RAPIDS

cuCIM - A GPU image I/O and processing library

Gregory R. Lee glee@quansight.com
Gigon Bae gbae@nvidia.com

Contributors: Rahul Choudhury, John Kirkham, and Benjamin Zaitlen

Quansight NVIDIA

Agenda

What is cuCIM?

Compatible APIs for OpenSlide and scikit-image

High Performance Image I/O & Processing

DEMO

Getting Started with cuCIM

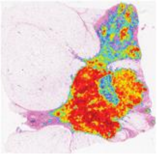
Image Processing Challenges

Background

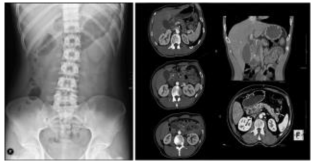
Image I/O:
Slow image loading and decoding

Image Processing:
Image pre and post processing with CPU only toolkits

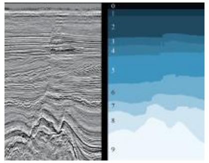
Applications in many fields using n-dimensional data share the problem.



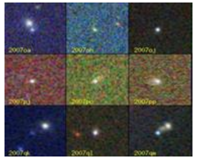
BioImaging



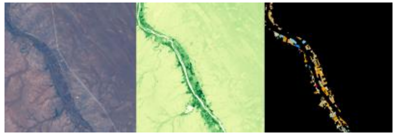
Medical imaging



Seismology



Astronomy



Remote Sensing

cuCIM

cuClara Image

cuCIM provides an **OpenSlide-like API** for loading various images including WSIs fast.

OpenSlide is a C/Python library to read whole-slide images (WSI) -- multi-resolution/tiled images

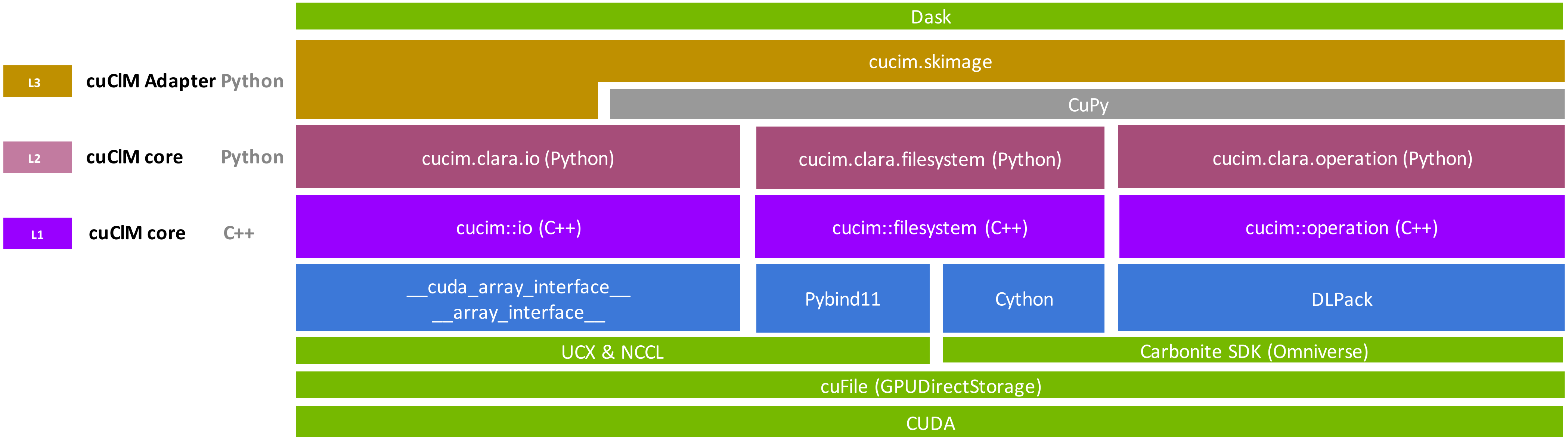
cuCIM provides a **scikit-image compatible API** with GPU-accelerated image processing.

scikit-image (a.k.a skimage) is a popular Python-based image processing library
: a collection of algorithms for image processing

cuCIM

Architecture

An extensible toolkit designed to provide GPU accelerated I/O, computer vision & image processing primitives for N-Dimensional images with a focus on biomedical imaging.



OpenSlide-like APIs

Loading a Partial Image from a TIFF File

OpenSlide

```python
from openslide import OpenSlide
from matplotlib import pyplot as plt

img = OpenSlide("image.tif")

count = img.level_count
dimensions = img.level_dimensions

for k,v in img.properties.items():
    print(k, v)

# Read whole slide at the lowest resolution
region = img.read_region(location=(0,0),
                         level=count-1,
                         size=dimensions[count-1])



plt.imshow(region)

```

cuCIM

```python
from cucim import CuImage
from matplotlib import pyplot as plt

img = CuImage("image.tif")

count = img.resolutions['level_count']
dimensions = img.resolutions['level_dimensions']


print(img.metadata)

## Read whole slide at the lowest resolution
# region = img.read_region(location=(0,0),
#                          size=dimensions[count-1],
#                          level=count-1)
# Or,
region = img.read_region(level=count-1)  # Same

plt.imshow(region)

```

Supporting Cache and Array Interface

Cache Usage

```python
from cucim import CuImage

img = CuImage('input/image.tif')
cache = CuImage.cache('per_process',
                      memory_capacity=2048,
                      record_stat=True)

region = img.read_region((0,0), (100,100))
print(f'cache hit: {cache.hit_count}, cache miss: {cache.miss_count}')
# cache hit: 0, cache miss: 1
print(region.__array_interface__)
# {'data': (93927971074032, False), ..., 'version': 3}

region = img.read_region((0,0), (100,100), device="cuda")
print(f'cache hit: {cache.hit_count}, cache miss: {cache.miss_count}')
# cache hit: 1, cache miss: 1
print(region.__cuda_array_interface__)
# {'data': (81888083968, False), ..., 'stream': 1}
```

Three strategies for 'Cache':

- no_cache
- per_process
- shared_memory (inter-process)

Support __array_interface__ and __cuda_array_interface__ for interoperability.

Loading and Processing Images with cuCIM

Load & Resize Image

```python
from matplotlib import pyplot as plt
import cupy as cp
from cucim import CuImage
from cucim.skimage.transform import resize

img = CuImage("image.tif")

region = img.read_region((10000, 10000), (4096, 4096))

# Transfer to GPU memory
array = cp.asarray(region)

resized_image = resize(array, (256, 256))

# Get a copy of the array on host memory and visualize
plt.imshow(resized_image.get())
```

An object returned by ***read_region()*** can be converted to ***cupy.ndarray*** object via ***cupy.asarray()***.

GPUDirect Storage (GDS) Support

Using cuFile API through CuFileDriver

Integration with cuFile

```python
from cucim.clara.filesystem import CuFileDriver
import cucim.clara.filesystem as fs
import os, cupy as cp, torch

# Assume a file ('nvme/input.raw') with size 10 in bytes

# Create a CuPy array with size 10 (in bytes)
cp_arr = cp.ones(10, dtype=cp.uint8)
# Create a PyTorch array with size 10 (in bytes)
cuda0 = torch.device('cuda:0')
torch_arr = torch.ones(10, dtype=torch.uint8, device=cuda0)

# Using CuFileDriver
# (Opening a file with O_DIRECT flag is required for GDS)
with os.open("nvme/input.raw", os.O_RDONLY | os.O_DIRECT) as fno:
    with CuFileDriver(fno) as fd:
        # Read 8 bytes starting from file offset 0 into buffer offset 2
        read_count = fd.pread(cp_arr, 8, 0, 2)
        # Read 10 bytes starting from file offset 3
        read_count = fd.pread(torch_arr, 10, 3)

# Another way of opening file with cuFile
with fs.open("nvme/output.raw", "w") as fd:
    # Write 10 bytes from cp_array to file starting from offset 5
    write_count = fd.pwrite(cp_arr, 10, 5)
```

```
nvme/input.raw                   10 [101 102 103 104 105 106 107 108 109 110]

cp_arr                           10 [  1   1   1   1   1   1   1   1   1   1]

torch_arr                        10 [  1   1   1   1   1   1   1   1   1   1]


cp_arr            read_count:   8 [  1   1 101 102 103 104 105 106 107 108]

torch_arr         read_count:   7 [104 105 106 107 108 109 110   1   1   1]


nvme/output.raw write_count: 10 [0 0 0 0 0 1 1 101 102 103 104 105 106 107 108]
```

Quansight NVIDIA

scikit-image compatible APIs

image processing in python

cuCIM provides an increasingly large subset of the scikit-image API

Enables rapid porting of existing scikit-image code to the GPU

This cucim.skimage module is currently built on top of CuPy

Performance is typically much better than scikit-image itself

Other RAPIDS libraries provide complementary functionality in areas such as machine learning (cuML), signal processing (cuSignal) and graph algorithms (cuGraph).

scikit-image compatible APIs

image processing in python

Adjusting exposure with the scikit-image like API

scikit-image

```python
import numpy as np

from skimage import data
from skimage import exposure

# Load an example image
img = data.moon()

# Contrast stretching
p2, p98 = np.percentile(img, (2, 98))
img_rescale = exposure.rescale_intensity(img, in_range=(p2, p98))

# Equalization
img_eq = exposure.equalize_hist(img)

# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
```

cuCIM

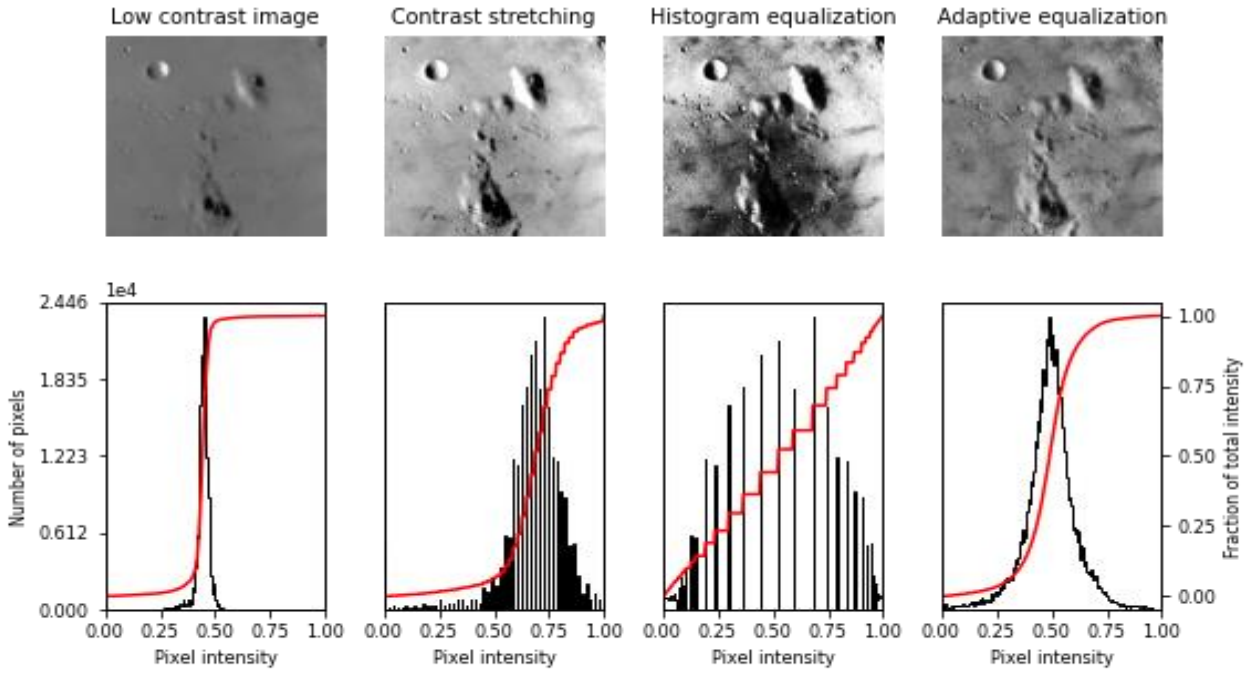
```python
import cupy as cp

from skimage import data
from cucim.skimage import exposure

# Load an example image
img = data.moon()

# Transfer to GPU memory
img = cp.asarray(img)

# Contrast stretching
p2, p98 = cp.asnumpy(cp.percentile(img, (2, 98)))
img_rescale = exposure.rescale_intensity(img, in_range=(p2, p98))

# Equalization
img_eq = exposure.equalize_hist(img)

# Adaptive Equalization
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
```

Quansight YOUR DATA EXPERTS NVIDIA

scikit-image compatible APIs

image processing in python

Adjusting exposure with the scikit-image like API



Adapted From: https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_local_equalize.html

scikit-image compatible APIs

image processing in python

Python Adaptation Layer - Current Status

Over 200+ Image Processing & Computer Vision Primitives Already GPU-enabled. Here are some examples

Feature Extraction

- canny
- corner_harris
- corner_shi_thomasi
- daisy
- match_templated
- shape_index
- structure_tensor
- …

Restoration

- calibrate_denoiser
- denoise_tv_chambolle
- richardson_lucy
- wiener
- unsupervised_wiener

Registration

- optical_flow_ilk
- optical_flow_tvl1
- phase_cross_correlation

Morphology

- binary_erosion
- binary_dilation
- erosion (greyscale)
- opening (greyscale)
- remove_small_objects
- …

Measure

- label
- centroid
- moments_central
- moments_hu
- shannon_entropy
- …

Metrics

- peak_signal_noise_ratio
- structural_similarity
- mean_square_error
- normalized_root_mse

Transforms

- resize
- rotate
- warp
- integral_image
- pyramid_gaussian
- …

Exposure

- histogram
- equalize_hist
- equalize_adaptive
- adjust_gamma
- match_histogram
- …

Segmentation

- random_walker
- morphological_chan_vese
- join_segmentations
- …

Color Conversions

- rgb2gray
- rgb2hsv
- rgb2yuv
- combine_stains
- separate_stains
- …

Filters

- gabor
- gaussian
- median
- sobel
- frangi
- hessian
- unsharp_mask
- threshold_local
- threshold_otsu
- threshold_niblack
- threshold_sauvola
- …

Quansight NVIDIA

scikit-image compatible APIs

image processing in python

Benefit of using CuPy & Dask

The cucim.skimage API operates on CuPy arrays.

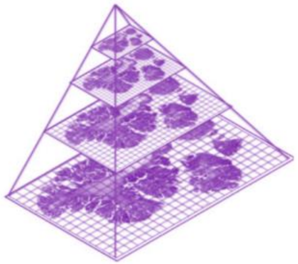
CuPy supports both DLPack and the __cuda_array_interface__ protocol for good interoperability with many other GPU-accelerated Python packages.
: Numba, Pytorch, Tensorflow, PaddlePaddle, MXNet, cuDF, and cuML.

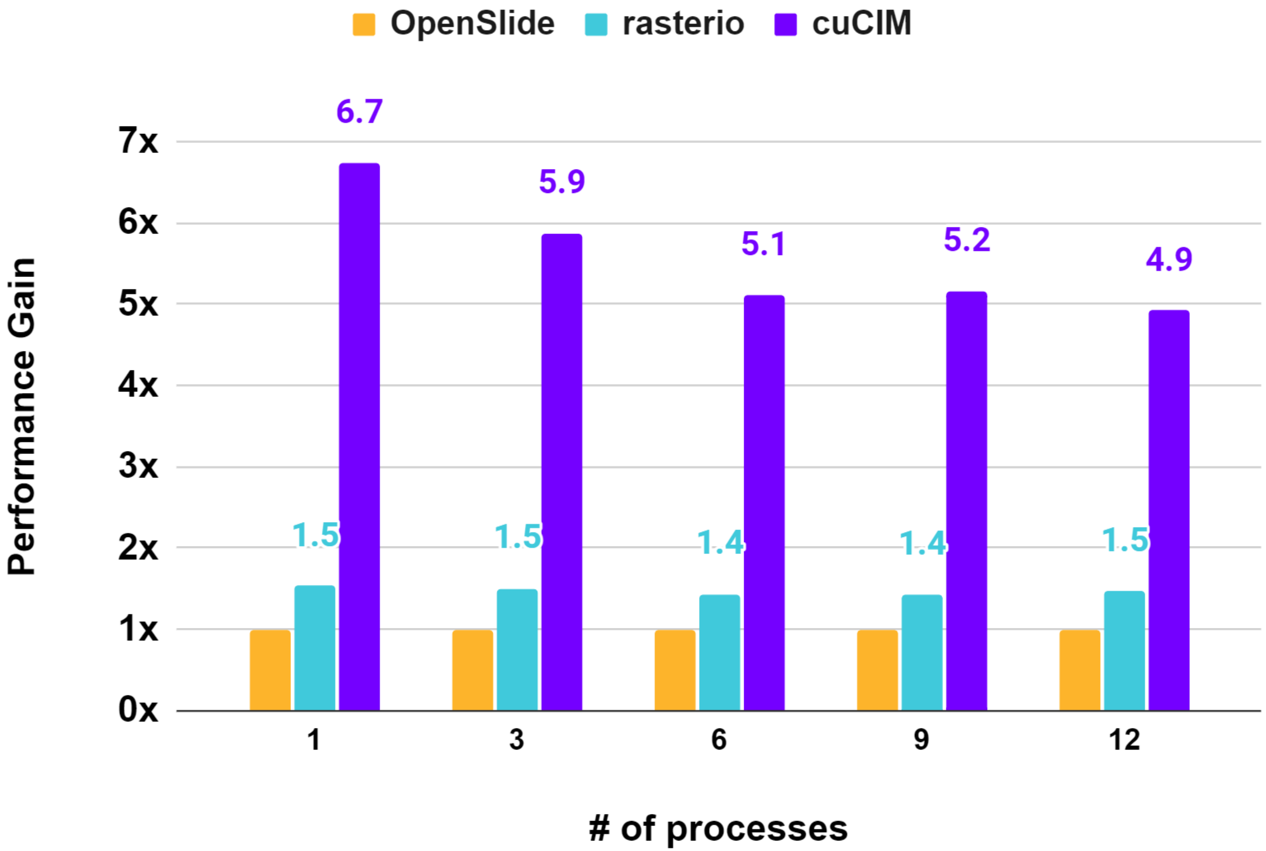
RAPIDS cuML, cuDF, cusignal and cugraph provide a lot of complementary functionality.

Can scale to distributed computation of large data using Dask (e.g. dask-cuda, dask-image)

High Performance Image I/O & Processing

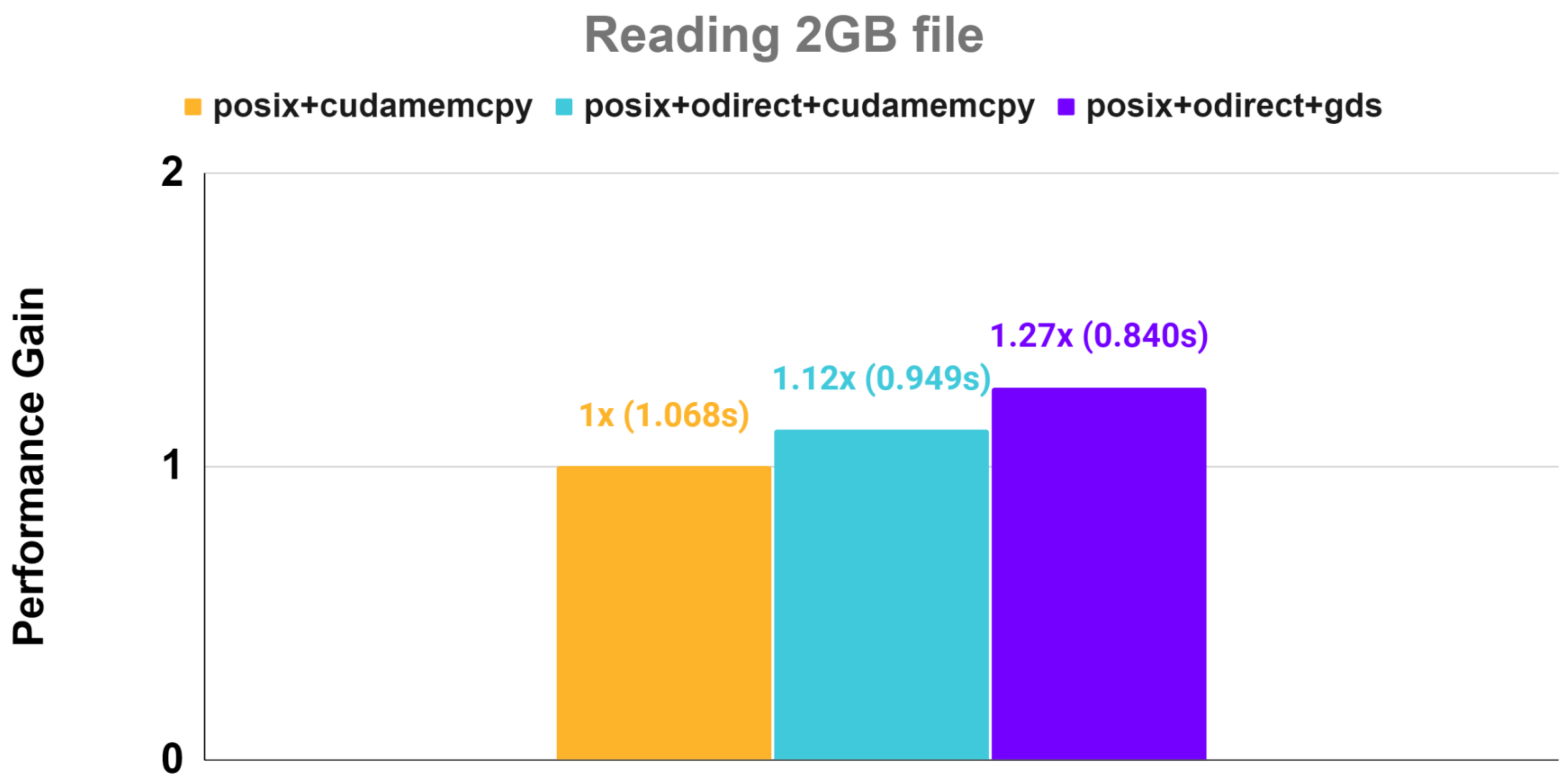
Great TIFF File Loading Performance





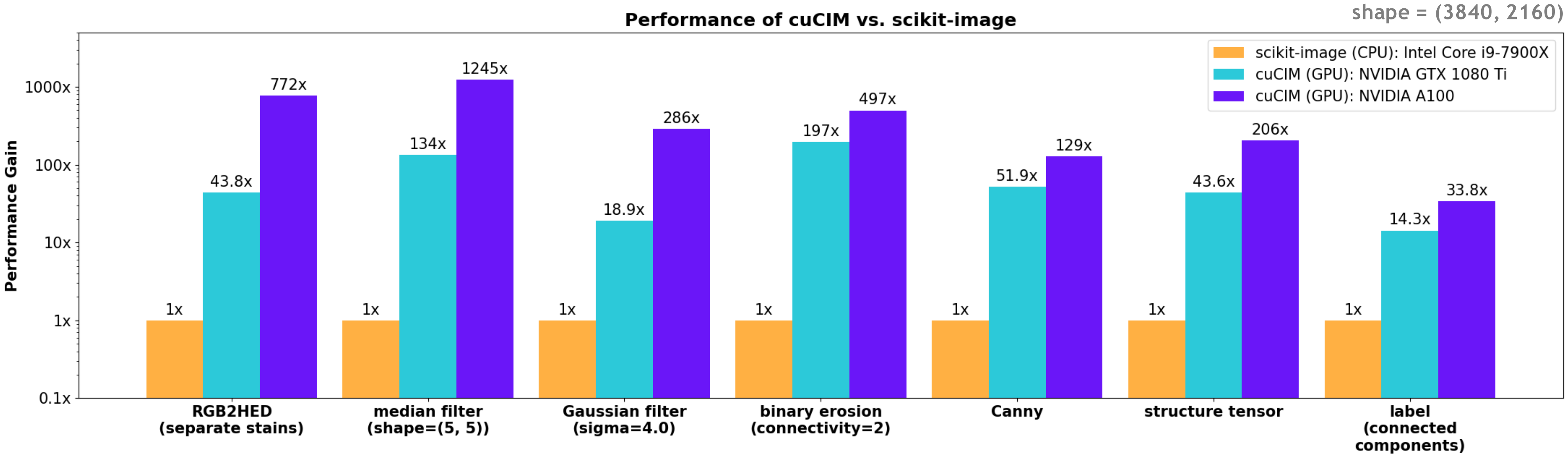
Used read_region() API to read all patches (256x256 size each) of a whole slide image (.tif) at the largest resolution level (92,344 x 81,017. Internal tile size is 256 x 256 with 95% JPEG compression quality level).
Original whole slide image (.svs : 1.6GB) was converted into .tif file (3.2GB) using OpenSlide & tifffile library in this experiment. Conducted experiments on Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz with Samsung SSD 850 EVO (1TB).
read_region() API doesn't use GPU-accelerated libraries for now. The results are based on only CPU. Detailed information is available [here](#).

Improved File Reading Performance with GPUDirect Storage (GDS)

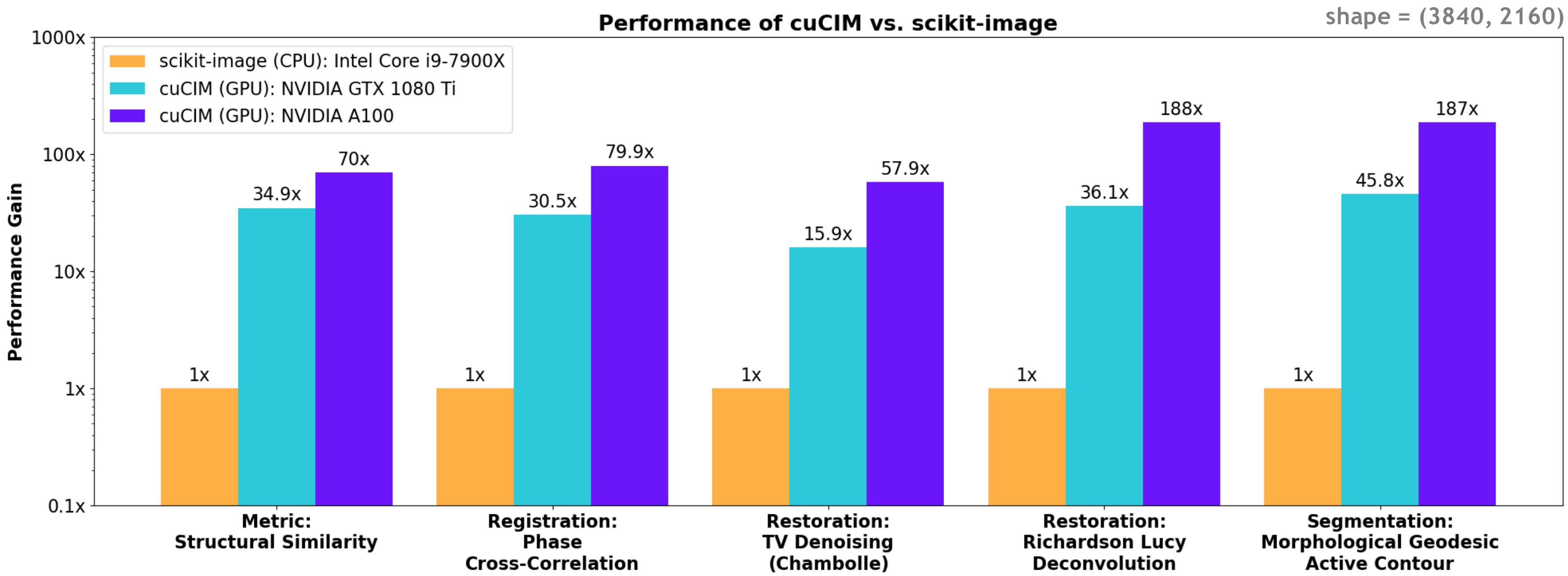


Used cuCIM's CuFileDriver class (cucim.clara.filesystem) with different configurations to read a raw 2 GB file into GPU memory.
Conducted experiments on Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz with Samsung SSD 970 PRO (NVMe SSD, 1TB).

Significant Performance Gain for Low-level Image Processing



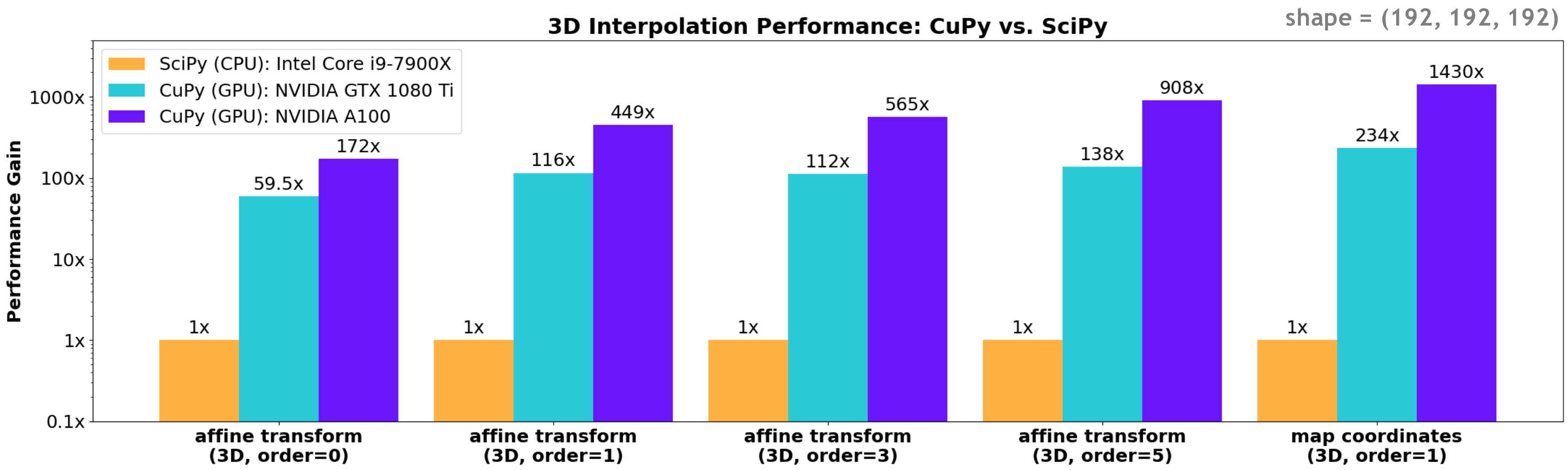
Significant Performance Gain for Complicated Operations



Community: CuPy scipy.ndimage coverage

n-dimensional Spline interpolation (orders 0-5) were contributed upstream

This new CuPy implementation matches the SciPy 1.6's updated API

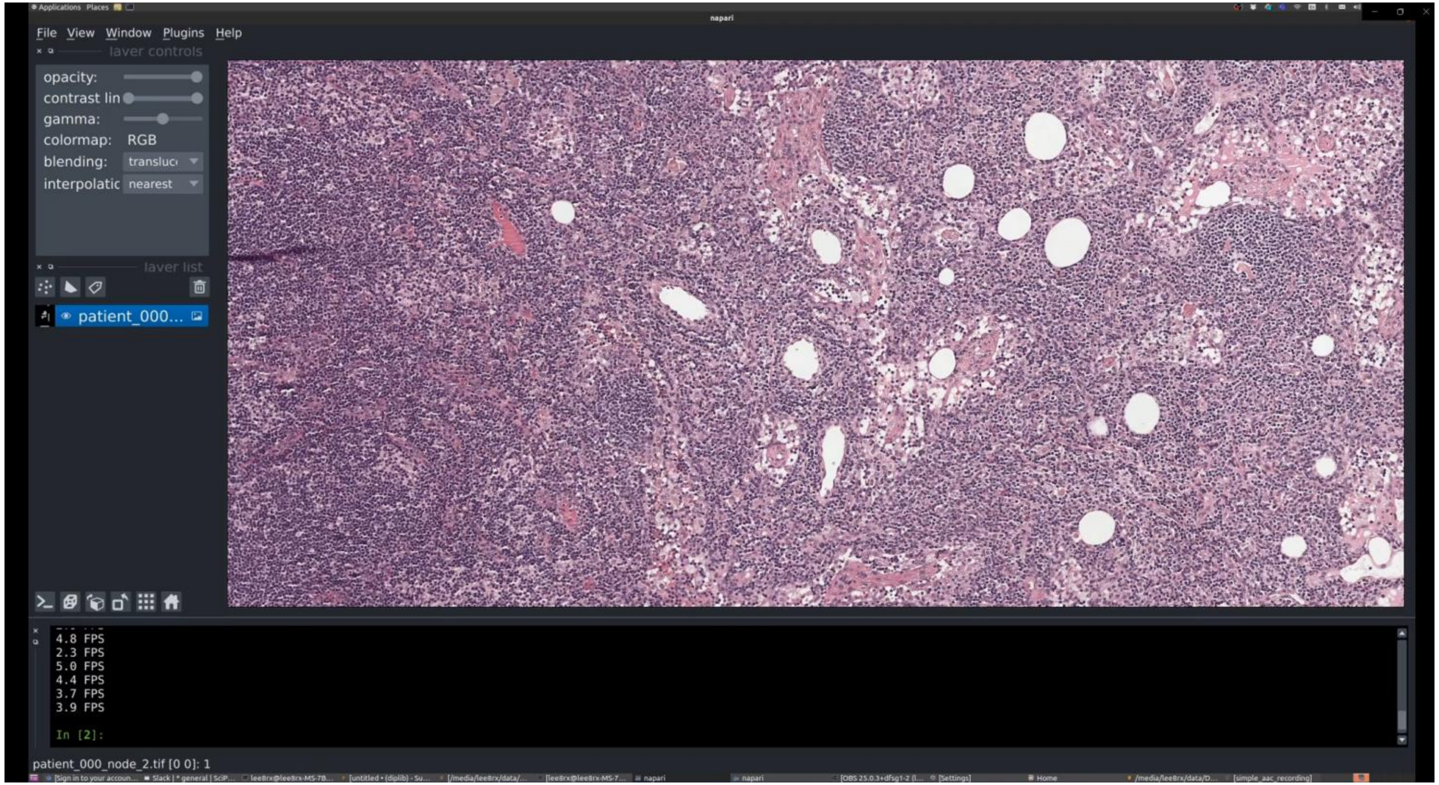


DEMO

Examples of using cuCIM API
Large Size Image Processing on Dask

Examples of using cuCIM API

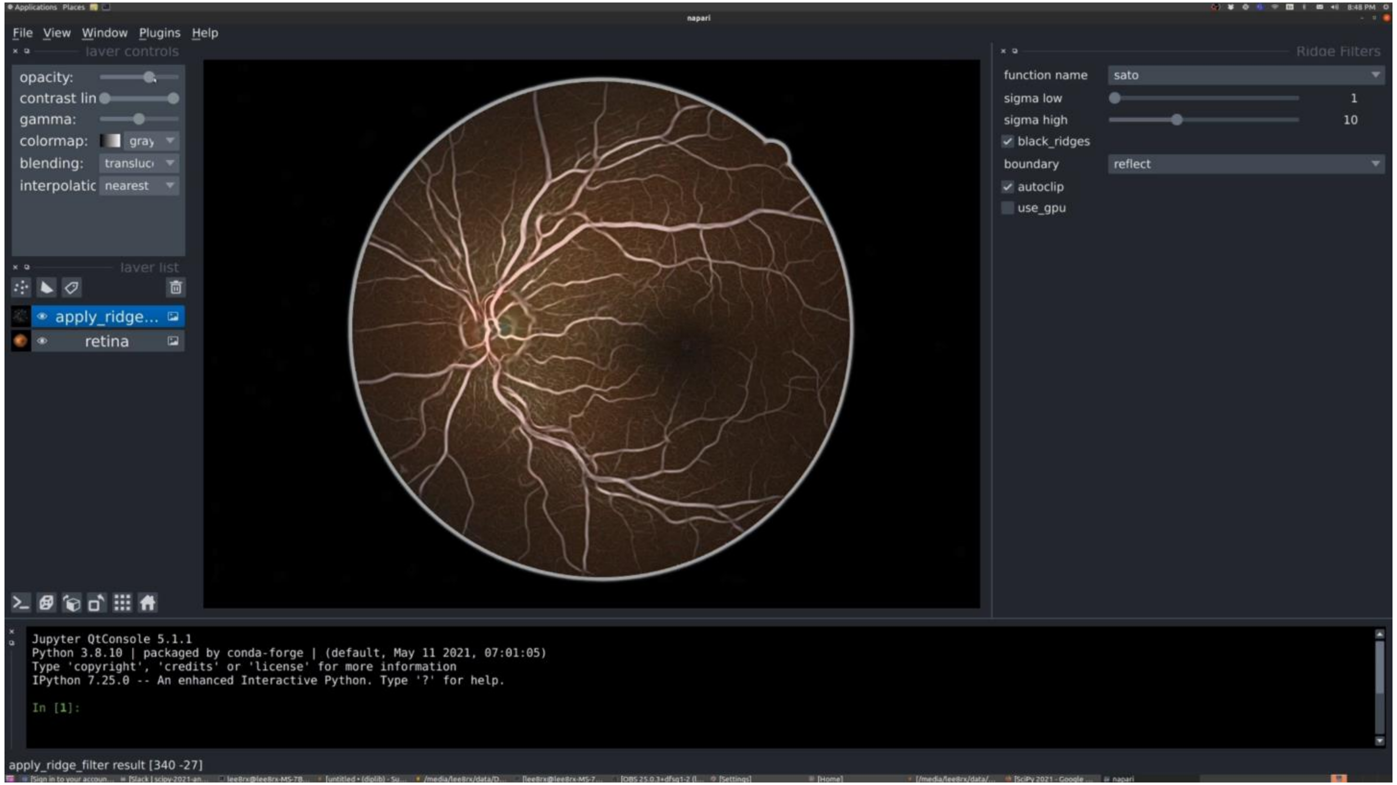
Napari Lazy-loading Demo



Applications Places
napari
6:11 PM
File View Window Plugins Help
layer controls
opacity:
contrast lin
gamma:
colormap: RGB
blending: transluc
interpolatic nearest
layer list
patient_000...
0.9 FPS
2.9 FPS
2.2 FPS
3.4 FPS
0.1 FPS
0.7 FPS
In [2]: viewer.window.qt_viewer.canvas.measure_fps()
patient_000_node_2.tif [154045 127816]: 6

Examples of using cuCIM API

Interactive Image Processing Demo



/media/lee8rx/data/Dropbox/Dropbox/Quansight/talks/SciPy2021/magicgui_vesselness_example.py (skimage, tests, tests, filters, viewer) - Sublime Text

```python
    sigma_high: int=10,
    black_ridges: bool=True,
    mode: str="reflect",
    autoclip: bool=True,
    use_gpu: bool=False,  #  cucim_available
) -> "napari.types.ImageData":

    if use_gpu:
        if not cucim_available:
            raise ValueError("cuCIM could not be imported")
        image = color_gpu.rgb2gray(cp.asarray(retina))
        filters_module = filters_gpu
        xp = cp
    else:
        image = color.rgb2gray(retina)
        filters_module = filters
        xp = np

    if sigma_high < sigma_low:
        raise ValueError("sigma_low must be < sigma_high")

    image = image.astype(np.float32)
    ridge_filter = getattr(filters_module, function_name)
    result = ridge_filter(image,
                          sigmas=range(sigma_low, sigma_high),
                          black_ridges=black_ridges,
                          mode=mode)
    if autoclip:
        # remove lowest and highest 0.5% of values
        vmin, vmax = xp.percentile(result, q=[0.5, 99.5])
        result = xp.clip(result, vmin, vmax)
```

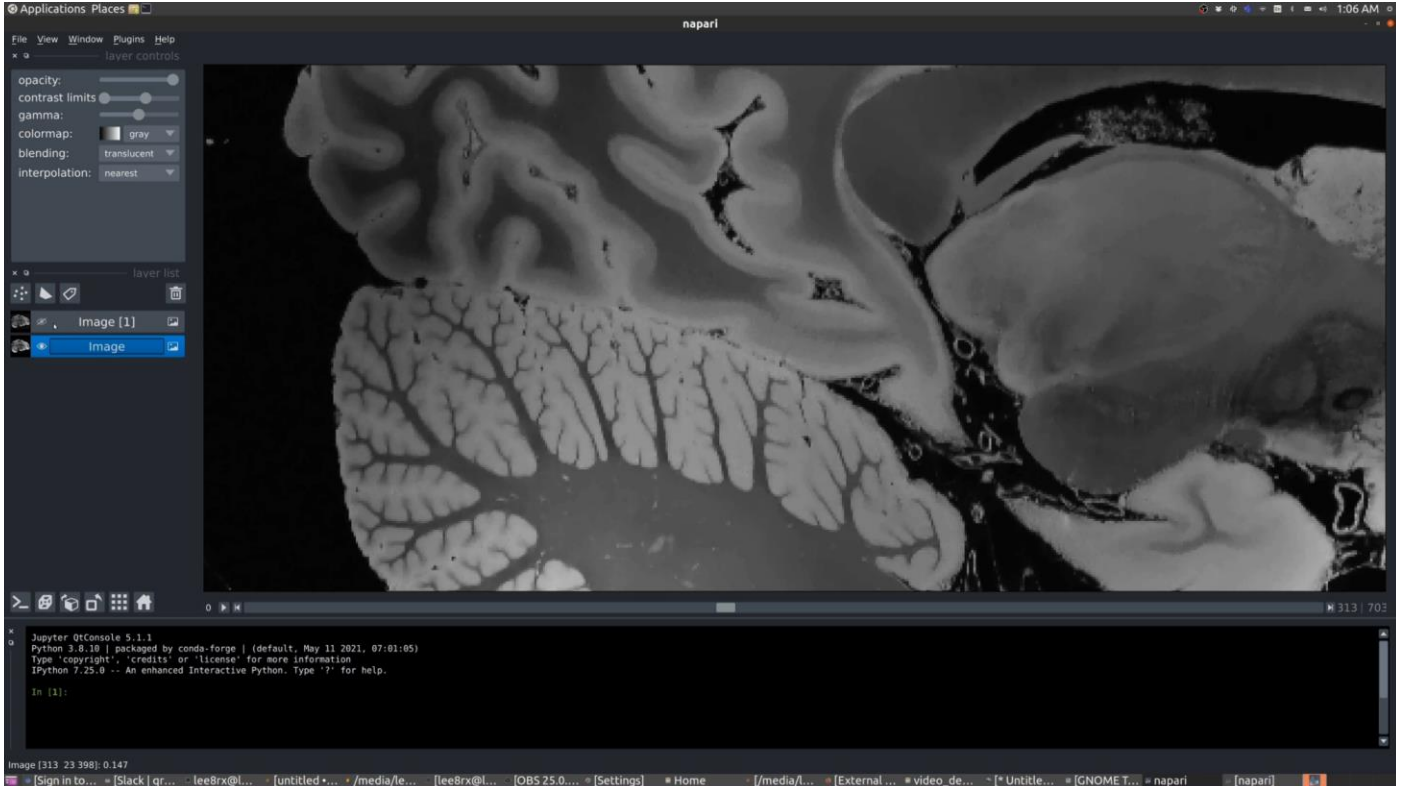
Line 58, Column 31 — Spaces: 4 — Python

lee8rx@lee8rx-MS-7B05: ~/Dropbox/Quansight/talks/SciPy2021

```
(napari-dev) lee8rx@lee8rx-MS-7B05:~/Dropbox/Quansight/talks/SciPy2021$ python magicgui_vesselness_example.py
/home/lee8rx/mambaforge/envs/napari-dev/lib/python3.8/site-packages/napari/_vispy/vispy_camera.py:109: RuntimeWarning: divide by zero encountered in true_divide
  zoom = np.min(canvas_size / scale)
(napari-dev) lee8rx@lee8rx-MS-7B05:~/Dropbox/Quansight/talks/SciPy2021$ subl ./demo1.sh
(napari-dev) lee8rx@lee8rx-MS-7B05:~/Dropbox/Quansight/talks/SciPy2021$ QT_SCALE_FACTOR=2 python magicgui_vesselness_example.py
/home/lee8rx/mambaforge/envs/napari-dev/lib/python3.8/site-packages/napari/_vispy/vispy_camera.py:109: RuntimeWarning: divide by zero encountered in true_divide
  zoom = np.min(canvas_size / scale)
(napari-dev) lee8rx@lee8rx-MS-7B05:~/Dropbox/Quansight/talks/SciPy2021$ QT_SCALE_FACTOR=2 python magicgui_vesselness_example.py
/home/lee8rx/mambaforge/envs/napari-dev/lib/python3.8/site-packages/napari/_vispy/vispy_camera.py:109: RuntimeWarning: divide by zero encountered in true_divide
  zoom = np.min(canvas_size / scale)
(napari-dev) lee8rx@lee8rx-MS-7B05:~/Dropbox/Quansight/talks/SciPy2021$
```

Large Size Image Processing on Dask

MRI Image Processing Demo



Quansight NVIDIA

Overview

Dask can be used for block-wise processing of large arrays

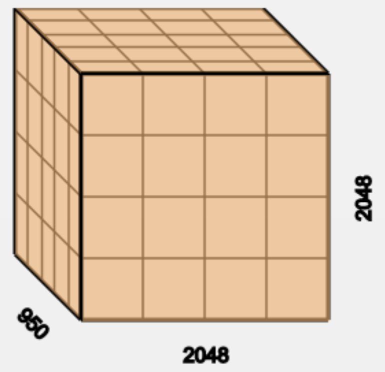
Common scenarios:

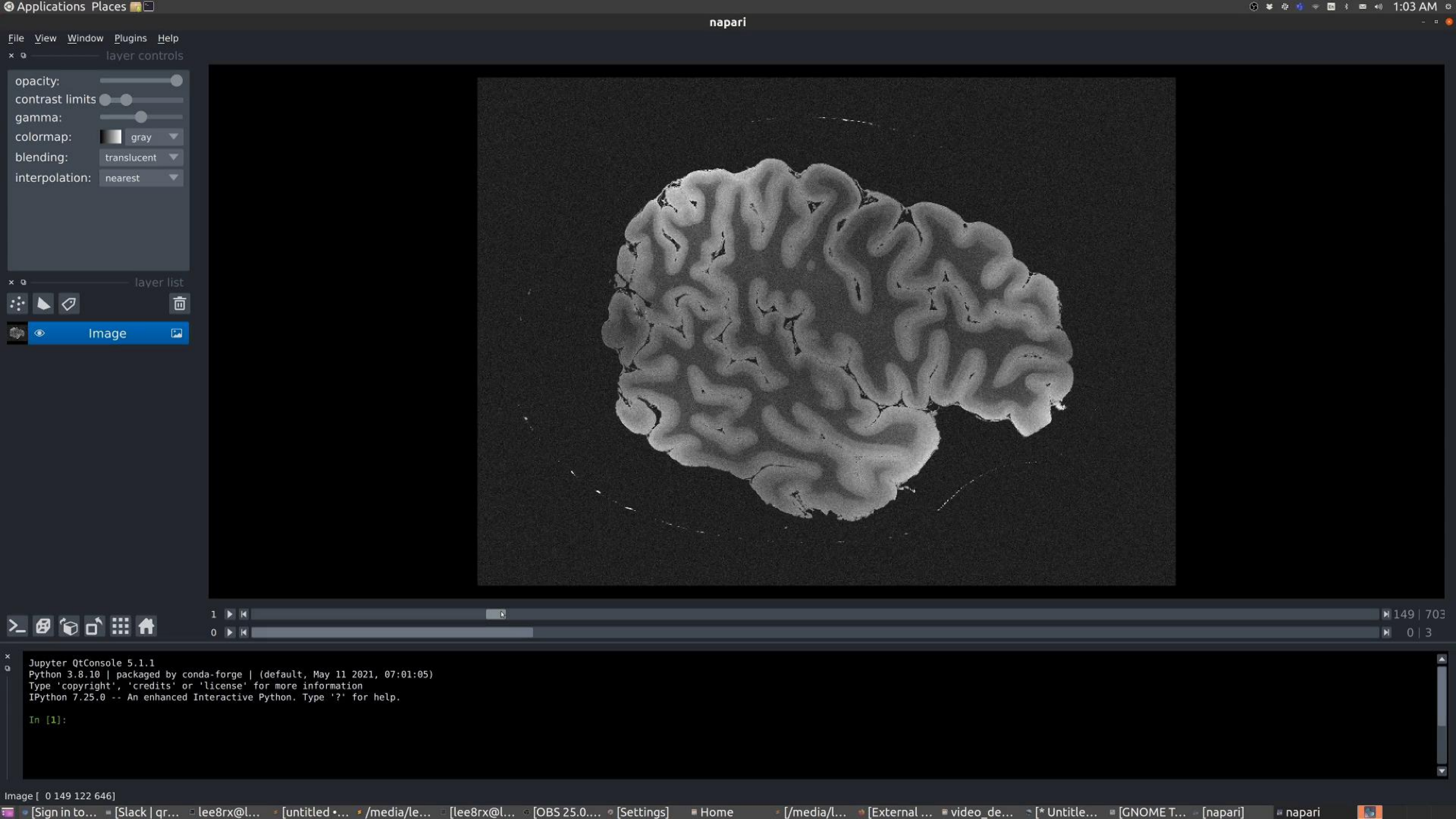
- Local block-wise processing to reduce peak memory requirements.
- Distributed block-wise processing to accelerate computations

Example:

Division of a large array into smaller "chunks" for block-wise processing.

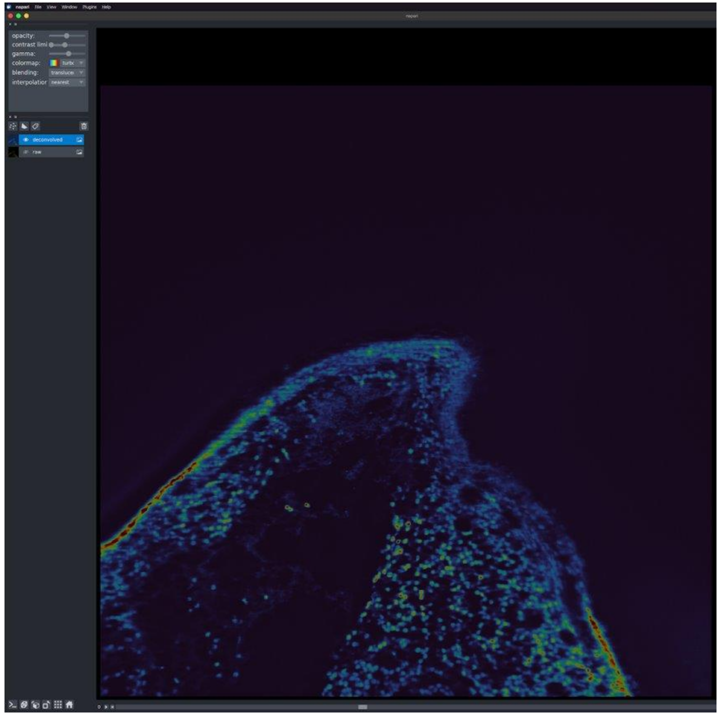
	Array	Chunk
Bytes	15.94 GB	199.23 MB
Shape	(950, 2048, 2048)	(190, 512, 512)
Count	80 Tasks	80 Chunks
Type	float32	cupy.ndarray

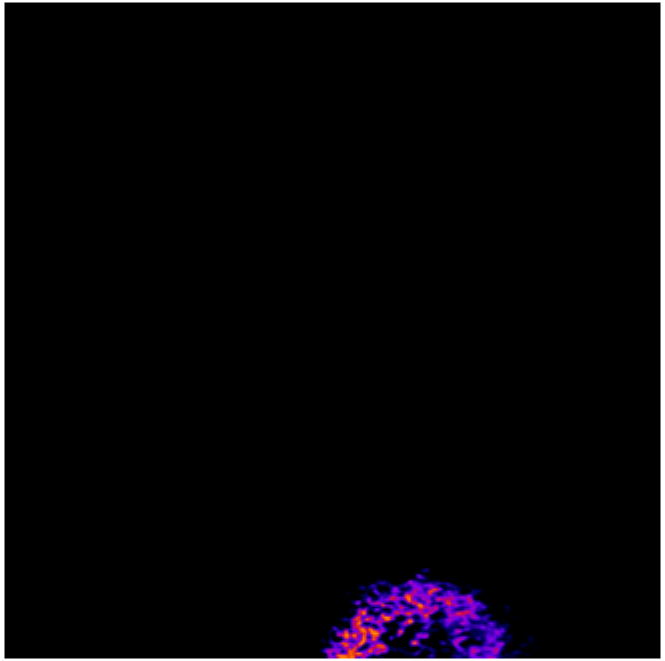



Applications Places
1:03 AM
napari
File View Window Plugins Help
layer controls
opacity:
contrast limits
gamma:
colormap: gray
blending: translucent
interpolation: nearest
layer list
Image
1
149 | 703
0
0 | 3
Jupyter QtConsole 5.1.1
Python 3.8.10 | packaged by conda-forge | (default, May 11 2021, 07:01:05)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.25.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
Image [0 149 122 646]
[Sign in to... [Slack | gr... lee8rx@l... [untitled •... /media/le... [lee8rx@l... [OBS 25.0.... [Settings] Home [/media/l... [External ... video_de... [* Untitle... [GNOME T... [napari] napari

Deconvolution

Interactive Deconvolution and Visualization with Napari





Blog post on using dask-cuda for blockwise multi-GPU computation

https://blog.dask.org/2020/11/12/deconvolution

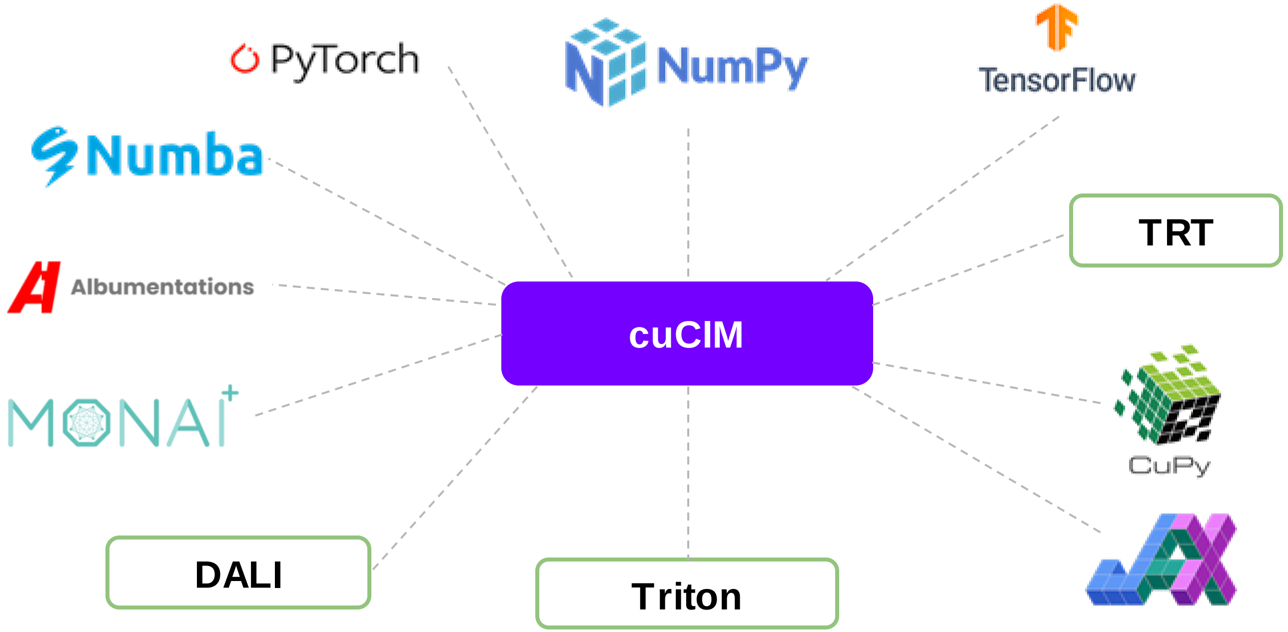
Getting Started with cuCIM

Remaining Challenges

- Improve scikit-image API coverage (currently about 2/3 of functions covered)
- Support additional image formats and filters
  - **Basic image formats:** Jpeg, Jpeg2000, PNG, BMP, etc.
  - **Digital Pathology**: Aperio (.svs), MIRAX (.mrxs), LEICA (.scn), tissue mask generation, stain normalization, etc.
  - **Radiology**: DICOM(.dcm), MetaIO(.mhd)
  - **Microscopy**: OME-Zarr in NGFF by Open Microscopy Environment (OME)
- Expand demos (help welcome!)

Remaining Challenges

- Interoperability with other libraries/frameworks



Get Started

http://github.com/rapidsai/cucim

```
$ conda install –c rapidsai –c nvidia –c conda-forge –c defaults cucim=21.xx.xx \
      python=3.8 cudatoolkit=11.y
```

example

https://github.com/rapidsai/cucim/tree/master/examples

https://github.com/rapidsai/cucim/tree/master/notebooks

https://github.com/grlee77/cucim-scipy2021-demos

Please give us your feedback!

https://github.com/rapidsai/cucim/issues

RAPIDS

Quansight
YOUR DATA EXPERTS

NVIDIA