EECS6895 Adv. Bigdata Analytics HW 3 - Sun-Yi Lin(sl3833)

Amazon EC2 + PyCUDA

Since I don't have a device with nVidia Graphic Card, so I choose to use Amazon AWS's Elastic Compute Cloud service plus PyCUDA to use GPU calculation.

Arc Length Calculation

The first algorithm, I decided to calculate the arc length that we used in our web front-end application. In our application, the calculation of arc length was completed by the D3 library, but the basic idea is simple: get the summation all numbers of reviews up, and divided each business's reviews with the sum, then multiply with the length of the whole circle.

So I wrote the Python code as follow and run in my EC2 instance. In this code, I just use a little part of original dataset since the capacity of GPU is limited.

```
import pycuda.gpuarray as gpuarray
import pycuda.driver as cuda
import pycuda.tools, pycuda.autoinit, pycuda.compiler
import numpy, math

start = cuda.Event()
end = cuda.Event()
d = [2.0, 3.0, 3.0, 3.0, 13.0, 5.0, 20.0, 5.0, 19.0, 12.0, 10.0, 17.0, 6.0, 2.0,
3.0, 2.0, 39.0, 14.0, 2.0, 31.0, 113.0, 33.0, 74.0, 61.0, 23.0, 21.0, 12.0, 4.0,
300.0, 8.0, 2.0, 89.0, 25.0, 2.0, 39.0, 7.0, 9.0, 2.0, 52.0, 39.0, 3.0, 4.0,
78.0, 2.0, 3.0, 5.0, 10.0, 10.0, 9.0, 4.0, 4.0, 19.0, 3.0, 2.0, 84.0, 43.0,
44.0, 2.0, 2.0, 3.0, 5.0, 25.0, 2.0, 16.0, 34.0, 4.0, 13.0, 68.0, 3.0, 2.0, 2.0,
25.0, 77.0, 5.0, 15.0, 5.0, 4.0, 4.0, 8.0, 10.0, 7.0, 8.0, 46.0, 27.0, 2.0, 8.0,
15.0, 10.0, 2.0, 61.0, 16.0, 99.0, 2.0, 13.0, 27.0, 36.0, 2.0, 5.0, 5.0, 8.0]
r = 500
a_gpu = gpuarray.to_gpu(numpy.array(d))

# Run on CPU
start.record()
r_cpu = 2 * math.pi * r / sum(d)
c_cpu = numpy.array([r_cpu * x for x in d])
end.record()
end.synchronize()

print "-" * 80
print "Coefficient of CPU operation:",
```

```
print r_cpu
print "CPU time: %fs" %(start.time_till(end) * 1e-3)
print c_cpu

# Run on GPU
start.record()
r_gpu = 2 * math.pi * r / gpuarray.sum(a_gpu).get()
c_gpu = (r_gpu * a_gpu).get()
end.record()
end.synchronize()

print "-" * 80
print "Coefficient of GPU operation:",
print r_gpu
print "GPU time: %fs" %(start.time_till(end) * 1e-3)
print c_gpu

print "-" * 80
print "CPU-GPU difference:"
print c_cpu - c_gpu

numpy.allclose(c_cpu, c_gpu)
```

The result is as follow. It's very interesting that the GPU is very much slower ten CPU in this calculation. Perhaps the large scale of array effect the speed of GPU when it take a lot of time for copying data for host(CPU) to device(GPU), and, in the other hand, maybe this calculation couldn't take any advantage from GPU's ability.

But anyway, this two calculations get the same result.

```
ubuntu@ip-172-31-61-7:~/HW3$ python Example_1.py
--------------------------------------------------------------------------------
Coefficient of CPU operation: 1.43714211052
CPU time: 0.000035s
[   2.87428422    4.31142633    4.31142633    4.31142633   18.68284744
    7.18571055   28.74284221    7.18571055   27.3057001    17.24570533
   14.37142111   24.43141588    8.62285266    2.87428422    4.31142633
    2.87428422   56.04854231   20.11998955    2.87428422   44.55140543
  162.39705849   47.42568965  106.34851618   87.66566874   33.05426854
   30.17998432   17.24570533    5.74856844  431.14263316   11.49713688
    2.87428422  127.90564784   35.92855276    2.87428422   56.04854231
   10.05999477   12.93427899    2.87428422   74.73138975   56.04854231
    4.31142633    5.74856844  112.09708462    2.87428422    4.31142633
    7.18571055   14.37142111   14.37142111   12.93427899    5.74856844
    5.74856844   27.3057001     4.31142633    2.87428422  120.71993728
   61.79711075   63.23425286    2.87428422    2.87428422    4.31142633
```

```
    7.18571055   35.92855276    2.87428422   22.99427377   48.86283176
    5.74856844   18.68284744   97.72566352    4.31142633    2.87428422
    2.87428422   35.92855276  110.65994251    7.18571055   21.55713166
    7.18571055    5.74856844    5.74856844   11.49713688   14.37142111
   10.05999477   11.49713688   66.10853708   38.80283698    2.87428422
   11.49713688   21.55713166   14.37142111    2.87428422   87.66566874
   22.99427377  142.27706894    2.87428422   18.68284744   38.80283698
   51.73711598    2.87428422    7.18571055    7.18571055   11.49713688]
--------------------------------------------------------------------------------
Coefficient of GPU operation: 1.43714211052
GPU time: 0.531286s
[   2.87428422    4.31142633    4.31142633    4.31142633   18.68284744
    7.18571055   28.74284221    7.18571055   27.3057001    17.24570533
   14.37142111   24.43141588    8.62285266    2.87428422    4.31142633
    2.87428422   56.04854231   20.11998955    2.87428422   44.55140543
  162.39705849   47.42568965  106.34851618   87.66566874   33.05426854
   30.17998432   17.24570533    5.74856844  431.14263316   11.49713688
    2.87428422  127.90564784   35.92855276    2.87428422   56.04854231
   10.05999477   12.93427899    2.87428422   74.73138975   56.04854231
    4.31142633    5.74856844  112.09708462    2.87428422    4.31142633
    7.18571055   14.37142111   14.37142111   12.93427899    5.74856844
    5.74856844   27.3057001     4.31142633    2.87428422  120.71993728
   61.79711075   63.23425286    2.87428422    2.87428422    4.31142633
    7.18571055   35.92855276    2.87428422   22.99427377   48.86283176
    5.74856844   18.68284744   97.72566352    4.31142633    2.87428422
    2.87428422   35.92855276  110.65994251    7.18571055   21.55713166
    7.18571055    5.74856844    5.74856844   11.49713688   14.37142111
   10.05999477   11.49713688   66.10853708   38.80283698    2.87428422
   11.49713688   21.55713166   14.37142111    2.87428422   87.66566874
   22.99427377  142.27706894    2.87428422   18.68284744   38.80283698
   51.73711598    2.87428422    7.18571055    7.18571055   11.49713688]
--------------------------------------------------------------------------------
CPU-GPU difference:
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
ubuntu@ip-172-31-61-7:~/HW3$
```

Fibonacci Sequence

Because the first example cannot show the benefits of using GPU programming, this time I decided to implement a very famous problem in algorithm: the n-th Fibonacci sequence. Since its hard to call recursion in GPU, I'll use matrix approach to calculate it. The matrix approach's

formula is as follow:

$$\begin{bmatrix} F(n+1) \\ F(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

For the convenient of programming, I make all matrixes at 2x2 scale, and calculate the 30-th Fibonacci sequence in both CPU and GPU.

```
import numpy as np
import pycuda.autoinit
import pycuda.driver as cuda
from pycuda import compiler, gpuarray, tools

start = cuda.Event()
end = cuda.Event()
MATRIX_SIZE = 2
seq = 30
kernel_code_template = """
__global__ void MatrixMulKernel(float *a, float *b, float *c)
{
    // 2D Thread ID (assuming that only *one* block will be executed)
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Pvalue is used to store the element of the matrix
    // that is computed by the thread
    float Pvalue = 0;

    // Each thread loads one row of M and one column of N,
    //   to produce one element of P.
    for (int k = 0; k < %(MATRIX_SIZE)s; ++k) {
        float Aelement = a[ty * %(MATRIX_SIZE)s + k];
        float Belement = b[k * %(MATRIX_SIZE)s + tx];
        Pvalue += Aelement * Belement;
    }

    // Write the matrix to device memory;
    // each thread writes one element
    c[ty * %(MATRIX_SIZE)s + tx] = Pvalue;
}
"""
# Run on CPU
a_cpu = np.matrix('1 1; 1 0').astype(np.float32)
b_cpu = np.matrix('1 0; 1 0').astype(np.float32)
c_cpu = a_cpu

start.record()
```

```
for x in range(0, 29):
    c_cpu = c_cpu * a_cpu
c_cpu = c_cpu * b_cpu
end.record()
end.synchronize()

print "-" * 80
print "CPU time: %fs" %(start.time_till(end) * 1e-3)
print "Fibonacci(30) by CPU:",
print c_cpu.item(2)


# Run on GPU
a_gpu = gpuarray.to_gpu(a_cpu)
b_gpu = gpuarray.to_gpu(b_cpu)
c_gpu = gpuarray.to_gpu(a_cpu)

kernel_code = kernel_code_template % { 'MATRIX_SIZE': MATRIX_SIZE }
mod = compiler.SourceModule(kernel_code)
matrixmul = mod.get_function("MatrixMulKernel")

start.record()
for x in range(0, 29):
    matrixmul(c_gpu, a_gpu, c_gpu, block = (MATRIX_SIZE, MATRIX_SIZE, 1),)
matrixmul(c_gpu, b_gpu, c_gpu, block = (MATRIX_SIZE, MATRIX_SIZE, 1),)
end.record()
end.synchronize()

print "-" * 80
print "GPU time: %fs" %(start.time_till(end) * 1e-3)
print "Fibonacci(30) by GPU:",
print c_gpu.get().item(2)

print "-" * 80
print "CPU-GPU difference:"
print c_cpu.item(2) - c_gpu.get().item(2)

np.allclose(c_cpu, c_gpu.get())
```

Although the performance of GPU is still not better then CPU, the time difference between them is smaller.

```
ubuntu@ip-172-31-61-7:~/HW3$ python Example_2.py
--------------------------------------------------------------------------------
CPU time: 0.000251s
Fibonacci(30) by CPU: 1346269.0
--------------------------------------------------------------------------------
GPU time: 0.001506s
Fibonacci(30) by GPU: 1346269.0
--------------------------------------------------------------------------------
CPU-GPU difference:
0.0
ubuntu@ip-172-31-61-7:~/HW3$
```