benchmarking_nvidia_rtx_2080_200603

June 3, 2020

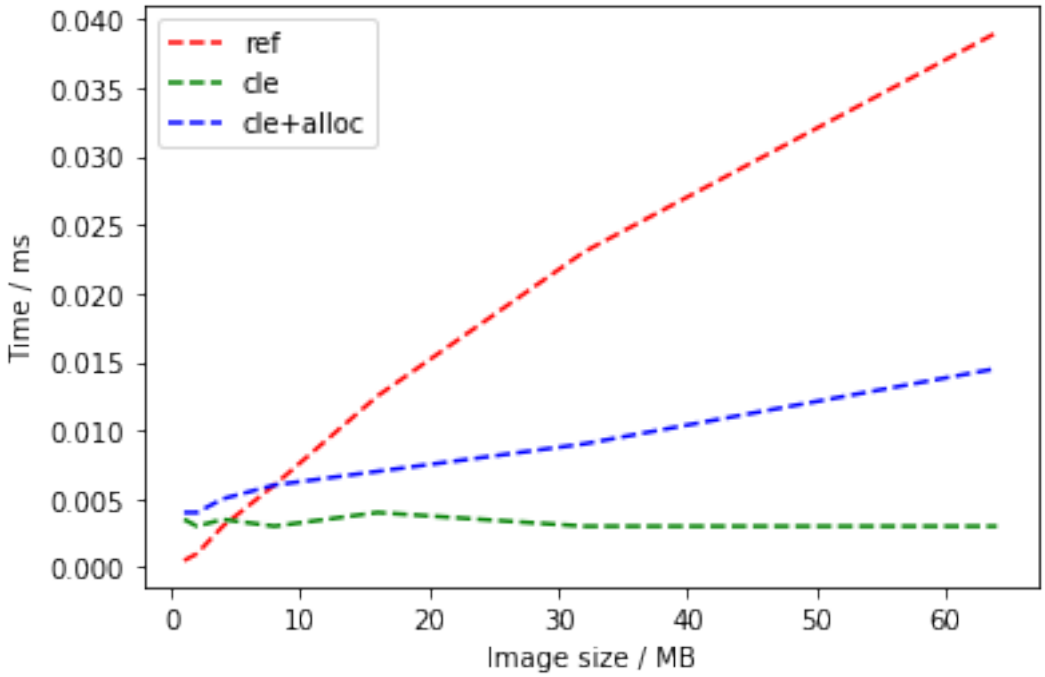
1 Benchmarking numpy / scikit-image / scipy vs clesperanto

[1]:
```
import clesperanto as cle
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

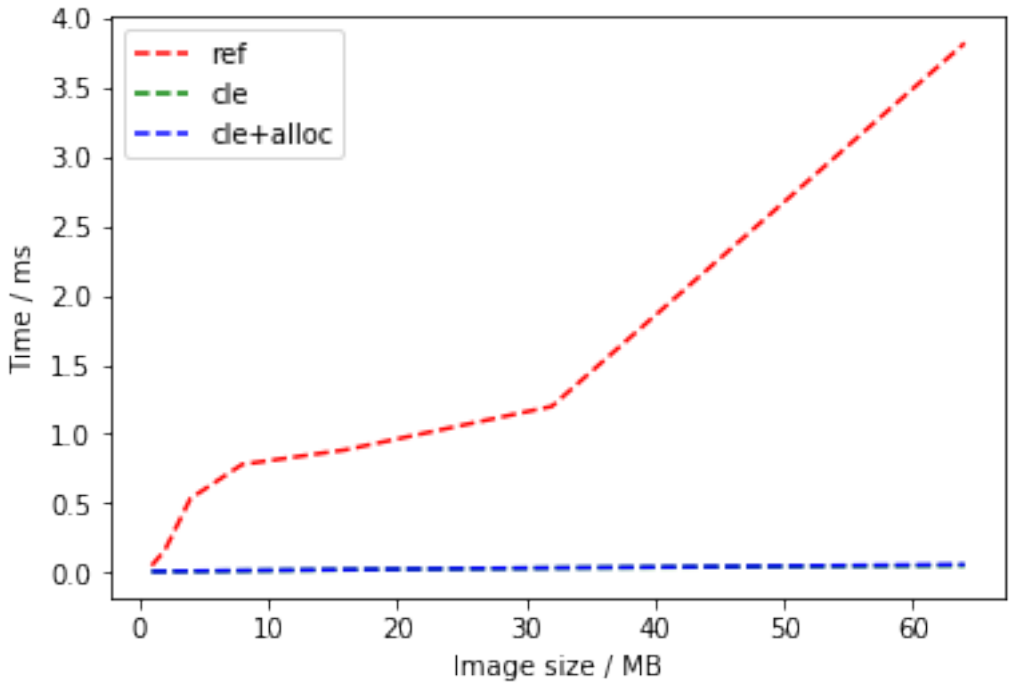
[2]:
```
# RED: thresholding of a numpy array
def threshold_ref(image):
    thresholded = image > 100
    return thresholded

# 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)        [0.0005 0.001  0.003  0.006  0.0125 0.023  0.039 ]
Times cle (s)        [0.0035 0.003  0.0035 0.003  0.004  0.003  0.003 ]
Times cle+alloc (s) [0.004  0.004  0.005  0.006  0.007  0.009  0.0145]
```

1.2 Gaussian blur radius 2

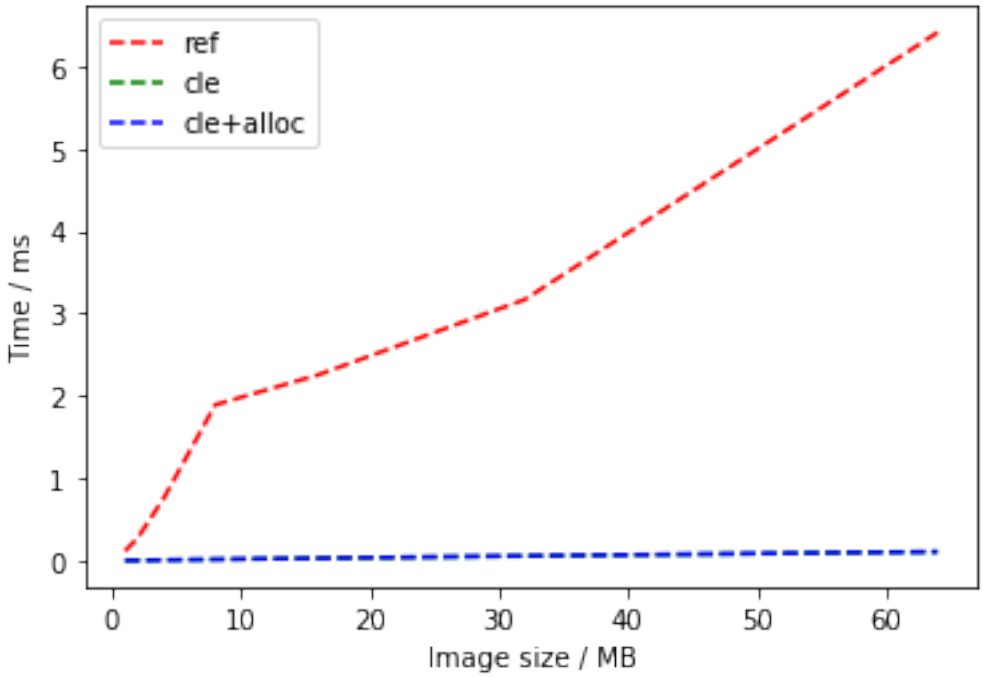
[3]:
```
from skimage.filters import gaussian

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)        [0.051  0.1581 0.534  0.7797 0.8848 1.1981 3.818 ]
Times cle (s)        [0.008  0.009  0.01   0.013  0.02   0.0335 0.052 ]
Times cle+alloc (s) [0.008  0.009  0.011  0.015  0.022  0.034  0.0581]
```

1.3 Gaussian blur radius 10

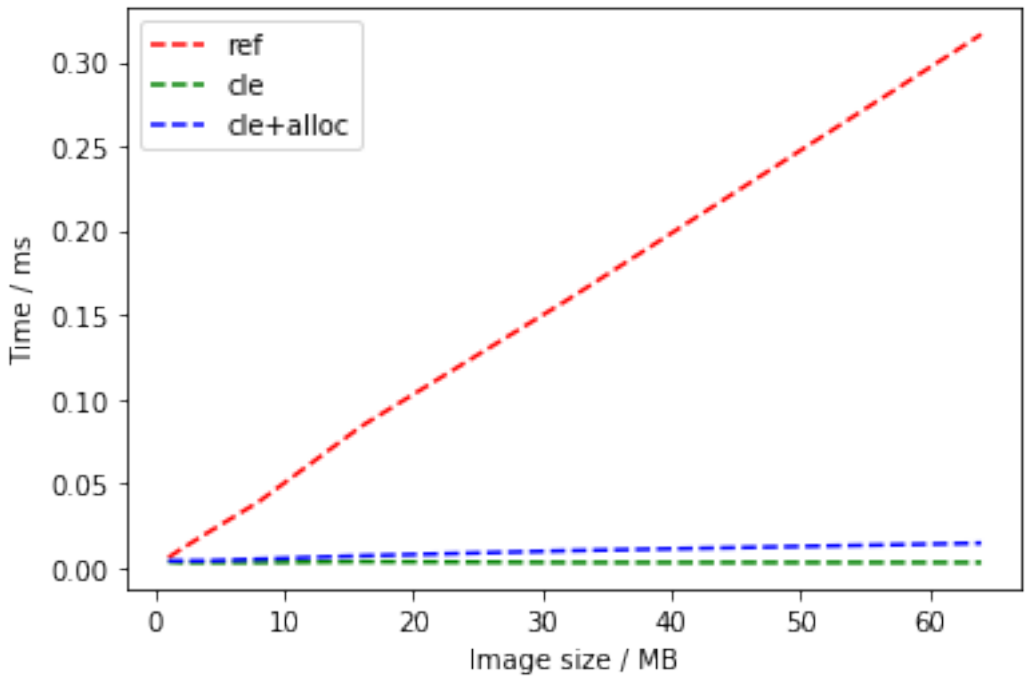
[4]:
```
radius = 10
benchmark_size(gaussian_blur_filter_ref, gaussian_blur_filter_cle,␣
↪gaussian_blur_filter_cle_alloc)
```




```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.1231 0.2858 0.7614 1.8947 2.2588 3.1704 6.4054]
Times cle (s)        [0.009  0.011  0.015  0.023  0.038  0.0621 0.1146]
Times cle+alloc (s) [0.009  0.0115 0.016  0.023  0.0371 0.0661 0.1176]
```

1.4 Binary erosion

[5]:
```
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)
```

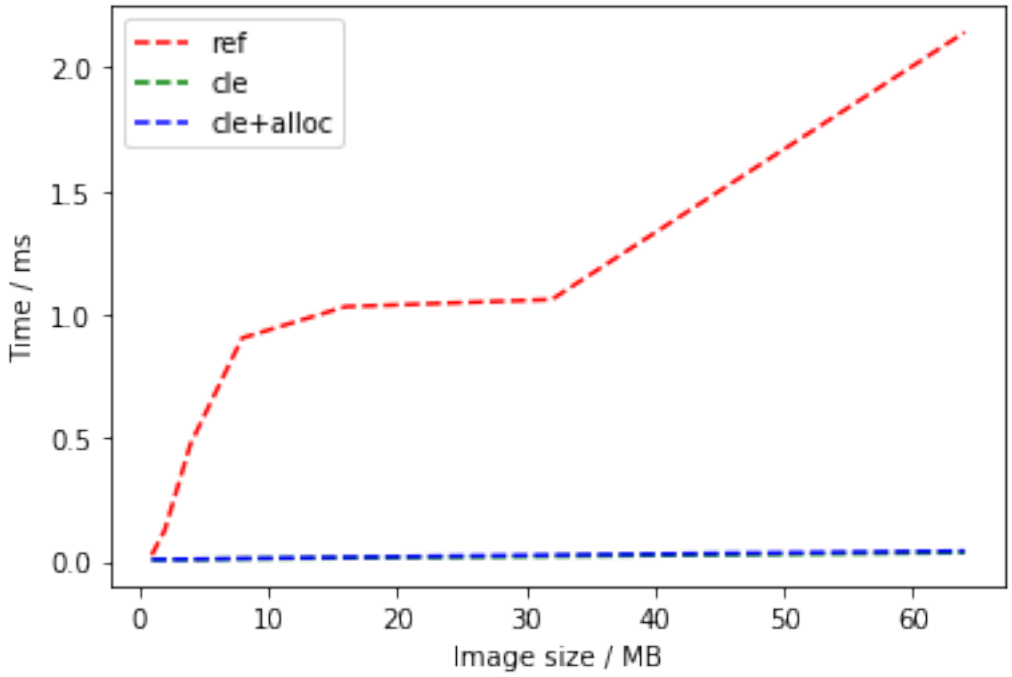



```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.006  0.011  0.0205 0.039  0.0841 0.1591 0.3163]
Times cle (s)        [0.0035 0.003  0.003  0.003  0.0035 0.003  0.003 ]
Times cle+alloc (s) [0.004  0.004  0.004  0.005  0.007  0.01   0.0145]
```

1.5 Mean filter radius=2

[6]:
```
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

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)
```




```
Sizes (MB)         [1, 2, 4, 8, 16, 32, 64]
Times ref (s)        [0.027  0.1261 0.4752 0.9048 1.0319 1.061  2.142 ]
Times cle (s)        [0.008 0.009 0.009 0.012 0.015 0.022 0.037]
Times cle+alloc (s) [0.009 0.009 0.01  0.013 0.018 0.027 0.043]
```

[ ]: