INTRODUCCIÓN A NUMBA

ARQUITECTURAS E INFRAESTRUCTURAS PARA INTELIGENCIA ARTIFICIAL
MÁSTER EN INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE ALICANTE

David Mulero-Pérez <dmulero@dtic.ua.es>
Manuel Benavent-Lledó <mbenavent@dtic.ua.es>
José García-Rodríguez <jgarcia@dtic.ua.es>

CONTENIDO

¿QUÉ ES NUMBA?

COMPILADOR PARA PYTHON

NUMBA ES UN COMPILADOR **JIT** (JUST-IN-TIME) PARA PYTHON

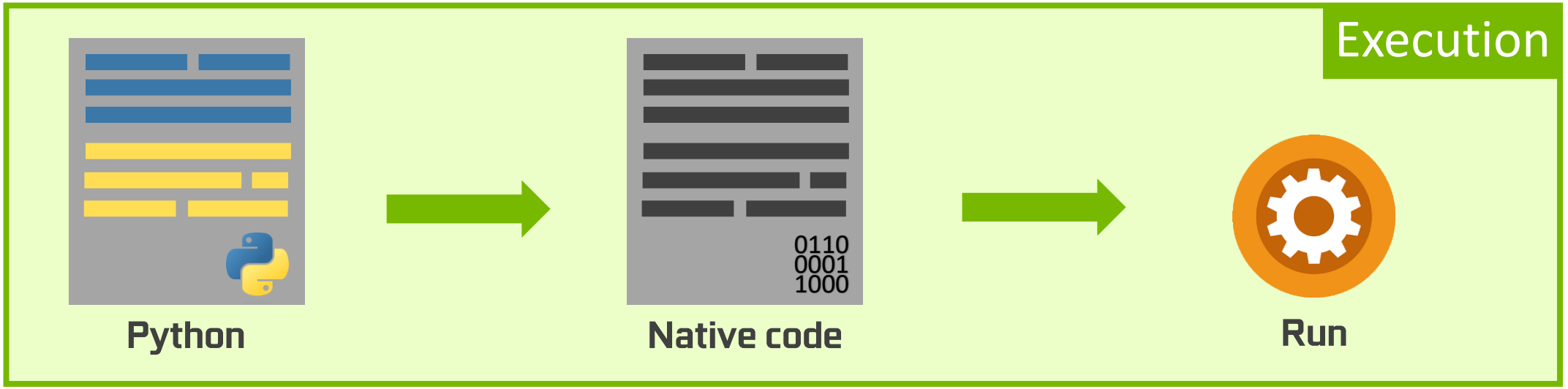
PERMITE TRANSFORMAR FUNCIONES DE PYTHON PURO EN CÓDIGO COMPILADO PARA EJECUCIÓN EN LA **CPU** O **GPU**.

FUNCIONA BIEN CON FUNCIONES Y ARRAYS DE **NUMPY**

¿QUÉ ES NUMBA?

¿Y QUÉ ES JUST-IN-TIME?

JUST-IN-TIME (JIT) ES UNA TÉCNICA DE COMPILACIÓN QUE PERMITE **COMPILAR** Y EJECUTAR EL CÓDIGO **EN TIEMPO DE EJECUCIÓN**, EN LUGAR DE COMPILARLO ANTES DE SU EJECUCIÓN. ESTO PERMITE QUE EL COMPILADOR **OPTIMICE EL CÓDIGO** SEGÚN EL CONTEXTO DE LA EJECUCIÓN Y PUEDE OFRECER UNA MEJORA SIGNIFICATIVA EN LA VELOCIDAD DE LA EJECUCIÓN.

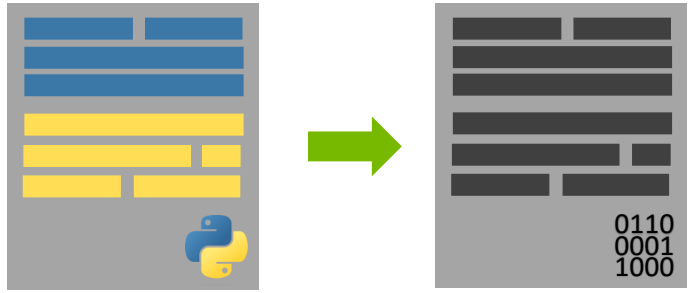


FUNCIONES JIT

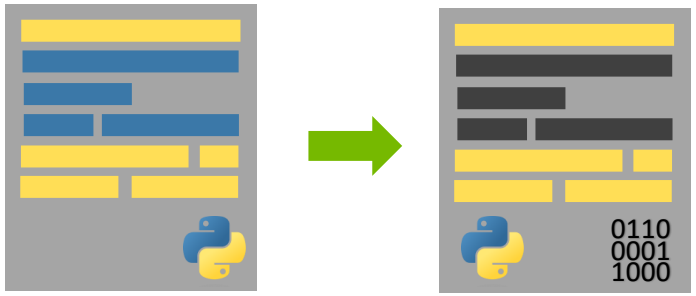
MODO NOPYTHON Y OBJECT

EL DECORADOR **@JIT** SE AÑADE A EN LAS FUNCIONES PYTHON A COMPILAR Y TIENE DOS MODOS FUNDAMENTALES:

NOPYTHON: PERMITE COMPILAR LA FUNCIÓN PARA QUE SE EJECUTE COMPLETAMENTE EN CÓDIGO MÁQUINA, LO CUAL PROPORCIONA EL MEJOR RENDIMIENTO



OBJECT: NUMBA IDENTIFICARÁ LOS BUCLES QUE PUEDE COMPILAR Y LOS CONVERTIRÁ EN FUNCIONES QUE SE EJECUTAN EN CÓDIGO DE MÁQUINA, MIENTRAS QUE EL RESTO DEL CÓDIGO SE EJECUTARÁ EN EL INTÉRPRETE DE PYTHON



FUNCIONES JIT

DEFINICIÓN DE FUNCIÓN JIT

OBJECT: **@jit**

```
from numba import jit

@jit
def f(x, y):
    return x + y

a = np.arange(200, dtype=np.int64)
b = np.arange(200, dtype=np.int64)
f(a,b)
```

NOPYTHON: **@jit(nopython=True)**
@njit

```
from numba import jit

@jit(nopython=True)
def f(x, y):
    return math.sqrt(square(x) + square(y))

a = np.arange(200, dtype=np.int64)
b = np.arange(200, dtype=np.int64)
f(a,b)
```

FUNCIONES JIT

DEFINICIÓN DE FUNCIÓN JIT

OBJECT: **@jit**

```
from numba import jit
import pandas as pd

@jit
def f(x, y):
  df = pd.DataFrame({'x': x, 'y': y})
  df['sum'] = df['x'] + df['y’]
  return df

a = np.arange(200, dtype=np.int64)
b = np.arange(200, dtype=np.int64)
f(a,b)
```

NOPYTHON: **@jit(nopython=True)** **@njit**

```
from numba import jit
import pandas as pd

@jit(nopython=True)
def f(x, y):
   df = pd.DataFrame({'x': x, 'y': y})
   df['sum’]=math.sqrt(square(x)+square(y))
   return df

a = np.arange(200, dtype=np.int64)
b = np.arange(200, dtype=np.int64)
f(a,b)
```

FUNCIONES JIT

FUNCIONES COMPATIBLES CON JIT

ESTRUCTURAS PYTHON:

IF ..

ELIF ..

ELSE

WHILE

FOR .. IN

BREAK

CONTINUE

YIELD

ASSERT

FUNCIONES PYTHON:

ABS()

MIN()

MAX()

LENGTH()

MAP()

PRINT()

SHORT()

MATH.ATAN()

MATH.EXP()

FUNCIONES NUMPY:

RESHAPE()

SORT()

SUM()

NUMPY.LINALG.SOLVE()

NUMPY.LINALG.DET()

NUMPY.LINALG.INV()

NUMPY.PERCENTILE()

NUMPY.ARGWHERE()

NUMPY.COV()

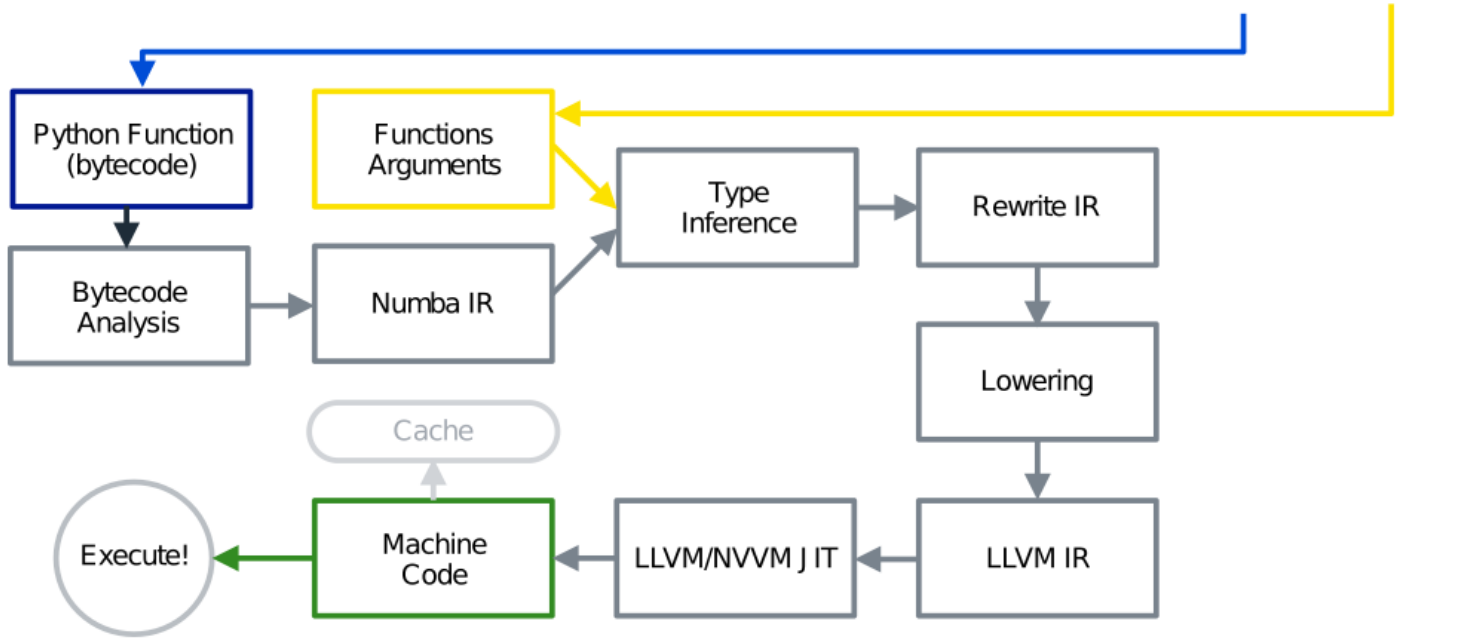
INTEGRACIÓN CON CUDA

EJECUCIÓN DE KERNEL EN GPU

NUMBA PERMITE **COMPILAR** CÓDIGO PARA LA **GPU** USANDO **CUDA**. ESTÁ RESTRINGIDO A UN SUBCONJUNTO DE INSTRUCCIONES DE CÓDIGO PYTHON.

```
@cuda.jit
def axpy(r, a, x, y)
   ...
>>>  axpy (r, a, x,y)
```

LOS **KERNELS** ESCRITOS EN NUMBA GESTIONAN DE FORMA TRANSPARENTE AL PROGRAMADOR **ARRAYS** DE **NUMPY**.



INTEGRACIÓN CON CUDA

TERMINOLOGÍA

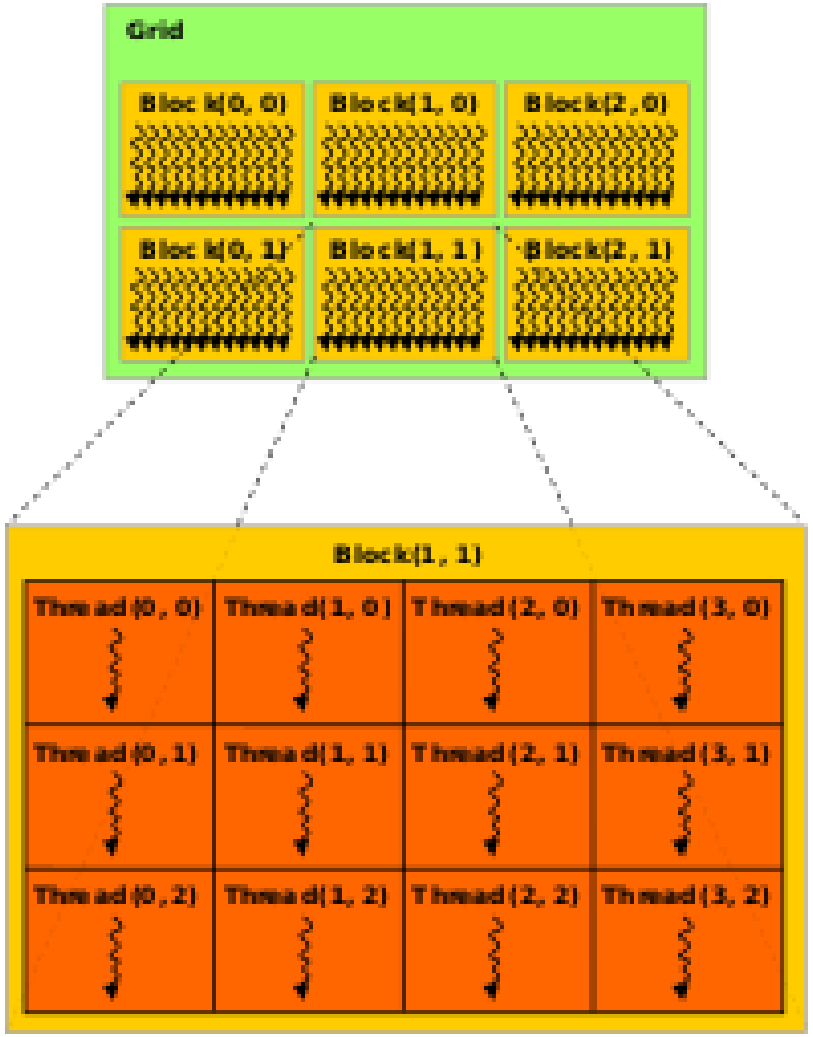
TERMINOLOGÍA EMPLEADA EN CUDA:

HOST: CPU (PROCESADOR)

DEVICE: GPU (TARJETA GRÁFICA)

KERNEL: FUNCIÓN EJECUTADA EN GPU

THREAD: UNIDAD MÍNIMA DE EJECUCIÓN DE CÓDIGO EN CUDA

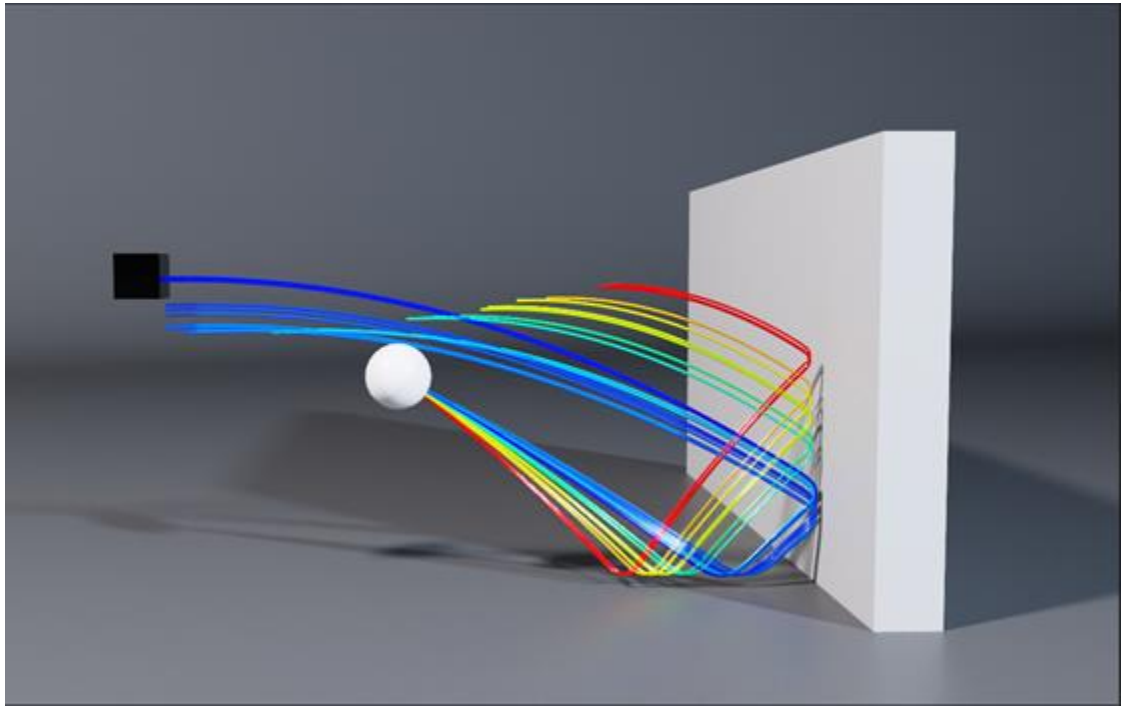


CASOS DE USO

EJEMPLOS

ES POSIBLE UTILIZAR KERNELS DE CUDA PARA ACELERAR CÁLCULOS EN DIFERENTES ÁMBITOS:

SIMULACIONES FÍSICAS Y NUMÉRICAS:
SIMULACIONES DE SISTEMAS FÍSICOS COMPLEJOS, COMO LA DINÁMICA DE PARTÍCULAS, LA MECÁNICA DE FLUIDOS.



CASOS DE USO

EJEMPLOS

ES POSIBLE UTILIZAR KERNELS DE CUDA PARA ACELERAR CÁLCULOS EN DIFERENTES ÁMBITOS:

SIMULACIONES FÍSICAS Y NUMÉRICAS

PROCESAMIENTO DE IMÁGENES:
EN ALGORITMOS DE DETECCIÓN DE BORDES, SEGMENTACIÓN, FILTROS, ETC.

CASOS DE USO

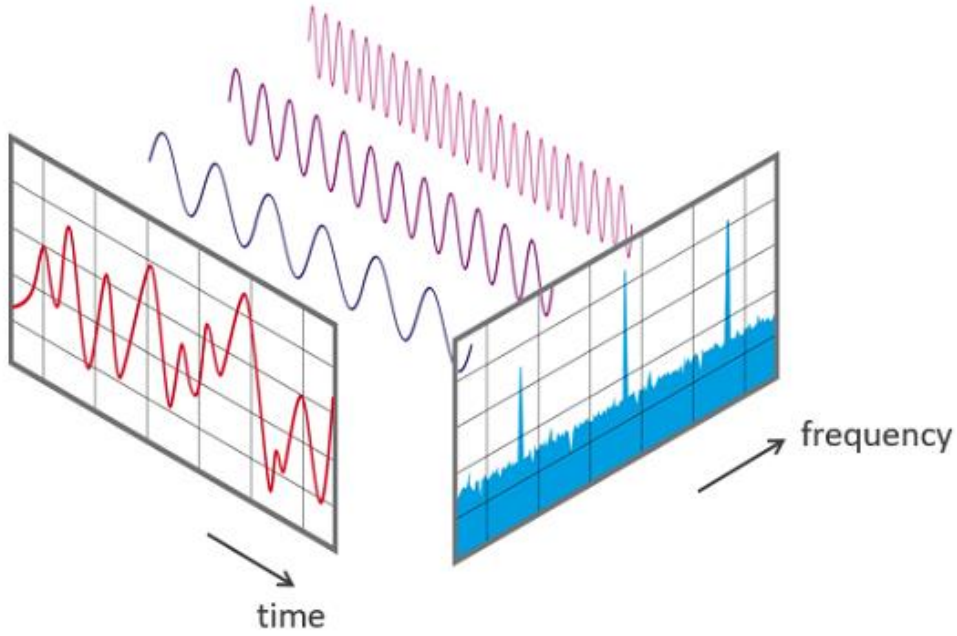
EJEMPLOS

ES POSIBLE UTILIZAR KERNELS DE CUDA PARA ACELERAR CÁLCULOS EN DIFERENTES ÁMBITOS:

SIMULACIONES FÍSICAS Y NUMÉRICAS

PROCESAMIENTO DE IMÁGENES

ACELERACIÓN DE ALGORÍTMOS PARALELIZABLES:
SIMULACIÓN DE MONTECARLO, TRANSFORMADA RÁPIDA DE FOURIER



CONSEJOS Y BUENAS PRÁCTICAS

a

EVITAR TRANSFERENCIAS DE DATOS INNECESARIAS: LAS TRANSFERENCIAS ENTRE LA CPU Y LA GPU SON COSTOSAS EN TÉRMINOS DE TIEMPO

UTILIZAR TIPOS DE DATOS ADECUADOS: UTILIZA TIPOS DE DATOS NUMÉRICOS QUE SEAN COMPATIBLES CON LA GPU

REALIZAR PRUEBAS INCREMENTALES: REALIZAR PRUEBAS INCREMENTALES AUMENTANDO LA COMPLEJIDAD. ESTO TE PERMITE IDENTIFICAR ERRORES Y OPTIMIZAR EL RENDIMIENTO DE MANERA MÁS EFECTIVA.

¿ALGUNA PREGUNTA?

ARQUITECTURAS E INFRAESTRUCTURAS PARA INTELIGENCIA ARTIFICIAL
MÁSTER EN INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE ALICANTE

David Mulero-Pérez <dmulero@dtic.ua.es>
Manuel Benavent-Lledó <mbenavent@dtic.ua.es>
José García-Rodríguez <jgarcia@dtic.ua.es>

PRÁCTICA CUDA

CLICK AQUÍ