Open in Colab

Python + cuda

Let us repeat the previous exercises in python.

In [25]:
```
!nvidia-smi
```

```
Mon Nov  7 10:45:37 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   76C    P0    33W /  70W |   3178MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
```

In [26]:
```
from numba import cuda
from numba import jit
import numpy as np
from numba import vectorize, int32, int64, float32, float64
import matplotlib.pyplot as plt

%matplotlib inline

N = 2**26
x = np.arange(N, dtype=np.float64) # [0...N] on the host

print(f"Number of elements: {N} \nMemory size of array element in [MB]: {x.nbytes/1E6}")
```

```
Number of elements: 67108864
Memory size of array element in [MB]: 536.870912
```

Reduction

In [27]:
```
# reference: https://numba.pydata.org/numba-doc/dev/cuda/reduction.html

@cuda.reduce
def sum_reduce(a, b):
    return a + b

expect = x.sum()      # numpy sum reduction
got = sum_reduce(x)   # cuda sum reduction
assert expect == got
```

```
/usr/local/lib/python3.7/dist-packages/numba/cuda/dispatcher.py:488: NumbaPerformanceWarning: Grid s
ize 64 will likely result in GPU under-utilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
/usr/local/lib/python3.7/dist-packages/numba/cuda/cudadrv/devicearray.py:885: NumbaPerformanceWarnin
g: Host array used in CUDA kernel will incur copy overhead to/from device.
  warn(NumbaPerformanceWarning(msg))
/usr/local/lib/python3.7/dist-packages/numba/cuda/dispatcher.py:488: NumbaPerformanceWarning: Grid s
ize 1 will likely result in GPU under-utilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
```

In [28]:
```
#Lambda functions can also be used here:
sum_reduce_lam = cuda.reduce(lambda a, b: a + b)

expect = x.sum()      # numpy sum reduction
got = sum_reduce_lam(d_x)   # cuda sum reduction
assert expect == got
```

```
/usr/local/lib/python3.7/dist-packages/numba/cuda/dispatcher.py:488: NumbaPerformanceWarning: Grid s
ize 64 will likely result in GPU under-utilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
/usr/local/lib/python3.7/dist-packages/numba/cuda/dispatcher.py:488: NumbaPerformanceWarning: Grid s
ize 1 will likely result in GPU under-utilization due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
```

In [29]:
```
%timeit x.sum()    # NumPy on CPU
```

```
44.5 ms ± 346 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [30]:
```
%timeit sum_reduce_lam(x) # Numba on GPU - data from host
```

```
/usr/local/lib/python3.7/dist-packages/numba/cuda/cudadrv/devicearray.py:885: NumbaPerformanceWarnin
g: Host array used in CUDA kernel will incur copy overhead to/from device.
  warn(NumbaPerformanceWarning(msg))
251 ms ± 31.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [31]:
```
%timeit sum_reduce_lam(d_x) # Numba on GPU - prefetched data
```

```
2.57 ms ± 17.1 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

SAXPY

SAXPY stands for “Single-Precision A·X Plus Y”. It is a function in the standard Basic Linear Algebra Subroutines (BLAS) library.

In [32]:
```
a = 10
y = np.copy(x)

d_a = cuda.to_device(a) # Copy of a on the device
d_x = cuda.to_device(x) # Copy of x on the device
d_y = cuda.to_device(y) # Copy of y on the device
d_out = cuda.device_array_like(d_x) # Like np.array_like, but for device arrays
```

In [33]:
```
@vectorize(['float64(int64, float64, float64)'], target='cuda') # Type signature and target are req
def add_ufunc(a, x, y):
    return a*x + y
```

In [34]:
```
expect = a*x + y      # numpy sum reduction
got = add_ufunc(d_a, d_x, d_y)   # cuda sum reduction
# assert expect == got

np.allclose(expect, got)
```

Out[34]:
```
True
```