result_gpu)

# Transfer result back to CPU
result_cpu = cp.asnumpy(result_gpu)
print("Result on CPU:", result_cpu)
```

Creates an array on the device

Operations occur on the GPU

Converts NumPy array to CuPy array

Operations occur on the GPU

Returns an array on the host memory from an arbitrary source array (device in this case)

# Verifying that CuPy code example runs on the AMD GPU

- Now run with the `AMD_LOG_LEVEL` environment variable set

```
export AMD_LOG_LEVEL=3
python3 cupy_array_sum.py
```

- Lots of output now – showing just a little bit:

```
hipMemcpyAsync ( 0x559ea98f65f0, 0x7f4556800000, 40, hipMemcpyDeviceToHost, stream:<null> )
Signal = (0x7f4d5efff280), Translated start/end = 1083534945452078 / 1083534945453358,
  Elapsed = 1280 ns, ticks start/end = 27091222405615 / 27091222405647, Ticks elapsed = 32
Host active wait for Signal = (0x7f4d5efff200) for -1 ns
Set Handler: handle(0x7f4d5efff180), timestamp(0x559eaabead90)
Host active wait for Signal = (0x7f4d5efff180) for -1 ns
hipMemcpyAsync: Returned hipSuccess : : duration: 5948d us
hipStreamSynchronize ( stream:<null> )
Handler: value(0), timestamp(0x559eaa7e7350), handle(0x7f4d5efff180)
hipStreamSynchronize: Returned hipSuccess :
hipSetDevice ( 0 )
hipSetDevice: Returned hipSuccess :
CuPy Array: [1 2 3 4 5]
```

# Unified Memory Programming on CuPy

## Unified memory programming (UMP) support (**experimental!**)

It is possible to make both NumPy and CuPy use/share system allocated memory on Heterogeneous Memory Management (HMM) or Address Translation Services (ATS) enabled systems, such as the NVIDIA Grace Hopper Superchip. To activate this capability, currently you need to:

1. Install numpy_allocator
2. Set the environment variable `CUPY_ENABLE_UMP=1`
3. Make a memory pool for CuPy to draw system memory (`malloc_system`), for example:

```
import cupy as cp
cp.cuda.set_allocator(cp.cuda.MemoryPool(cp.cuda.memory.malloc_system).malloc)
```

4. Switch to the aligned allocator for NumPy to draw system memory

```
import cupy._core.numpy_allocator as ac
import numpy_allocator
import ctypes
lib = ctypes.CDLL(ac.__file__)
class my_allocator(metaclass=numpy_allocator.type):
    _calloc_ = ctypes.addressof(lib._calloc)
    _malloc_ = ctypes.addressof(lib._malloc)
    _realloc_ = ctypes.addressof(lib._realloc)
    _free_ = ctypes.addressof(lib._free)
my_allocator.__enter__()  # change the allocator globally
```

On AMD GPUs, you additionally need:
`export HSA_XNACK=1`

this will enable unified shared memory on MI300A

or managed memory on MI200s and MI300X

With this setup change, all the data movement APIs such as `get()`, `asnumpy()` and `asarray()` become no-op (no copy is done), and the following code is accelerated:

```
# a, b, c are np.ndarray, d is cp.ndarray
a = np.random.random(100)
b = np.random.random(100)
c = np.add(a, b)
d = cp.matmul(cp.asarray(a), cp.asarray(c))
```

Essentially, the distinction of CPU/GPU *memory* spaces is gone, and `np` / `cp` are used to represent the *execution* space (whether the code should run on CPU or GPU).

Apart from the setup configuration for NumPy/CuPy, no user code chage is required.

source: cupy docs

# Unified Memory Programming on CuPy – benchmark 1/2

Unified memory example in training examples repo:
https://github.com/amd/HPCTrainingExamples/blob/main/Python/cupy/unified_bench.py

How to run the code:

```
gcapodag@ppac-pl1-s24-26:~/repos/HPCTrainingExamples/Python/cupy$ python3 cupy_ump_bench.py --help
usage: cupy_ump_bench.py [-h] [--sizes [SIZES ...]] [--repeats REPEATS] [--warmups WARMUPS] [--dtype DTYPE] [--csv CSV]

CuPy UMP benchmark (baseline vs UMP).

options:
  -h, --help            show this help message and exit
  --sizes [SIZES ...]   Sizes like: 8MiB 64MiB 1GiB (default: 16MiB 256MiB 1GiB)
  --repeats REPEATS     Repeats per measurement (default: 100)
  --warmups WARMUPS     Warmup iterations per measurement (default: 3)
  --dtype DTYPE         CuPy dtype name (default: float32)
  --csv CSV             Optional CSV path
```

The benchmark will transfer data and run some simple operations comparing a baseline (no UMP) with the UMP case (`export CUPY_ENABLE_UMP=1` and `export HAS_XNACK=1`)

# Unified Memory Programming on CuPy – benchmark 2/2

It measures the following:

- host to device (HtoD) copies with `cp.asarray(np_arr)`
- device to host (DtoH) copies with `cp_arr.get()`
- adding two arrays on device (x+=y)
- pipeline mixing the above three operations (copy to GPU, add on GPU and copy back to CPU)

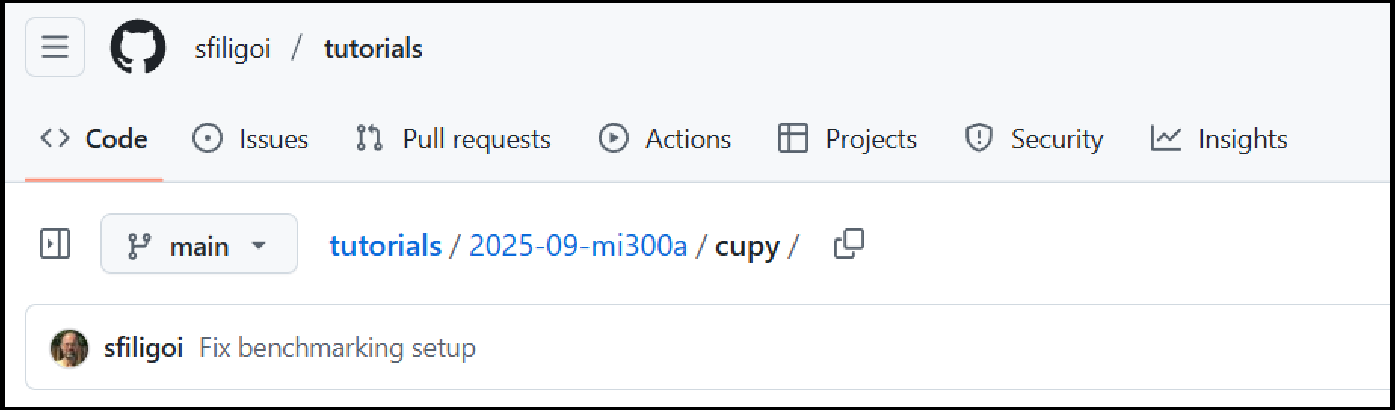
Results:

```
gcapodag@ppac-pl1-s24-26:~/repos/HPCTrainingExamples/Python/cupy$ python3 cupy_ump_bench.py

=== Running BASELINE (CUPY_ENABLE_UMP=0) ===

=== Running UMP (CUPY_ENABLE_UMP=1) ===

=== RESULTS (avg over repeats) ===
Size     | Mode | HtoD ms | HtoD GiB/s | DtoH ms | DtoH GiB/s | Add ms | Pipeline ms
---------+------+---------+------------+---------+------------+--------+------------
1.00 GiB | BASE | 18.699  | 53.480     | 116.304 | 8.598      | 1.310  | 160.730
1.00 GiB | UMP  | 0.004   | 242413.024 | 125.652 | 7.958      | 1.381  | 125.040
[BASELINE] device=AMD Instinct MI300A, SMs=228, mem=94.17 GiB, dtype=float32, repeats=100, warmups=3
[UMP] device=AMD Instinct MI300A, SMs=228, mem=94.17 GiB, dtype=float32, repeats=100, warmups=3
```



AMD together we advance_

# Additional Examples on CuPy




Code examples by Igor Sfiligoi

```
git clone https://github.com/sfiligoi/tutorials.git
cd tutorials/2025-09-mi300a/cupy
```

From 2025 ICPP AI tutorial
by San Diego Supercomputing Center
and AMD.

Compare `python3 center_matrix_naive.py` with `CUPY_ENABLE_UMP=1 \`
`HSA_XNACK=1 python3 center_matrix_naive_apu.py`

```
Small on CPU:  0.027914762496948242
Small on GPU:  0.011859893798828125
Medium matrix shape:  (8382, 8382)
Medium on CPU:  0.21364235877990723
Medium on GPU:  0.11696290969848633
Large matrix shape:  (25145, 25145)
Large on CPU:  1.9389822483062744
Large on GPU:  1.1117446422576904
... Retrying
Large on CPU:  1.8522789478302002
Large on GPU:  0.890662670135498
```

```
Small on CPU:  0.033936262130737305
Small on GPU:  0.06000709533691406
Medium matrix shape:  (8382, 8382)
Medium on CPU:  0.22278118133544922
Medium on GPU:  0.4624929428100586
Large matrix shape:  (25145, 25145)
Large on CPU:  1.8617160320281982
Large on GPU:  4.33487343788147
... Retrying
Large on CPU:  1.800774097442627
Large on GPU:  0.9342784881591797
```

# CuPy-Xarray: Xarray on GPUs

- Xarray: Python™ library to work with labelled multi-dimensional arrays
  - Popular for applications where multi-dimensional data needs to be handled (such as climate modeling)
  - Built on top of NumPy
  - Has built-in support for NetCDF
  - Can wrap custom duck array objects (i.e. NumPy-like arrays) that follow specific protocols.

- When used together, Xarray and CuPy can provide an easy way to take advantage of GPU acceleration for scientific computing tasks.

- CuPy-Xarray provides an interface for using CuPy in Xarray, providing accessors on the Xarray objects.
  - CuPy-Xarray relies on an existing CuPy installation, install CuPy first

- Cupy-Xarray github repo: https://github.com/xarray-contrib/cupy-xarray
  - Install with `pip install cupy-xarray --no-deps` after installing CuPy

- Issue with `dask`: https://github.com/xarray-contrib/cupy-xarray/pull/62
  - Did not make it into the latest release
  - Make sure to install dask with `pip install dask`

source: cupy-xarray documentation

# Simple CuPy-Xarray code example

- First get the example to run from the training examples repository

```
git clone https://github.com/amd/HPCTrainingExamples
cd HPCTrainingExamples/Python/cupy
```

- Set up the environment: note the "module" below is not the Python™ module

```
module load cupy
```

- Run the example

```
python cupy_xarray_test.py
```

```
Is the array used to create da_np on device? False
Is the array used to create da_cp on device? True
da_cp.data is of type: <class 'cupy.ndarray'>
check that arr_gpu and cupy_array are the same with CuPy: True
check the arr_gpu and cupy_array are the same with NumPy (interoperability): True
arr_gpu is on device: <CUDA Device 0>
arr_cpu is on device: cpu
total number of available devices: 8
arr_gpu2 is on device: <CUDA Device 2>
```

source: [cupy-xarray documentation]

# Simple CuPy-Xarray code example: a closer look

```
import cupy as cp
import numpy as np
import xarray as xr
import cupy_xarray
```

Adds .cupy to Xarray objects

```
arr_cpu = np.random.rand(10, 10, 10)
arr_gpu = cp.random.rand(10, 10, 10)
```

Creates an array on the CPU with NumPy
Creates an array on the GPU with CuPy

```
da_np = xr.DataArray(arr_cpu, dims=["x", "y", "time"])
da_cp = xr.DataArray(arr_gpu, dims=["x", "y", "time"])
```

Creates a DataArray using NumPy array
Creates a DataArray using CuPy array

. . . (some code omitted) . . .

```
cupy_array = da_cp.data
```

Access the underlying CuPy array used to create the xarray.DataArray

```
print("check that arr_gpu and cupy_array are the same with CuPy:",
cp.allclose(cupy_array,arr_gpu))
```

Use CuPy to check that the array used to create the xarray and the one given by xarray are the same

```
print("check the arr_gpu and cupy_array are the same with NumPy
(interoperability):", np.allclose(cupy_array,arr_gpu))
```

Use NumPy to check that the array used to create the xarray and the one given by xarray are the same

. . . (some code omitted) . . .

```
with cp.cuda.Device(2):
    arr_gpu2 = cp.array([1, 2, 3, 4, 5])
print("arr_gpu2 is on device:", arr_gpu2.device)
```

Use the device context manager to create data on a different device

# Additional Resources

- CuPy vs NumPy speed comparison: https://cupy-xarray.readthedocs.io/latest/examples/01_cupy-basics.html#cupy-vs-numpy-speed-comparison

- Real world example of Cupy-Xarray: https://cupy-xarray.readthedocs.io/latest/examples/06_real-example.html
  - Note: you might need to modify the data read line to this if it is taking too long to get the data:
    ```
    da = xr.open_mfdataset(file_objs, engine="h5netcdf", compat="override", coords='minimal')[var].load()
    ```

- Cupy with Xarray vs NumPy with Xarray performance comparison: https://cupy-xarray.readthedocs.io/latest/examples/03_basic-computations.html#comparing-performance-cupy-with-xarray-vs-numpy-with-xarray

# Disclaimer

AMD