

# Experiencias con Python y CUDA en Computación de Altas Prestaciones

Sergio Armas    Lionel Mena    Alejandro Samarín  
Vicente Blanco<sup>1</sup>    Alberto Morales    Francisco Almeida

<sup>1</sup>Departamento de Estadística, I.O. y Computación  
Universidad de La Laguna

XXII Jornadas de Paralelismo  
7, 8 y 9 de septiembre 2011  
La Laguna (Tenerife)

# Índice

- 1 Introducción
- 2 Computación con Python y CUDA
  - PyCUDA
  - PyCUBLAS
- 3 Caso práctico: Detección de movimiento
  - Algoritmo
  - Implementación
- 4 Conclusiones

# Introducción

- CUDA es una estructura de programación paralela desarrollada por NVIDIA.
  - Emplea una variación de C como *lingua franca*.
  - Diversos programadores, ajenos a NVIDIA, han creado *wrappers* en Java, Perl...
- Python es un lenguaje interpretado de muy alto nivel.
  - Su uso está en auge.
  - Librerías NumPy/SciPy.
- PyCUDA es un *wrapper* de Python para CUDA desarrollado por Andreas Klöckner.

# Índice

- 1 Introducción
- 2 Computación con Python y CUDA
  - PyCUDA
  - PyCUBLAS
- 3 Caso práctico: Detección de movimiento
  - Algoritmo
  - Implementación
- 4 Conclusiones

# Computación con Python y CUDA

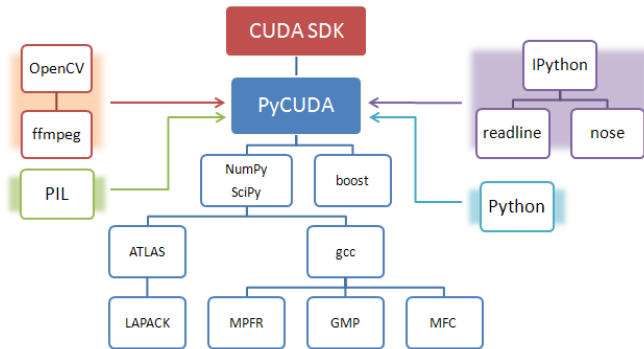


Figure: Un posible esquema de dependencias de PyCUDA

# Computación con Python y CUDA

## Flujo de ejecución de PyCUDA:

- Inclusión de las librerías necesarias.

---

```
import pycuda.autoinit
import pycuda.driver as cuda
from pycuda.compiler import SourceModule
```

---

- Carga de los datos en memoria.

---

```
import numpy
a = numpy.random.rand(4, 4).astype(numpy.float32)
```

---

- Reserva de espacio en el dispositivo.

---

```
a_gpu = cuda.mem_alloc(a.nbytes)
```

---

# Computación con Python y CUDA

- Transferencia de datos al dispositivo.

---

```
cuda.memcpy_htod(a_gpu, a)
```

---

- Ejecución del *kernel*.

---

```
func = mod.get_function("kernel_name")  
func(a_gpu, block=(4, 4, 1))
```

---

- Transferencia al *host*.

---

```
a_doubled = numpy.empty_like(a)  
cuda.memcpy_dtoh(a_doubled, a_gpu)
```

---

# Computación con Python y CUDA

- Alto nivel de abstracción.
- Hasta ahora, la mayoría de los dispositivos solo soportaban precisión simple.
- Python (lenguaje interpretado) + PyCUDA (*wrapper*) inducen un cierto *overhead* respecto a la ejecución nativa en C.



# Computación con Python y CUDA

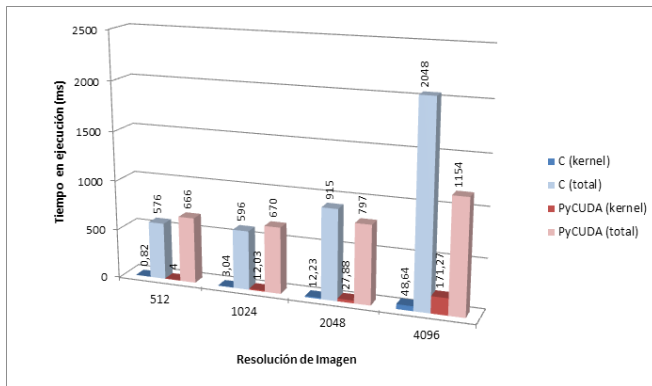


Figure: Comparativa entre C y Python sobre un filtro de convolución

# Índice

- 1 Introducción
- 2 Computación con Python y CUDA
  - PyCUDA
  - PyCUBLAS
- 3 Caso práctico: Detección de movimiento
  - Algoritmo
  - Implementación
- 4 Conclusiones

# Computación con Python y CUDA

- BLAS (Basic Linear Algebra Subprograms) es una interfaz estándar publicada en 1979.
- Numerosos fabricantes de hardware han creado versiones altamente optimizadas.
- CUBLAS es la implementación incluida por NVIDIA en el SDK de CUDA.
- PyCUBLAS es un *wrapper* de Python para CUBLAS desarrollado por Derek Anderson.

# Computación con Python y CUDA

## Ejemplo de multiplicación de matrices con PyCUBLAS:

---

```
import numpy as np
from pycublas import CUBLASMatrix

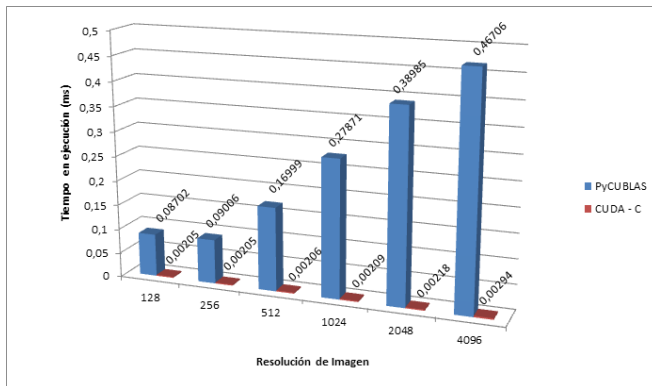
a = np.random.randn(width,length).astype(np.float32)
b = np.random.randn(width,length).astype(np.float32)

a = CUBLASMatrix(np.mat(a))
b = CUBLASMatrix(np.mat(b))

c = a * b
```

---

# Computación con Python y CUDA



**Figure:** Comparativa entre CUDA C y PyCUBLAS sobre la multiplicación de matrices cuadradas

# Índice

- 1 Introducción
- 2 Computación con Python y CUDA
  - PyCUDA
  - PyCUBLAS
- 3 Caso práctico: Detección de movimiento
  - Algoritmo
  - Implementación
- 4 Conclusiones

# Caso práctico: Detección de movimiento

- 1 Conversión a escala de grises de las 2 imágenes.
- 2 Aplicación del filtro de diferencia a las 2 imágenes.
- 3 Aplicación del filtro Threshold.
- 4 Aplicación del filtro de Erosión.
- 5 Mezcla en el canal R de la imagen original con la imagen resultado de los filtros.

# Caso práctico: Detección de movimiento



Imagen 1 – Frame 1



Imagen 2 – Frame 2

**Figure:** Extracto de dos frames consecutivos del vídeo de trabajo



# Caso práctico: Detección de movimiento



Imagen 3 - Diferencia

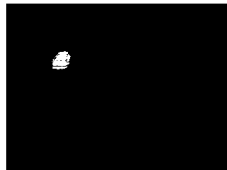


Imagen 4 - Threshold

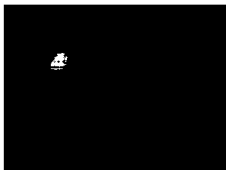


Imagen 5 - Erosión



Imagen 6 - Resultado

Figure: Aplicación del algoritmo paso a paso

# Índice

- 1 Introducción
- 2 Computación con Python y CUDA
  - PyCUDA
  - PyCUBLAS
- 3 Caso práctico: Detección de movimiento
  - Algoritmo
  - **Implementación**
- 4 Conclusiones

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ | ≡ ≡ ↺ 🔍 ↻

# Caso práctico: Detección de movimiento

CUDAHandler: Abstracción de la comunicación con la GPU.

VideoHandler: Abstracción de la manipulación de vídeo.

Filter: Interfaz común de la que heredan los filtros.

MotionDetector: Dirección del proceso de detección.

## Ejemplo: Aplicación del filtro de diferencia

---

```
cuda.loadData(input=self.images, output=self.post_img)  
from Kernels.difference_kernel import diffkernel  
cuda.setKernel(diffkernel)  
cuda.Launch((32, 32, 1))
```

---

# Conclusiones

- Implementación sencilla para problemas complejos.
- Menor curva de aprendizaje.
- El *overhead* inherente al uso de *wrappers* y lenguajes interpretados disminuye conforme aumenta el tamaño del problema.
- Ideal para investigadores con necesidad de cálculo masivo paralelo.

# Bibliografía destacada I



Andreas Klöckner *et ál.*

PyCUDA: GPU run-time code generation for  
high-performance computing  
[arXiv:0911.3456v2](#)



Derek Anderson

PyCUBLAS - Easy Python/NumPy/CUDA/CUBLAS  
integration  
<http://kered.org>



Zhiyi Yang, Yating Zhu, Yong Pu

Parallel image processing based on CUDA  
CSSE (3). 2008 pp. 198-201, IEEE Computer Society

# Preguntas



# Gracias por su atención