

# 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
  - Resultados
- 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
  - Resultados
- 4 Conclusiones

# Computación con Python y CUDA

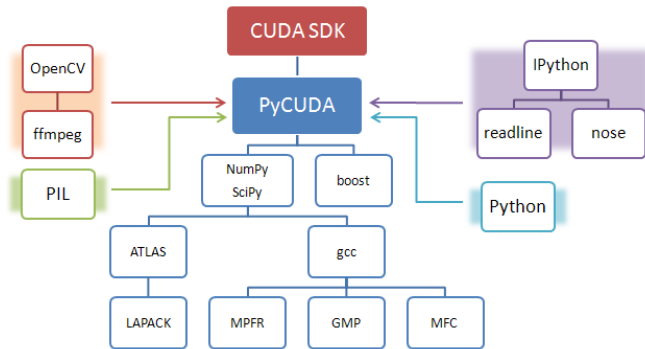


Figure: Un posible esquema de dependencias de PyCUDA

# 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

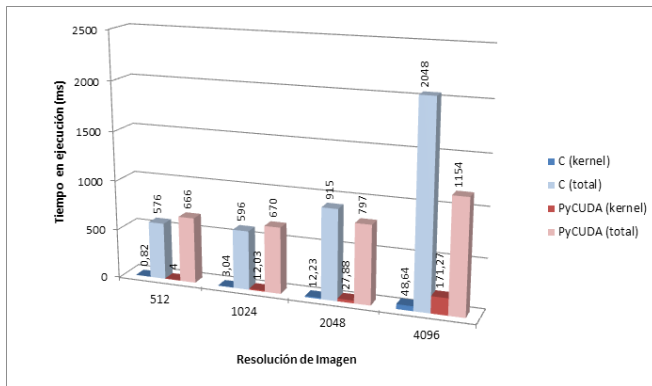


Figure: Comparativa entre C y Python sobre un filtro de convolución

# Índice

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

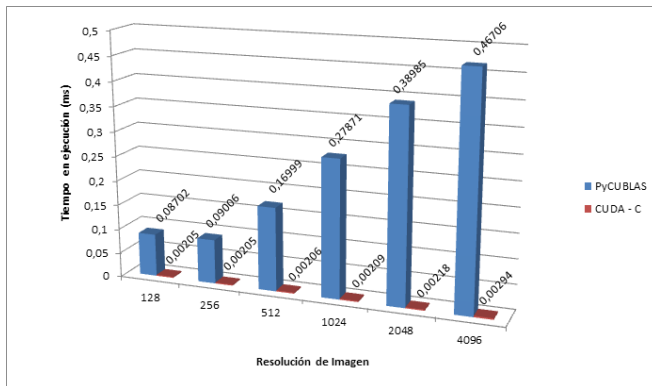
a = np.random.randn(width,length).astype(np.float32)
b = np.random.randn(width,length).astype(np.float32)

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
  - Resultados
- 4 Conclusiones

# Make Titles Informative.

# Make Titles Informative.



# Make Titles Informative.

# Índice

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

# Make Titles Informative.

# Make Titles Informative.

# Make Titles Informative.

# 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.

# For Further Reading I



A. Author.

*Handbook of Everything.*

Some Press, 1990.



Andreas Klöckner *et ál.*

PyCUDA: GPU Run-Time Code Generation for  
High-Performance Computing

[arXiv:0911.3456v2](#)