benchmarking_radeonVII_20240307

March 7, 2024

1 Benchmarking numpy / scikit-image / scipy vs clesperanto

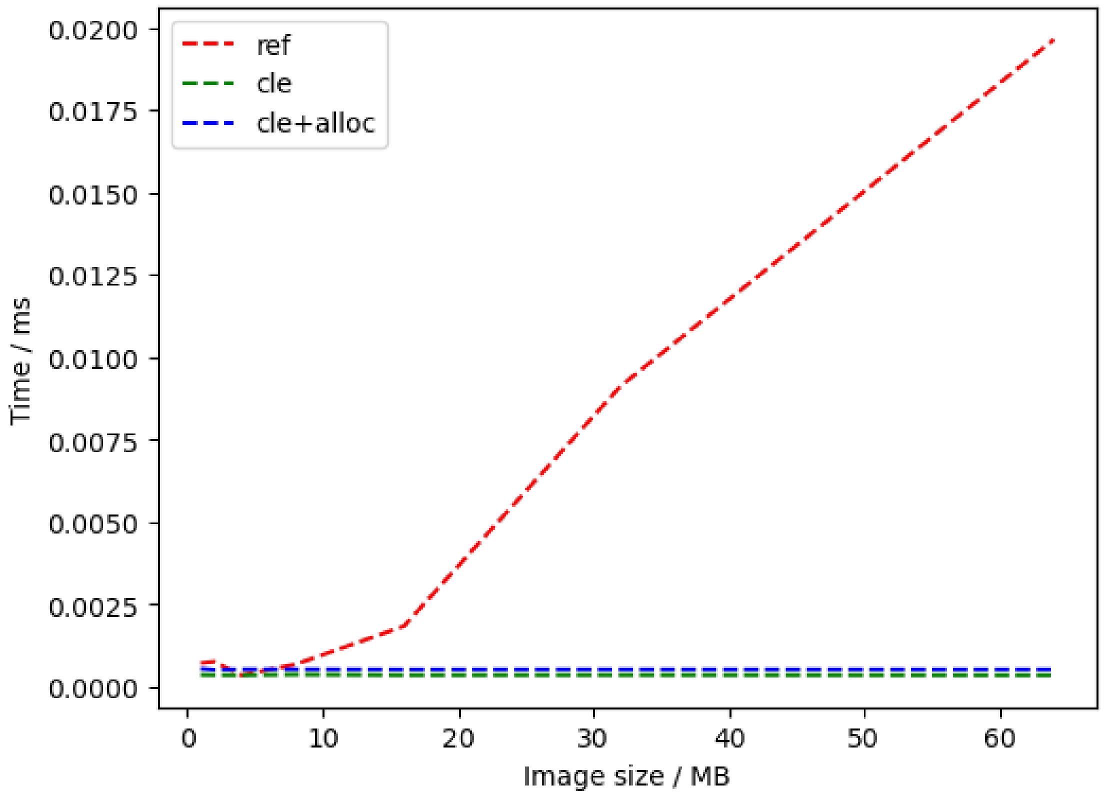
[ ]:
```
import pyclesperanto_prototype as cle

cle.select_device('906')
```

[ ]:
```
<gfx906:sramecc+:xnack- on Platform: AMD Accelerated Parallel Processing (2
refs)>
```

[ ]:
```
import numpy as np
import time
import matplotlib.pyplot as plt

num_iterations = 10

# measure execution time of a given method
def benchmark(function, kwargs):
    times = []
    for i in range(0, num_iterations):
        start_time = time.time()
        function(**kwargs)
        delta_time = time.time() - start_time
        times = times + [delta_time]
        # print(delta_time)

    # return median of measurements to ignore warmup-effects
    return np.median(times)



def benchmark_size(method_np, method_cle, method_cle_alloc):
    times_ref = []
    times_cle = []
    times_cle_alloc = []
    sizes = []
    for size in [1, 2, 4, 8, 16, 32, 64]:
        input1 = np.zeros((1024, 1024, size))
        cl_input1 = cle.push(input1)
        cl_input2 = cle.create(cl_input1.shape)

        time_ref = benchmark(method_np, {"image":input1})
        time_cle = benchmark(method_cle, {"image":cl_input1, "output":
 ↪cl_input2})
        time_cle_alloc = benchmark(method_cle_alloc, {"image":cl_input1})

        times_ref = times_ref + [time_ref]
        times_cle = times_cle + [time_cle]
        times_cle_alloc = times_cle_alloc + [time_cle_alloc]
        sizes = sizes + [size]

    plt.plot(sizes, times_ref,  'r--', sizes, times_cle, 'g--', sizes,␣
 ↪times_cle_alloc, 'b--');
    plt.ylabel('Time / ms')
    plt.xlabel('Image size / MB')
    plt.legend(("ref", "cle", "cle+alloc"));
    plt.show()


    print("\nSizes (MB)        " + str(sizes))
    print("Times ref (s)       " + str(np.round(times_ref, 4)))
    print("Times cle (s)       " + str(np.round(times_cle, 4)))
    print("Times cle+alloc (s) " + str(np.round(times_cle_alloc, 4)))
```

1.1 Thresholding

[ ]:
```
# RED: thresholding of a numpy array
def threshold_ref(image):
    thresholded = image > 100
    return thresholded

# GREEN: thresholding of a pre-existing opencl array (no push, pull or alloc)
def threshold_cle(image, output):
    cle.greater_constant(image, output, 100)

# BLUE: allocate result memory + thresholding
def threshold_cle_alloc(image):
    thresholded = cle.create(image.shape)
    cle.greater_constant(image, thresholded, 100)

benchmark_size(threshold_ref, threshold_cle, threshold_cle_alloc)
```

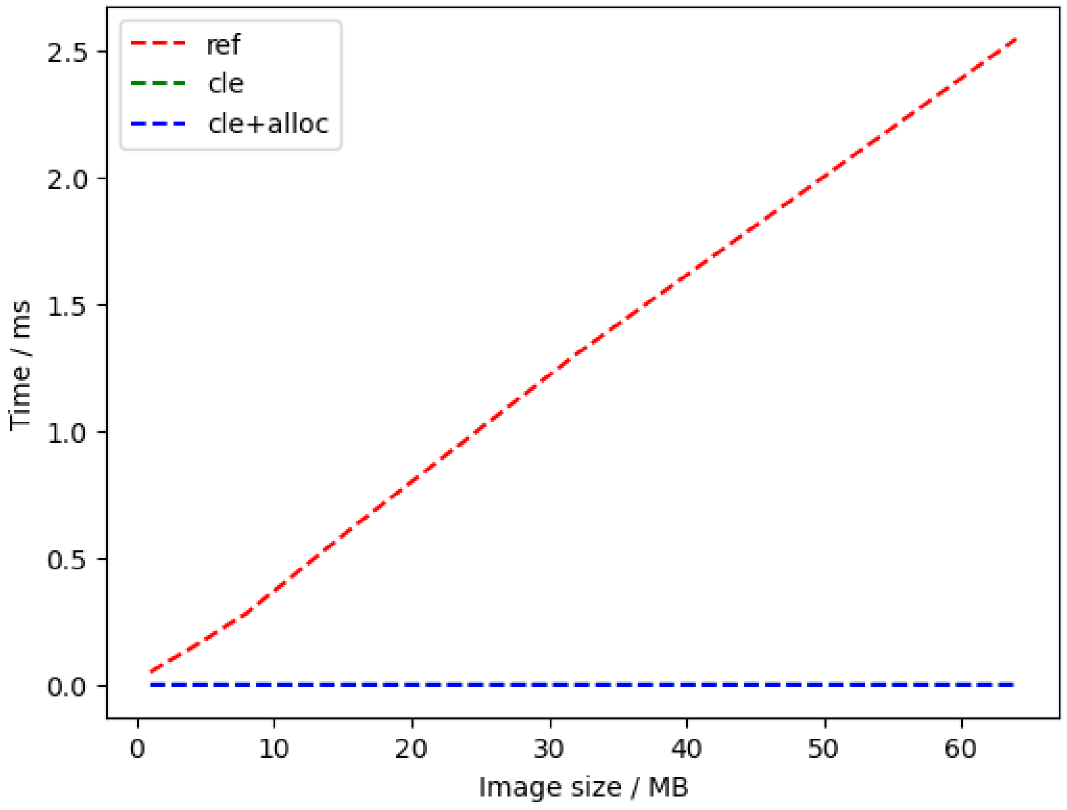


```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.0007 0.0007 0.0003 0.0007 0.0018 0.0091 0.0196]
Times cle (s)        [0.0003 0.0003 0.0003 0.0004 0.0003 0.0003 0.0003]
Times cle+alloc (s) [0.0005 0.0005 0.0005 0.0005 0.0005 0.0005 0.0005]
```

1.2 Gaussian blur radius 2

[ ]:
```
from skimage.filters import gaussian

radius = 2

def gaussian_blur_filter_ref(image):
    filtered = gaussian(image, sigma=radius)
    return filtered

def gaussian_blur_filter_cle(image, output):
    cle.gaussian_blur(image, output, radius, radius, radius)

def gaussian_blur_filter_cle_alloc(image):
    filtered = cle.create(image.shape)
    cle.gaussian_blur(image, filtered, radius, radius, radius)
```

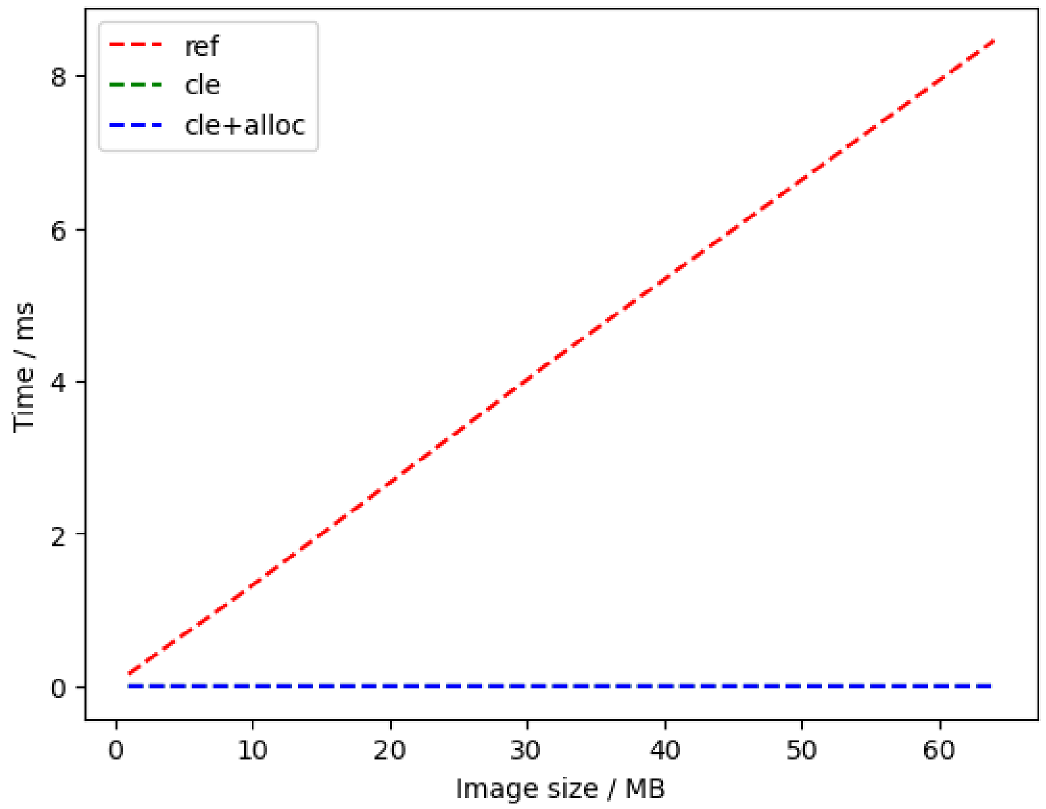
```
benchmark_size(gaussian_blur_filter_ref, gaussian_blur_filter_cle,␣
  ↪gaussian_blur_filter_cle_alloc)
```



```
Sizes (MB)        [1, 2, 4, 8, 16, 32, 64]
Times ref (s)       [0.0506 0.0845 0.1459 0.2832 0.6326 1.3032 2.5449]
Times cle (s)       [0.0013 0.0013 0.0014 0.0014 0.0014 0.0014 0.0014]
Times cle+alloc (s) [0.0014 0.0014 0.0014 0.0015 0.0014 0.0015 0.0015]
```

1.3 Gaussian blur radius 10

[ ]:
```
radius = 10
benchmark_size(gaussian_blur_filter_ref, gaussian_blur_filter_cle,␣
  ↪gaussian_blur_filter_cle_alloc)
```

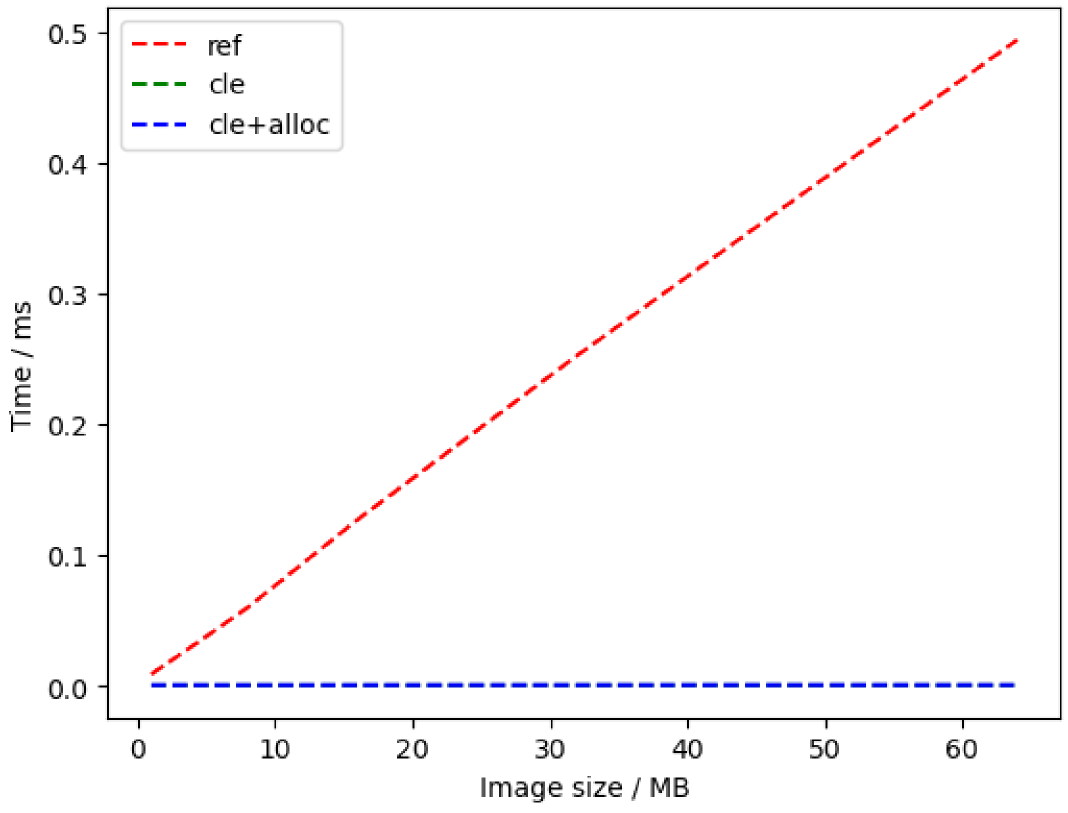


```
Sizes (MB)          [1, 2, 4, 8, 16, 32, 64]
Times ref (s)         [0.1601 0.2883 0.5476 1.0531 2.1222 4.2846 8.4555]
Times cle (s)         [0.0013 0.0013 0.0013 0.0013 0.0013 0.0014 0.0013]
Times cle+alloc (s) [0.0014 0.0015 0.0014 0.0014 0.0014 0.0015 0.0014]
```

1.4 Binary erosion

[ ]:
```
from skimage.morphology import binary_erosion


def binary_erosion_ref(image):
    filtered = binary_erosion(image)
    return filtered

def binary_erosion_cle(image, output):
    cle.erode_box(image, output)

def binary_erosion_cle_alloc(image):
    filtered = cle.create(image.shape)
    cle.erode_box(image, filtered)
```

```
benchmark_size(binary_erosion_ref, binary_erosion_cle, binary_erosion_cle_alloc)
```



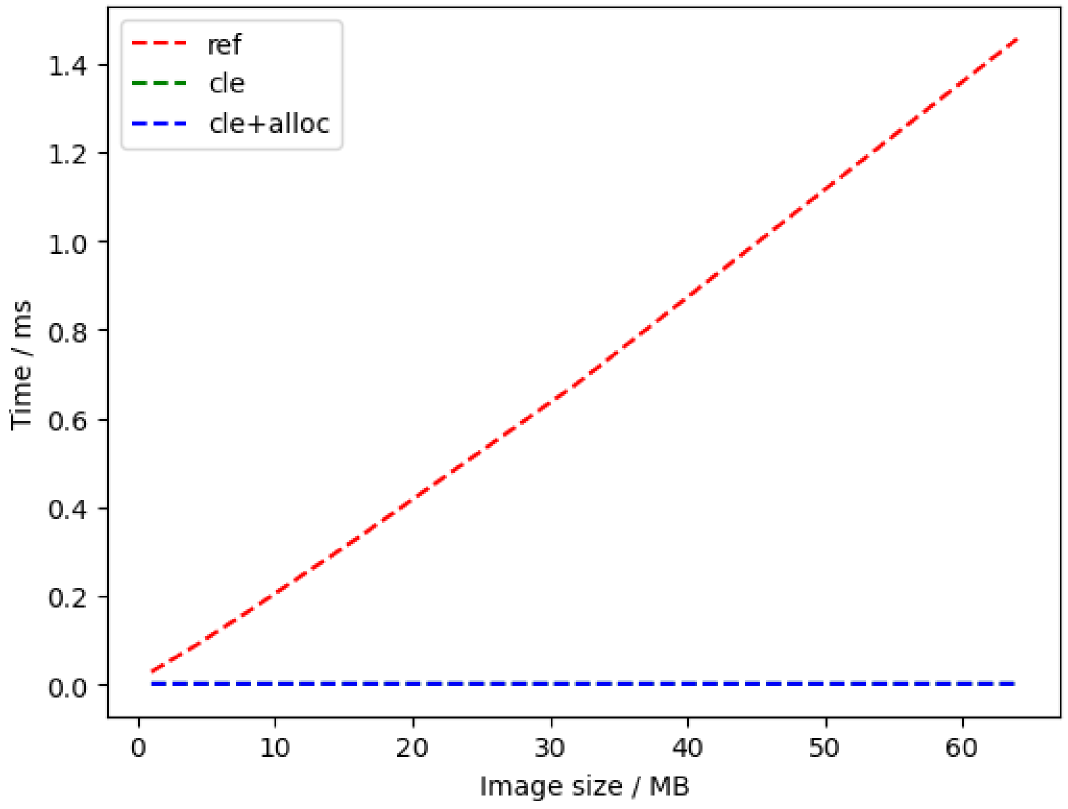
```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.0088 0.0161 0.0303 0.0597 0.1271 0.2528 0.4942]
Times cle (s)        [0.0003 0.0003 0.0003 0.0003 0.0003 0.0003 0.0003]
Times cle+alloc (s) [0.0005 0.0005 0.0004 0.0005 0.0005 0.0005 0.0005]
```

1.5 Mean filter radius=2

[ ]:
```
import scipy.ndimage.filters as spf


radius = 2
def mean_filter_ref(image):
    # todo: not sure if size is a radius or a diameter. Check documentation
    # https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.
↪uniform_filter.html#scipy.ndimage.uniform_filter
    filtered = spf.uniform_filter(image, size=radius)
    return filtered
```

```python
def mean_filter_cle(image, output):
    cle.mean_box(image, output, radius, radius, radius)

def mean_filter_cle_alloc(image):
    filtered = cle.create(image.shape)
    cle.mean_box(image, filtered, radius, radius, radius)

benchmark_size(mean_filter_ref, mean_filter_cle, mean_filter_cle_alloc)
```

```
/tmp/ipykernel_1029367/1961446685.py:8: DeprecationWarning: Please import
`uniform_filter` from the `scipy.ndimage` namespace; the `scipy.ndimage.filters`
namespace is deprecated and will be removed in SciPy 2.0.0.
  filtered = spf.uniform_filter(image, size=radius)
```



```
Sizes (MB)        [1, 2, 4, 8, 16, 32, 64]
Times ref (s)       [0.0283 0.0471 0.0833 0.1634 0.3297 0.6787 1.4553]
Times cle (s)       [0.0013 0.0013 0.0013 0.0013 0.0014 0.0013 0.0014]
Times cle+alloc (s) [0.0014 0.0014 0.0014 0.0014 0.0015 0.0014 0.0014]
```

```
[ ]:
```