# Experiencias con Python y CUDA en Computación de Altas Prestaciones

Sergio Armas,Lionel Mena,Alejandro Samarín,Vicente Blanco[1],Alberto Morales y Francisco Almeida


Fig. 1. Diagrama de bloques de una GPU Tesla M2070 (Fermi)

*Resumen*--- **La computacion...**
*Palabras clave*--- **Python, CUDA, PyCUDA**

## I. Introducción

PyCUDA es un wrapper de Python para CUDA desarrollado por Andreas Klöckner. En este artículo se hará un estudio acerca de si las ventajas de utilizar un lenguaje interpretado de muy alto nivel para ejecutar algoritmos en una GPU compensa la menor velocidad inherente a los lenguajes de script. Para ello, se abordan dos problemas característicos: el producto matricial y la detección de movimiento.

...utilizando PyCUDA [1]

y una gráfica 1

y un código 1

## II. Producto matricial

La manera tradicional de abordar la multiplicación de matrices pasa por ejecutar un algoritmo secuencial de complejidad casi cúbica en las mejores implementaciones. Su versión paralela, en cambio, permite el cálculo simultáneo de las filas de la primera matriz multiplicando con la correspondiente columna de la segunda para formar el resultado.

PyCUBLAS o nuestros matrixmult?

## III. Producto matricial

## IV. Detección de movimiento

El algoritmo de detección de movimiento consiste, en su versión más simple, en la aplicación de tres filtros (por este orden: Difference, Thresold y Erosion) a cada par de fotogramas de la secuencia que se analiza. Como resulta fácil constatar, a poco que tratemos la detección de movimiento en un vídeo de pocos segundos de duración, la carga computacional será bastante elevada.

En el anexo de este artículo se proporciona el paquete Filters, que se estructura, esencialmente, en torno a dos clases: CUDAHandler, concebida para abstraer aún mas la comunicación con el dispositivo (informalmente, podríamos considerarla como un wrapper del propio PyCUDA), y Filter, clase abstracta cuyas clases hijas se corresponden con cada uno de los filtros implementados.

Además, la clase MotionDetector se encarga de realizar la detección en sí haciendo uso del paquete antes mencionado, y la clase VideoHandler abstrae la manipulación de las operaciones de gestión de vídeo.

## Referencias

[1] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan C. Catanzaro, Paul Ivanov, and Ahmed Fasih, ``Pycuda: Gpu run-time code generation for high-performance computing,'' *CoRR*, vol. abs/0911.3456, 2009.

[1]Dpto. Estadística, I.O. y Computación, Univ. La Laguna, e-mail: vblanco@ull.es

Listing 1
Codigo de filtros en Python

```
#!/opt/python2.7/bin/python

import sys
import Image
from abc import ABCMeta, abstractmethod

#                                       #

class Filter:
    """ Clase padre de filtros """
    __metaclass__ = ABCMeta

    # Constantes para indicar con que dispositivo se debe procesar el filtro
    CPU = 0
    CUDA = 1
    OPENCL = 2

    # Atributos
    images = []
    post_img = None

    def __init__(self, *images):
        for im in images:
            self.images.append(im)

    # TODO Esquema de colores como parametro
    def new_post_img(self, mode, size):
        self.post_img = Image.new(mode, size)

    def fetch_result(self):
        return self.post_img

    @abstractmethod
    def Apply(self):
        pass

#                                       #

class ErosionFilter(Filter):
    def __init__(self, *images):
        super(ErosionFilter, self).__init__(*images)

    def Apply(self):
        pass

#                                       #

class DifferenceFilter(Filter):
    def __init__(self, *images):
        super(DifferenceFilter, self).__init__(*images)

    def Apply(self, mode):
        self.new_post_img(self.images[0].mode,
                          (self.images[0].size[0], self.images[0].size[1]))
        for x in xrange(self.images[0].size[0]):
            for y in xrange(self.images[0].size[1]):
                # "diff" resultara ser una tupla de 3 elementos (en imagenes RGB) con la
                # diferencia en valor absoluto por cada canal en ese pixel, comparado con el
                # mismo pixel de la imagen anterior
                diff = tuple([abs(a   b) for a,b in zip(self.images[0].getpixel((x, y)), self.images[1].getpixel((x, y)))])
                # img.putpixel((x, y), value)
                self.post_img.putpixel((x, y), diff)

#                                       #

im1 = Image.open(sys.argv[1])
im2 = Image.open(sys.argv[2])
print sys.argv[1], ": ", im1.format, im1.size, im1.mode, '\n'
diferencia = DifferenceFilter(im1, im2)
diferencia.Apply(Filter.CPU)
post = diferencia.fetch_result()
post.save("post.png", "PNG")
```