LABORATORY N ° 3 PARALLEL PROGRAMMING

Katherine Camacho Calderón - Juan José Hoyos Urcué

In this laboratory, the parallelization of a sequential algorithm that helps to find the numerical derivative of a given function was developed. This algorithm is based on the numerical derivation method called backward derivative.

For parallelization, full use was made of the parallel computing platform provided by CUDA for Python whose execution library is explicitly Pycuda, therefore the purpose of this laboratory is to exploit the advantages of the GPU compared to normal and sequential execution done in the CPU, to demonstrate the great utility provided by the multiple cores that the Nvidia Mx110 GPU device offers.

The function that will be used to derive will be:

$$f(x) = 3x^2$$

Sequential algorithm

```
def secuencial(x,h,z,n):
    for i in range(n):
       z[i] = ((3*i*i)-(3*((i-h)*(i-h))))

    return z/h
```

Parallel algorithm

```
kernel = ElementwiseKernel(
        "int *x, int h,int *z",
        "z[i] = 3*(x[i]*x[i])-3*((x[i]-h)*(x[i]-h))",
        "ker")

def gpu(x,h,z):
    x_gpu = gpuarray.to_gpu(x)
    z_gpu = gpuarray.to_gpu(z)
    kernel(x_gpu,h,z_gpu)

    return z_gpu.get()/h
```

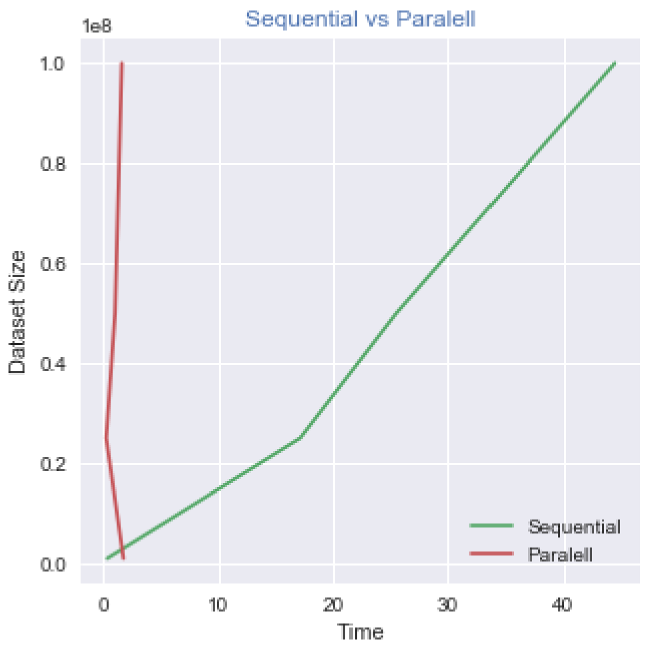
Both the sequential and parallel algorithms will have an array with n values that will be evaluated in the function, therefore each algorithm will find n derivatives.

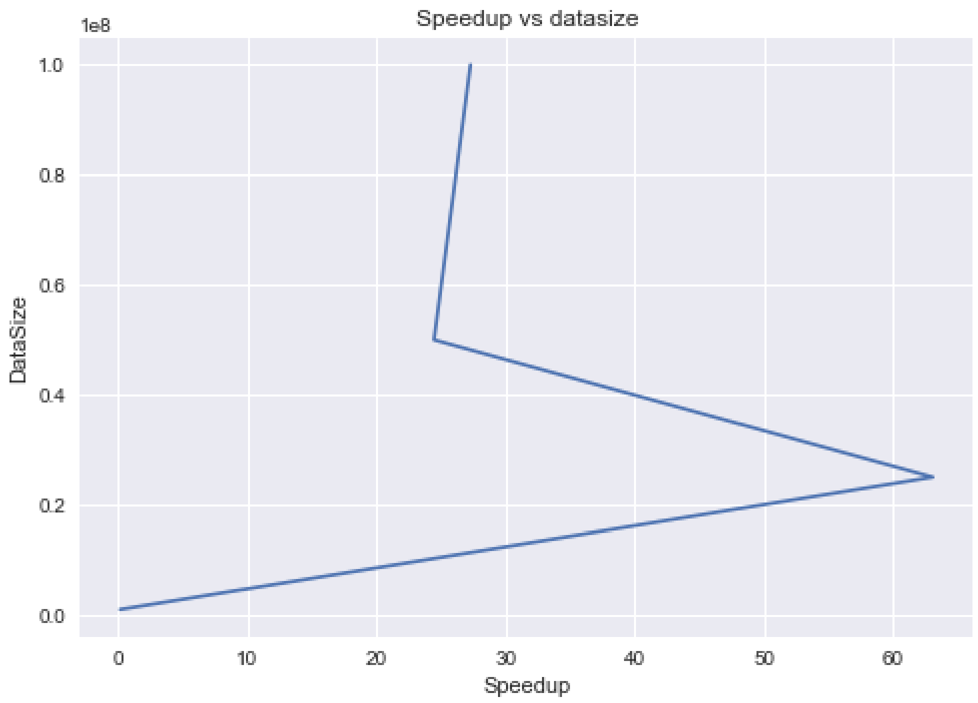
For a better analysis then four different size arrays were used, in the first iteration an array of size 1000000 was used, in the second iteration an array of size 25000000 was used, in the third iteration an array of size 50,000,000 was used and finally the last array used was 10 ^ 8 in size.

Algorithm comparing sequential vs parallel

```python
def graficar(x1,x2,y):
    sty = 'seaborn'
    mpl.style.use(sty)
    fig, ax = plt.subplots(figsize=(5, 5))

    ax.set_title("Sequential vs Paralell", color='C0')

    ax.plot(x1, y, 'C1', label='Sequential')
    ax.plot(x2, y, 'C2', label='Paralell')
    ax.set_ylabel(u'Dataset Size')
    ax.set_xlabel(u'Time')
    ax.legend()
    plt.show()

def main():
    datasize = [1000000,25000000,50000000,10**8]
    time_seq,time_par = list(),list()
    h = 1
    for n in datasize:
        x,z = np.arange(n),np.arange(n)

        #secuencial
        t1 = time()
        seq = secuencial(x,h,z,n)
        t2 = time()
        tf = t2-t1
        print tf
        time_seq.append(tf)

        #Paralelo
        t1_ = time()
        par = gpu(x,h,z)
        t2_ = time()
        tf = t2_ - t1_
        print tf
        time_par.append(tf)

        comp = seq==par
        if comp.all():
            print "Los resultados son iguales"
    graficar(time_seq,time_par,datasize)
    print "Secuencial : {}".format(time_seq)
    print "Paralelo : {}".format(time_par)

    speed_up = list()
    for i in range(len(datasize)):
        speed_up.append(time_seq[i]/time_par[i])
    plt.title(u"Speedup vs datasize")
    plt.xlabel(u"Speedup")
    plt.ylabel(u"DataSize")
    print "SpeedUp : {}".format(speed_up)
    plt.plot(speed_up,datasize)
main()
```

From this we obtain the following two graphs:





In graph 1 we can see the execution time it takes for each algorithm depending on the size of the data set, where the powerful improvement of the parallel algorithm with respect to the sequential one is evident

In the second graph we can observe the behavior of the algorithm's speedup depending on the size of the data set, which is extremely good, but at some point it deteriorates with respect to its highest peak, but it is still much better than the sequential algorithm.

As we runned the algorithm with 4 different dataset size, we calculate the time and speedup for eachone of them:

```
Secuencial : [0.3619999885559082, 17.161999940872192, 25.532000064849854,
44.50999999046326]
Paralelo : [1.7749998569488525, 0.2720000743865967, 1.0429999828338623,
1.630000114440918]
SpeedUp : [0.20394367196072366, 63.095570762527274, 24.47938685049509,
27.306746543223383]
```

Results

The previous results allowed us to conclude that when the amount of data in the array is small, the sequential algorithm is more efficient, while when the amount of data in the array increases, the parallel algorithm is highly efficient, all this thanks to the number of cores avaliable in the GPU, which allow these operations to be performed with very large speedups.