singleThread6M-mask_3array

July 10, 2022

1 pre-proc and CSC merge

[1]:
```
ncpu = 1
nthread = 1
import os, sys
os.environ['OMP_NUM_THREADS']="1"
print(os.environ['SLURM_CPUS_PER_TASK'])
```

```
1
```

[2]:
```
sys.path.insert(0, '/home/esrf/wright/git/pyFAI/build/lib.linux-x86_64-3.8/')
import numpy
import fabio
import pyFAI
from pyFAI.test.utilstest import UtilsTest
import pyFAI.azimuthalIntegrator
pyFAI
```

[2]:
```
<module 'pyFAI' from
'/home/esrf/wright/git/pyFAI/build/lib.linux-x86_64-3.8/pyFAI/__init__.py'>
```

[3]:
```
img = fabio.open(UtilsTest.getimage("Pilatus6M.cbf")).data
ai = pyFAI.load(UtilsTest.getimage("Pilatus6M.poni"))

ai.detector.set_mask( img == -1 )
dummyval = -2
deltaval = 0.5

npt = 1000
split = "full"
method = [split, "csr", "python"]
unit="r_mm"
ai
```

[3]:
```
Detector Pilatus 6M      PixelSize= 1.720e-04, 1.720e-04 m
Wavelength= 1.033200e-10m
SampleDetDist= 3.000000e-01m    PONI= 2.254060e-01, 2.285880e-01m
```

```
rot1=0.000000  rot2= 0.000000  rot3= 0.000000 rad
DirectBeamDist= 300.000mm       Center: x=1329.000, y=1310.500 pix
Tilt=0.000 deg  tiltPlanRotation= 0.000 deg
```

[4]:
```
ref_no_dummy = reference = ai.integrate1d(img, npt, unit=unit, method=method)
%timeit _ = ai.integrate1d(img, npt, unit=unit, method=method )
```

```
197 ms ± 1.51 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[5]:
```
reference = ai.integrate1d(img, npt, unit=unit, method=method, dummy=dummyval,␣
 ↪delta_dummy=deltaval)
%timeit _ = ai.integrate1d(img, npt, unit=unit, method=method, dummy=dummyval,␣
 ↪delta_dummy=deltaval)
```

```
200 ms ± 1.6 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[6]:
```
import numba
@numba.njit(nogil=True)
def masked_3sum_dummy( raw, normalization, static_mask, dummy, delta_dummy,
               indices, indptr, data, sum_1, sum_signal, sum_normalization,␣
 ↪dark=None):
    for i in range(raw.size):
        if static_mask.flat[i] > 0:
            continue
        signal = numpy.float32( raw.flat[i] )
        if abs(signal - dummy) < delta_dummy:
            continue
        norm32 = numpy.float32( normalization.flat[i] )
        if dark is not None:
            signal -= dark.flat[i]
        # pixel split here
        k = indices[ indptr[i] ]
        for j in range( indptr[i], indptr[i+1] ):
            sum_1[ k ] += data[ j ]                          # sum (1-mask)
            sum_signal[ k ] += signal * data[ j ]
            sum_normalization[ k ] += norm32 * data[ j ]
            k += 1

# These will need to be cached somewhere, and updated when necessary.
# first the csc matrix
csc = [j for i, j in ai.engines.items() if i.implementation=="python"][0].
 ↪engine._csr.tocsc()
# then the 2D image (also could need flat, absorbtion)
normalization_image = ai.polarization()*ai.solidAngleArray()
# then the mask
static_mask = ai.detector.get_mask().astype(numpy.uint8)
```

```
def allocate_and_run( img, norm, static_mask, dummy, delta_dummy, indices,␣
 ↪indptr, data, dark=None ):
    result = numpy.zeros( (3,npt), numpy.float32 )
    s1, ss, sn = result
    masked_3sum_dummy( img, norm, static_mask,
                           numpy.float32(dummy), numpy.float32(delta_dummy),
                           indices, indptr, data, s1, ss, sn, dark)
    return result


ret = allocate_and_run( img, normalization_image, static_mask, dummyval,␣
 ↪deltaval, csc.indices, csc.indptr, csc.data)

%timeit allocate_and_run( img, normalization_image, static_mask, dummyval,␣
 ↪deltaval, csc.indices, csc.indptr, csc.data )
```

```
44.8 ms ± 414 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[7]:
```
# it should match the reference, but not the no_dummy
for ref in ref_no_dummy, reference:
    print( numpy.allclose(ref.count, ret[0]),
           numpy.allclose(ref.sum_signal, ret[1]),
           numpy.allclose(ref.sum_normalization, ret[2]))
```

```
False False False
True True True
```

[ ]: