계산속도 향상을 위한 Python 확장

Python과 컴파일 언어들과의 결합

C

- ctypes
- CFFI (C Foreign Function Interface)
- Cython
- Python/C API

C++

- boost.python
- pybind11

FORTRAN

- f2py
- ctypes

CUDA (for NVIDIA GPU)

- PyCUDA

OpenCL (for AMD GPU, etc.)

- PyOpenCL

첫 번째 예제 (SAXPY)

Single precision $aX + Y$

Numpy 버전

saxpy.py

```
import numpy as np

def saxpy(a, x, y):
    y[:] = a*x + y


def main():
    n = 2**20
    a = np.random.rand()
    x = np.random.rand(n)
    y = np.random.rand(n)

    saxpy(a, x, y)


if __name__ == '__main__':
    main()
```

Python + C 버전

- ctypes와 numpy.ctypeslib 모듈 이용
- Pure C 코드를 그대로 사용 가능
- 컴파일된 shared library (.so) 파일을 Python에서 직접 호출함

saxpy_core.c

```
void saxpy(int n, float a, float *x, float *y) {
    int i;

    for (i=0; i<n; i++) {
        y[i] = a*x[i] + y[i];
    }
}
```

컴파일

```
$ gcc -O3 -c -fPIC saxpy_core.c -o saxpy_core.o
$ gcc -shared -o saxpy_core.so saxpy_core.o
```

saxpy_c.py

```
from ctypes import c_int, c_float
from datetime import datetime
from numpy.testing import assert_array_equal as a_equal
import numpy as np
import numpy.ctypeslib as npct


def saxpy_numpy(a, x, y):
    y = a*x + y


class SAXPY:
    def __init__(self):
        # load the library using numpy
        libm = npct.load_library('saxpy_core', './')
        self.saxpy_c = getattr(libm, 'saxpy')

        # set the arguments and return types
        arr_f4 = npct.ndpointer(ndim=1, dtype='f4')
        self.saxpy_c.argtypes = [c_int, c_float, arr_f4, arr_f4]
        self.saxpy_c.rettype = None


def main():
    n = 2**20
    a = np.random.rand()
    x = np.random.rand(n)
    y = np.random.rand(n)
    y2 = y.copy()

    t1 = datetime.now()
    saxpy_numpy(a, x, y)
    dt_np = datetime.now() - t1

    obj = SAXPY()
    t2 = datetime.now()
    obj.saxpy_c(n, a, x, y2)
    dt_c = datetime.now() - t2

```

```
40      print('n={}'.format(n))
41      print('numpy: {}'.format(dt_np))
42      print('c    : {}'.format(dt_c))
43
44      a_equal(y, y2)
45
46
47  if __name__ == '__main__':
48      main()
```

12번 라인의 SAXPY 클래스는 C 함수를 Python 함수로 맵핑해주는 역할을 한다.

15번 라인에서 numpy.ctypeslib.npct.load_library() 함수를 이용하여 saxpy_core.so 라이브러리 파일을 읽어들인다.

16번 라인에서 C 라이브러리에서 saxpy 함수를 saxpy_c 이름으로 맵핑한다.

19,20,21 라인들은 C 함수의 인자들과 반환값의 자료형을 맞춰주는데 **이 부분에서 에러가 발생하지 않도록 주의해야 한다**.

datetime 함수는 numpy 버전과 C 버전의 계산시간을 측정하기 위해 사용하였다.

44번 라인의 a_equal() 함수는 numpy.testing.assert_array_equal() 함수를 짧은 별칭으로 사용하는 것인데, 두 Numpy 배열의 값을 비교하여 하나라도 값이 다르면 에러를 발생시킨다. 에러가 발생하지 않았다면 두 배열의 값이 완전히 동일한 것이다.

계산 시간 비교

n의 크기를 적절히 조절하여 위 코드를 실행하면 numpy 버전과 C 함수 버전의 계산 시간 차이가 상당함을 확인할 수 있다.

```
1  $ python saxpy_c.py
2  n=1073741824
3  numpy: 0:00:07.355140
4  c    : 0:00:00.928299
```

두 번째 예제 - 2차원 파동 시뮬레이션

호수에 돌을 던지면 원형으로 퍼지는 물결파를 볼 수 있다. 2차원 파동 방정식을 풀면 이와 같은 원형 파동을 시뮬레이션 할 수 있다. 2차원 파동 방정식은 다음과 같이 시간과 공간에 대한 2차 미분으로 이루어져 있다.

$$\frac{\partial^2 u}{\partial t^2} = a^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

수치 미분을 위해 중심차분방법(Central Finite-Difference Method)를 이용하여 미분항을 이산화하면 다음과 같다.

$$\frac{\partial^2 u}{\partial t^2} \simeq \frac{u|_{i,j}^{n-1} - 2u|_{i,j}^{n} + u|_{i,j}^{n+1}}{\Delta_t^2}$$

$$\frac{\partial^2 u}{\partial x^2} \simeq \frac{u|_{i-1,j}^{n} - 2u|_{i,j}^{n} + u|_{i+1,j}^{n}}{\Delta_x^2}$$

윗첨자 n은 시간을 이산화한 인덱스로 n은 현재 타임스텝, $n-1$은 이전 타임스텝, $n+1$은 다음 타임스텝을 의미한다. 아래첨자 i, j는 2차원 공간을 이산화한 인덱스로 $i-1, i+1$들은 좌우 격자점들을, $j-1, j+1$들은 아래위 격자점들을 의미한다. 이 식들을 방정식에 대입하여 정리하면, 다음과 같다.

$$u|_{i,j}^{n+1} = \left(a \frac{\Delta_t}{\Delta_x} \right)^2 \left(u|_{i-1,j}^{n} + u|_{i+1,j}^{n} + u|_{i,j-1}^{n} + u|_{i,j+1}^{n} - 4u|_{i,j}^{n} \right) + 2u|_{i,j}^{n} - u|_{i,j}^{n-1}$$

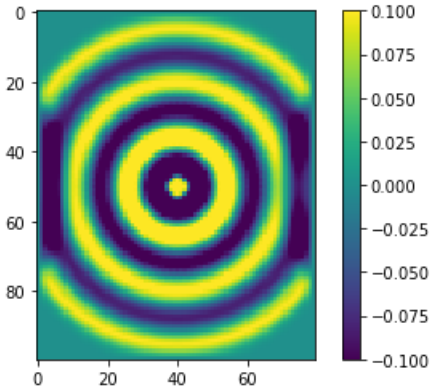
편의상 $\left(a\frac{\Delta_t}{\Delta_x}\right)^2 = 0.25$ 라고 하자.

Numpy 버전

wave2d.py

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt


def update(f, g):
    sl = slice(1, -1)
    f[sl,sl] = 0.25*(g[:-2,sl] + g[2:,sl] + g[sl,:-2] + g[sl,2:] - 4*g[sl,sl]) + 2*g[sl,sl] - f


def main():
    # setup
    nx, ny = 100, 80
    tmax = 50

    # allocation
    f = np.zeros((nx, ny), 'f4')
    g = np.zeros((nx, ny), 'f4')

    # plot
    imag = plt.imshow(f, vmin=-0.1, vmax=0.1)
    plt.colorbar()

    # time loop
    for tstep in range(1, tmax+1):
        g[nx//2,ny//2] = np.sin(0.4*tstep)
        update(f, g)
        update(g, f)

        '''
        if tstep%10 == 0:
            print('tstep={}'.format(tstep))
            imag.set_array(f)
            imag.savefig('png/wave2d_{:03d}.png'.format(tstep))
        '''

    imag.set_array(f)
    plt.show()


if __name__ == '__main__':
    main()
```



Python + C 버전

- ctypes와 numpy.ctypeslib 모듈 이용
- Pure C 코드를 그대로 사용 가능
- 컴파일된 shared library (.so) 파일을 Python에서 직접 호출함

wave2d_core.c

```
1  void update(int nx, int ny, float *f, float *g) {
2      int i, j, ij;
3
4      for (i=1; i<nx-1; i++) {
5          for (j=1; j<ny-1; j++) {
6              ij = i*ny + j;
7              f[ij] = 0.25*(g[(ij-ny] + g[(ij+ny] + g[ij-1] + g[ij+1] - 4*g[ij]) + 2*g[ij] -
   f[ij];
8          }
9      }
10 }
```

f와 g 함수는 본래 2차원 함수이지만, Python에서 이 함수를 호출할 때의 편의를 위해 1차원 함수로 바꾸었다. 7번 라인에서 2차원 배열을 1차원으로 길게 늘였을 때의 인덱스 변환은 다음과 같다.

f[i][j] -> f[i*ny + j]

f[i+1][j] -> f[(i+1)*ny + j] -> f[(i*ny + j) + ny]

f[i][j+1] -> f[i*ny + (j+1)] -> f[(i*ny + j) + 1]

컴파일

```
1  $ gcc -O3 -c -fPIC wave2d_core.c -o wave2d_core.o
2  $ gcc -shared -o wave2d_core.so wave2d_core.o
```

saxpy_c.py

```
1  from ctypes import c_int, c_float
2  from datetime import datetime
3  from numpy.testing import assert_array_equal as a_equal
4  import numpy as np
5  import numpy.ctypeslib as npct
6
7
8  def update_numpy(f, g):
9      sl = slice(1, -1)
10     f[sl,sl] = 0.25*(g[:-2,sl] + g[2:,sl] + g[sl,:-2] + g[sl,2:] - 4*g[sl,sl]) + 2*g[sl,sl] -
   f[sl,sl]
11
12
13 class WAVE2D:
14     def __init__(self):
15         # load the library using numpy
16         libm = npct.load_library('wave2d_core', './')
17         self.update_c = getattr(libm, 'update')
18
19         # set the arguments and return types
20         arr_f4 = npct.ndpointer(ndim=1, dtype='f4')
21         self.update_c.argtypes = [c_int, c_int, arr_f4, arr_f4]
22         self.update_c.rettype = None
23
24
25 def main():
26     # setup
```

```
27      nx, ny = 100, 80
28      tmax = 50
29
30      # allocation
31      f = np.zeros((nx, ny), 'f4')
32      g = np.zeros((nx, ny), 'f4')
33      f2 = np.zeros((nx, ny), 'f4')
34      g2 = np.zeros((nx, ny), 'f4')
35
36      # time loop
37      t1 = datetime.now()
38      for tstep in range(1, tmax+1):
39          g[nx//2,ny//2] = np.sin(0.4*tstep)
40          update(f, g)
41          update(g, f)
42      dt_np = datetime.now() - t1
43
44      t2 = datetime.now()
45      for tstep in range(1, tmax+1):
46          g2[nx//2,ny//2] = np.sin(0.4*tstep)
47          update(nx, ny, f2.ravel(), g2.ravel())
48          update(nx, ny, g2.ravel(), f2.ravel())
49      dt_c = datetime.now() - t2
50
51      print('nx={}, ny={}, tmax={}'.format(nx, ny, tmax))
52      print('numpy: {}'.format(dt_np))
53      print('c    : {}'.format(dt_c))
54
55      a_equal(f, f2)
56      a_equal(g, g2)
57
58
59  if __name__ == '__main__':
60      main()
```

C 함수 라이브러리에 선언된 update() 함수를 WAVE2D 클래스의 update_c() 함수로 맵핑하는 방법은 첫 번째 예제와 거의 동일하다.

47, 48번 라인에서 f2, g2를 인자로 넘겨줄 때, 2차원 배열을 1차원 배열로 변환해주는 ravel() 함수를 사용하였다. Numpy 다차원 배열을 1차원 배열로 바꿔주는 함수는 flatten()과 ravel() 두 가지가 있는데, flatten()은 copy를 하며 ravel()은 view를 한다. 이런 경우에는 ravel() 함수가 성능에 더 유리하다.

결과를 그래프로 확인하는 부분은 코드 분량 상 삭제하였다.