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_numpy.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.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 -shared -fPIC -o saxpy.so saxpy.c
```

main.py

```
from ctypes import c_int, c_float
from datetime import datetime

import numpy as np
import numpy.ctypeslib as npct
from numpy.testing import assert_array_equal as a_equal



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


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

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

        # set public
        self.libm = libm

    def saxpy_c(self, n, a, x, y):
        self.libm.saxpy(n, a, x, y)


def main():
    n = 2**25
    a = np.float32(np.random.rand())
    x = np.random.rand(n).astype('f4')
    y = np.random.rand(n).astype('f4')
    y2 = y.copy()

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

```
41
42      obj = SAXPY_C()
43      t2 = datetime.now()
44      obj.saxpy_c(n, a, x, y2)
45      dt_c = datetime.now() - t2
46
47      print('n={}'.format(n))
48      print('numpy: {}'.format(dt_numpy))
49      print('c    : {}'.format(dt_c))
50
51      a_equal(y, y2)
52      print('Check result: OK!')
53
54
55  if __name__ == '__main__':
56      main()
```

14번 라인의 SAXPY_C 클래스는 C 함수를 Python 함수로 맵핑해주는 역할을 한다.

17번 라인에서 numpy.ctypeslib.npct.load_library() 함수를 이용하여 saxpy.so 라이브러리 파일을 읽어들인다.

20,21,22 라인들은 C 함수의 인자들과 반환값의 자료형을 맞춰주는데 **이 부분에서 에러가 발생하지 않도록 주의해야 한다**.

datetime 함수는 numpy 버전과 C 버전의 계산시간을 측정하기 위해 사용하였다.

51번 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

현재 n의 크기가 크지 않`아서 계산 시간이 짧지만, 그럼에도 불구하고 numpy 버전과 C 함수 버전의 계산 시간이 꽤 차이남을 확인할 수 있다.

```
1  $ python main.py
2  n=1073741824
3  numpy: 0:00:00.105556
4  c    : 0:00:00.023172
5  Check result: OK!
```

Python + CUDA 버전

- PyCUDA 모듈 이용
- Pure CUDA-C 코드를 그대로 사용 가능
- NVIDIA GPU에서 실행

saxpy.cu

```
1  __global__ void saxpy(int n, float a, float *x, float *y) {
2      int idx = blockIdx.x*blockDim.x + threadIdx.x;
3      if (idx >= n) return;
4
5      y[idx] = a*x[idx] + y[idx];
6  }
```

main.py

```
1  from datetime import datetime
2  import atexit
3
```

```
import numpy as np
import pycuda.driver as cuda
from pycuda.compiler import SourceModule
from numpy.testing import assert_array_equal as a_equal
from numpy.testing import assert_array_almost_equal as aa_equal



#---------------------------------------------------------------------------
# CUDA initialize
#---------------------------------------------------------------------------
cuda.init()
device = cuda.Device(0)
context = device.make_context()
atexit.register(context.pop)

# CUDA info
print("CUDA Compute Capability: {}.{}".format(*device.compute_capability()))
print("Device: {}".format(device.name()))
#---------------------------------------------------------------------------



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



class SAXPY_CUDA(object):
    def __init__(self):
        # read a CUDA kernel file
        with open('saxpy.cu', 'r') as f:
            mod = SourceModule(f.read())
            self.saxpy = mod.get_function('saxpy')


    def saxpy_cuda(self, n, a, x_gpu, y_gpu):
        self.saxpy(np.int32(n),
                   np.float32(a),
                   x_gpu,
                   y_gpu,
                   block=(512,1,1), grid=(n//512+1,1))



def main():
    n = 2**25

    a = np.float32(np.random.rand())
    x = np.random.rand(n).astype('f4')
    y = np.random.rand(n).astype('f4')
    x_gpu = cuda.to_device(x)
    y_gpu = cuda.to_device(y)
    y2 = np.zeros(n, 'f4')

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

    obj = SAXPY_CUDA()
    t2 = datetime.now()
    obj.saxpy_cuda(n, a, x_gpu, y_gpu)
    cuda.memcpy_dtoh(y2, y_gpu)
    dt_cuda = datetime.now() - t2

    print('n={}'.format(n))
    print('numpy: {}'.format(dt_numpy))
    print('cuda : {}'.format(dt_cuda))

    aa_equal(y, y2, 7)
```

```
74      print('Check result: OK!')
75
76
77
78  if __name__ == '__main__':
79      main()
80
```

15-18 라인은 CUDA 실행 환경을 구성한다. 만약 시스템에 2개 이상의 NVIDIA GPU가 있다면, 16번 라인에서 cuda.Device(0) 에 0 대신 다른 번호를 넣어주어 활성화할 GPU를 선택할 수 있다.

32번 라인의 SAXPY_CUDA 클래스는 CUDA 함수를 Python 함수로 맵핑해주는 역할을 한다. 앞의 C 버전에서는 미리 컴파일된 라이브러리를 읽어왔는데, 여기서는 CUDA 커널 소스 코드(saxpy.cu)를 문자열로 읽어와서, 36번 라인의 SourceModule() 함수의 인자로 주면 런타임 때 CUDA 컴파일을 한다.

40-45번 라인은 CUDA 커널 saxpy를 호출하는데, 인자 n과 a의 자료형을 32bit int, float 형으로 변환해주고 있다. block 인자는 CUDA의 thread block 크기를 지정하고, grid 인자는 CUDA의 grid 크기를 지정한다.

55-56 라인에서 GPU 메모리 공간에 할당될 배열을 선언한다.

73번 라인의 aa_equal() 함수는 numpy.testing.assert_array_almost_equal() 함수를 짧은 별칭으로 사용하는 것인데, 두 Numpy 배열의 값을 비교하여 소수점 이하 7(세번째 인자) 자리의 값이 다르면 에러를 발생시킨다. 에러가 발생하지 않았다면 두 배열의 값은 소수점 이하 7자리까지 동일한 것이다.

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.

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

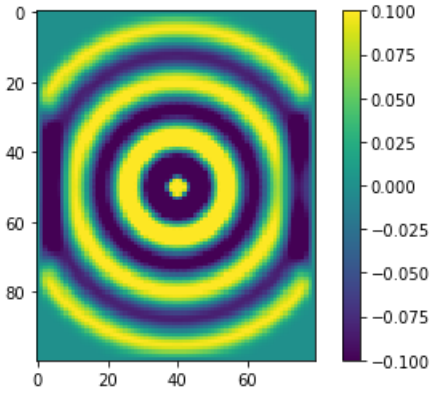
편의상 $\left(a \frac{\Delta_t}{\Delta_x} \right)^2 = 0.25$ 라고 하자.

Numpy 버전

wave2d_numpy.py

In [1]:

```
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]) W
               + 2*g[sl,sl] - f[sl,sl]


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.c

```
void update(int nx, int ny, float *f, float *g) {
    int i, j, idx;

    for (i=1; i<nx-1; i++) {
        for (j=1; j<ny-1; j++) {
            idx = i*ny + j;
            f[idx] = 0.25*(g[idx-ny] + g[idx+ny] + g[idx-1] + g[idx+1] - 4*g[idx])
                    + 2*g[idx] - f[idx];
        }
    }
}
```

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

```
gcc -O3 -shared -fPIC -o update.so update.c
```

main.py

```
from ctypes import c_int, c_float
from datetime import datetime

import numpy as np
import numpy.ctypeslib as npct
import matplotlib.pyplot as plt
from numpy.testing import assert_array_equal as a_equal
from numpy.testing import assert_array_almost_equal as aa_equal



def update_numpy(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[sl,sl]



class WAVE2D_C(object):
    def __init__(self):
        # load the library using numpy
        libm = npct.load_library('update', './')

        # set the arguments and retun types
        arr_f4 = npct.ndpointer(ndim=1, dtype='f4')
        libm.update.argtypes = [c_int, c_int, arr_f4, arr_f4]
        libm.update.rettype = None

        # set public
        self.libm = libm

    def update_c(self, nx, ny, x, y):
        self.libm.update(nx, ny, x, y)



```

```
36 def main():
37     nx, ny = 1000, 800
38     tmax = 500
39 
40     # allocation
41     f = np.zeros((nx, ny), 'f4')
42     g = np.zeros((nx, ny), 'f4')
43     f2 = np.zeros_like(f)
44     g2 = np.zeros_like(f)
45 
46     # time loop
47     # numpy version
48     t1 = datetime.now()
49     for tstep in range(1, tmax+1):
50         g[nx//2, ny//2] = np.sin(0.1*tstep)
51         update_numpy(f, g)
52         update_numpy(g, f)
53     dt_numpy = datetime.now() - t1
54 
55     # C version
56     obj = WAVE2D_C()
57     t2 = datetime.now()
58     for tstep in range(1, tmax+1):
59         g2[nx//2, ny//2] = np.sin(0.1*tstep)
60         obj.update_c(nx, ny, f2.ravel(), g2.ravel())
61         obj.update_c(nx, ny, g2.ravel(), f2.ravel())
62     dt_cuda = datetime.now() - t2
63 
64     print('Wnnx={}, ny={}, tmax={}'.format(nx, ny, tmax))
65     print('numpy: {}'.format(dt_numpy))
66     print('cuda : {}'.format(dt_cuda))
67 
68     # check results
69     aa_equal(f, f2, 6)
70     print('Check result: OK!')
71 
72     # plot
73     plt.imshow(f.T, cmap='hot', origin='lower', vmin=-0.1, vmax=0.1)
74     plt.colorbar()
75     plt.show()
76 
77 
78 
79 if __name__ == '__main__':
80     main()
```

C 함수 라이브러리에 선언된 update() 함수를 WAVE2D_C 클래스의 update_c() 함수로 맵핑하는 방법은 첫 번째 예제와 거의 동일하다.

60-61번 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.

Python + CUDA 버전

- PyCUDA 모듈 이용
- Pure CUDA-C 코드를 그대로 사용 가능
- NVIDIA GPU에서 실행

update.cu

```
1 __global__ void update(int nx, int ny, float *f, float *g) {
2     int idx = blockIdx.x*blockDim.x + threadIdx.x;
3     int i, j;
```

```
4
5       i = idx/ny;
6       j = idx%ny;
7       if (i > 0 && i < nx-1 && j > 0 && j < ny-1) {
8           f[idx] = 0.25*(g[idx-ny] + g[idx+ny] + g[idx-1] + g[idx+1] - 4*g[idx])
9                    + 2*g[idx] - f[idx];
10      }
11  }
12
13
14
15  __global__ void update_src(int nx, int ny, int tstep, float *g) {
16      g[(nx/2)*ny + (ny/2)] = sin(0.1*tstep);
17  }
```

CUDA 커널에서는 for 루프를 명시적으로 사용하지 않으므로, i, j 인덱스의 유효 범위를 조건문으로 넣어주어야 한다(7번 라인). 그렇지 않으면 런타임 에러가 나지는 않지만, 결과가 모두 nan 으로 나오게 된다.

15-17번 라인은 소스 항을 추가하는 커널이다. 앞의 코드들에서는 main() 함수에서 직접 처리했지만, CUDA에서는 독립적인 커널을 만들어줘야 한다.

main.py

```
1   from datetime import datetime
2   import atexit
3
4   import numpy as np
5   import matplotlib.pyplot as plt
6   import pycuda.driver as cuda
7   from pycuda.compiler import SourceModule
8   from numpy.testing import assert_array_equal as a_equal
9   from numpy.testing import assert_array_almost_equal as aa_equal
10
11
12
13  #---------------------------------------------------------------------------
14  # CUDA initialize
15  #---------------------------------------------------------------------------
16  cuda.init()
17  device = cuda.Device(0)
18  context = device.make_context()
19  atexit.register(context.pop)
20
21  # CUDA info
22  print("CUDA Compute Capability: {}.{}".format(*device.compute_capability()))
23  print("Device: {}".format(device.name()))
24  #---------------------------------------------------------------------------
25
26
27
28  def update_numpy(f, g):
29      sl = slice(1, -1)
30      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]
31
32
33
34  class WAVE2D_CUDA(object):
35      def __init__(self):
36          # read a CUDA kernel file
37          with open('update.cu', 'r') as f:
38              mod = SourceModule(f.read())
39              self.update = mod.get_function('update')
40              self.update_src = mod.get_function('update_src')
41
42
43      def update_cuda(self, nx, ny, f_gpu, g_gpu):
```

```
44          self.update(np.int32(nx),
45                      np.int32(ny),
46                      f_gpu,
47                      g_gpu,
48                      block=(512,1,1), grid=((nx*ny)//512+1,1))
49
50
51      def update_src_cuda(self, nx, ny, tstep, g_gpu):
52          self.update_src(np.int32(nx),
53                          np.int32(ny),
54                          np.int32(tstep),
55                          g_gpu,
56                          block=(1,1,1), grid=(1,1))
57
58
59
60  def main():
61      nx, ny = 1000, 800
62      tmax = 500
63
64      # allocation
65      f = np.zeros((nx, ny), 'f4')
66      g = np.zeros((nx, ny), 'f4')
67      f_gpu = cuda.to_device(f)
68      g_gpu = cuda.to_device(g)
69      f2 = np.zeros((nx,ny), 'f4')
70
71      # time loop
72      # numpy version
73      t1 = datetime.now()
74      for tstep in range(1, tmax+1):
75          g[nx//2, ny//2] = np.sin(0.1*tstep)
76          update_numpy(f, g)
77          update_numpy(g, f)
78      dt_numpy = datetime.now() - t1
79
80      # cuda version
81      obj = WAVE2D_CUDA()
82      t2 = datetime.now()
83      for tstep in range(1, tmax+1):
84          obj.update_src_cuda(nx, ny, tstep, g_gpu)
85          obj.update_cuda(nx, ny, f_gpu, g_gpu)
86          obj.update_cuda(nx, ny, g_gpu, f_gpu)
87      cuda.memcpy_dtoh(f2, f_gpu)
88      dt_cuda = datetime.now() - t2
89
90      print('Wnnx={}, ny={}, tmax={}'.format(nx, ny, tmax))
91      print('numpy: {}'.format(dt_numpy))
92      print('cuda : {}'.format(dt_cuda))
93
94      # check results
95      aa_equal(f, f2, 6)
96      print('Check result: OK!')
97
98      # plot
99      plt.imshow(f.T, cmap='hot', origin='lower', vmin=-0.1, vmax=0.1)
100     plt.colorbar()
101     plt.show()
102
103
104
105 if __name__ == '__main__':
106     main()
```

CUDA 환경을 구성하고, 커널 소스 파일을 읽어와서 컴파일 하는 부분은 앞의 saxpy 예제와 거의 유사하다.

43-56번 라인에서 CUDA 커널들을 호출할 때, 인자들의 자료형을 명시적으로 변환해줘야 하는 것에 주의해야 한다.

계산 시간 비교

```
$ python main.py
CUDA Compute Capability: 6.0
Device: Tesla P100-PCIE-16GB

nx=1000, ny=800, tmax=500
numpy: 0:00:05.872658
cuda : 0:00:00.039346
Check result: OK!
```