INTRODUCTION TO NUMBA

JGPUQC - 2024

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

CONTENIDO

WHAT IS NUMBA?

COMPILER FOR PYTHON

NUMBA IS A **JUST-IN-TIME (JIT)** COMPILER FOR PYTHON

IT ALLOWS TRANSFORMING PURE PYTHON FUNCTIONS INTO COMPILED CODE FOR EXECUTION ON THE **CPU** OR **GPU**.

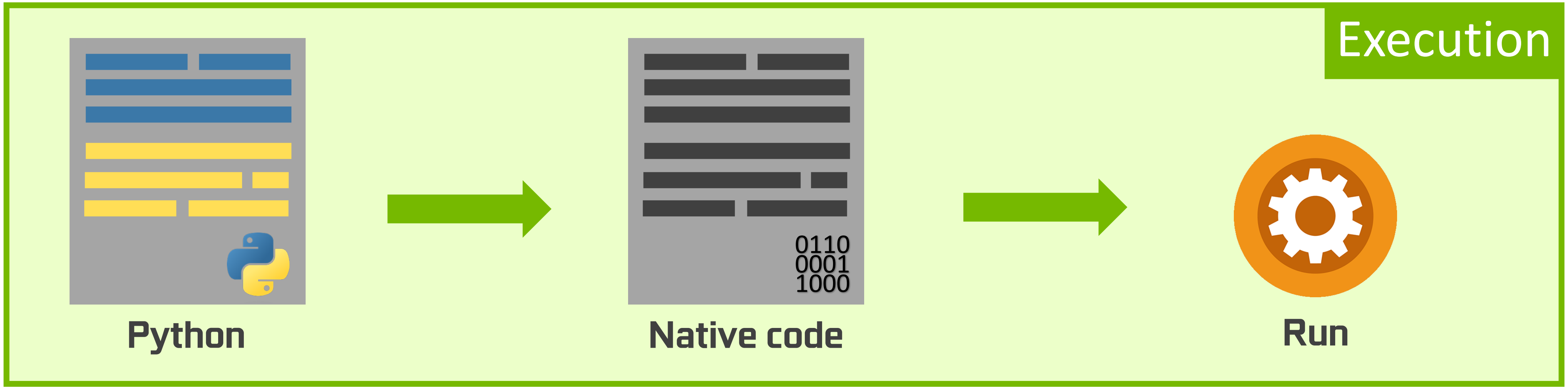
WORKS WELL WITH **NUMPY** FUNCTIONS AND ARRAYS

WHAT IS NUMBA?

AND WHAT IS JUST-IN-TIME?

JUST-IN-TIME (JIT) IS A COMPILATION TECHNIQUE THAT ALLOWS CODE TO BE **COMPILED** AND **EXECUTED AT RUNTIME**, RATHER THAN BEING COMPILED PRIOR TO EXECUTION.

THIS ALLOWS THE COMPILER TO **OPTIMIZE THE CODE** ACCORDING TO THE EXECUTION CONTEXT AND CAN OFFER A SIGNIFICANT IMPROVEMENT IN EXECUTION SPEED.

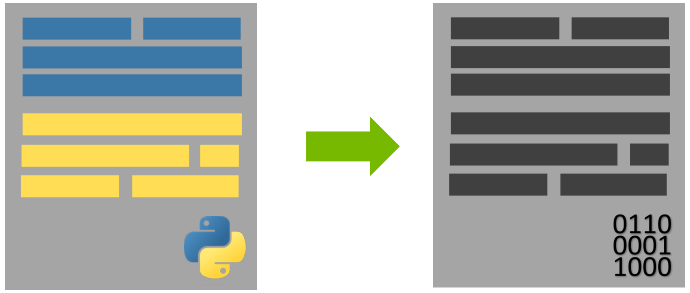


JIT FUNCTIONS

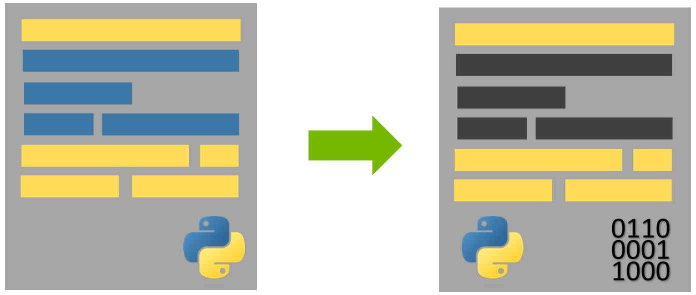
NOPYTHON AND OBJECT MODE

THE **@JIT** DECORATOR IS ADDED TO THE PYTHON FUNCTIONS TO BE COMPILED AND IT HAS TWO FUNDAMENTAL MODES:

NOPYTHON: ALLOWS COMPILING THE FUNCTION TO RUN ENTIRELY IN MACHINE CODE, WHICH PROVIDES THE BEST PERFORMANCE.



OBJECT: NUMBA WILL IDENTIFY THE LOOPS IT CAN COMPILE AND WILL CONVERT THEM INTO FUNCTIONS THAT RUN IN MACHINE CODE, WHILE THE REST OF THE CODE WILL BE EXECUTED IN THE PYTHON INTERPRETER.



JIT FUNCTIONS

JIT FUNCTION DEFINITION

OBJECT: **@jit**

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

```python
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)
```

```python
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)
```

JIT FUNCTIONS

JIT FUNCTION DEFINITION

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)
```

JIT FUNCTIONS

FUNCIONES COMPATIBLES CON JIT

PYTHON STRUCTURES:

IF ..

ELIF ..

ELSE

WHILE

FOR .. IN

BREAK

CONTINUE

YIELD

ASSERT

PYTHON FUNCTIONS:

ABS()

MIN()

MAX()

LENGTH()

MAP()

PRINT()

SHORT()

MATH.ATAN()

MATH.EXP()

NUMPY FUNCTIONS:

RESHAPE()

SORT()

SUM()

NUMPY.LINALG.SOLVE()

NUMPY.LINALG.DET()

NUMPY.LINALG.INV()

NUMPY.PERCENTILE()

NUMPY.ARGWHERE()

NUMPY.COV()

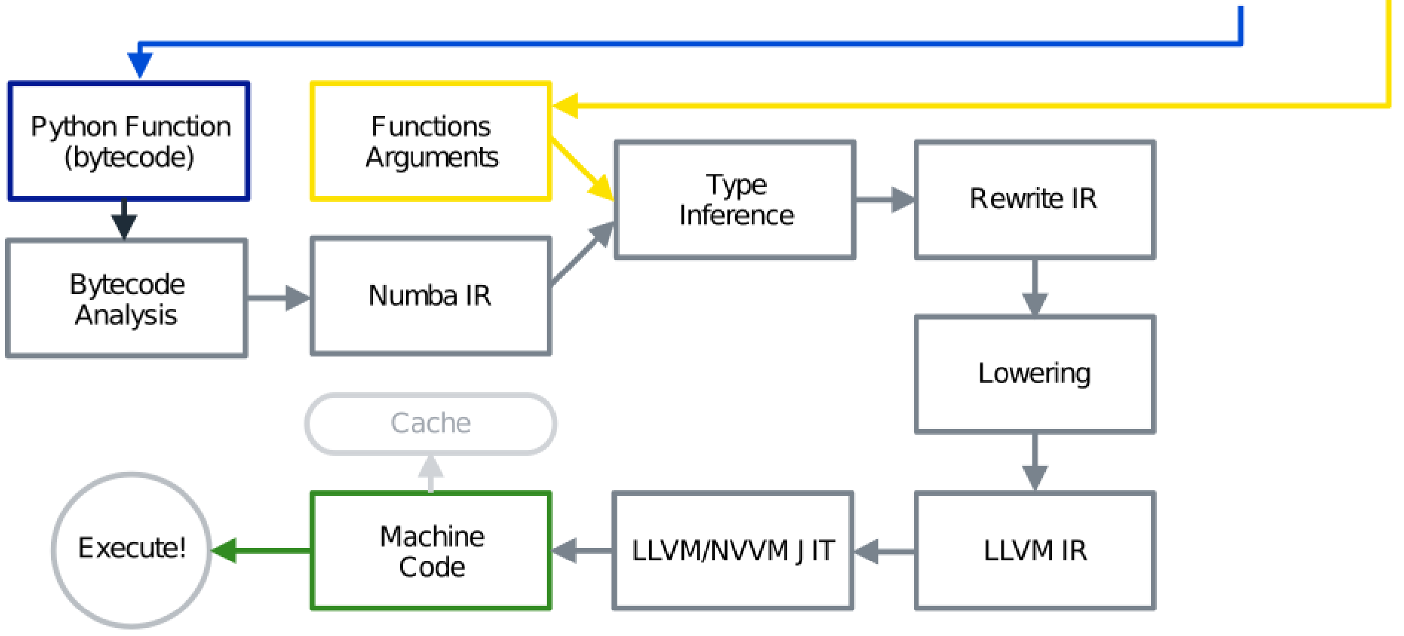
INTEGRATION WITH CUDA

GPU KERNEL EXECUTION

NUMBA ALLOWS **COMPILING** CODE FOR THE **GPU** USING **CUDA**. IT IS RESTRICTED TO A SUBSET OF PYTHON CODE INSTRUCTIONS.

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

KERNELS WRITTEN IN NUMBA TRANSPARENTLY **MANAGE NUMPY ARRAYS** FOR THE PROGRAMMER.



INTEGRATION WITH CUDA

TERMINOLOGY

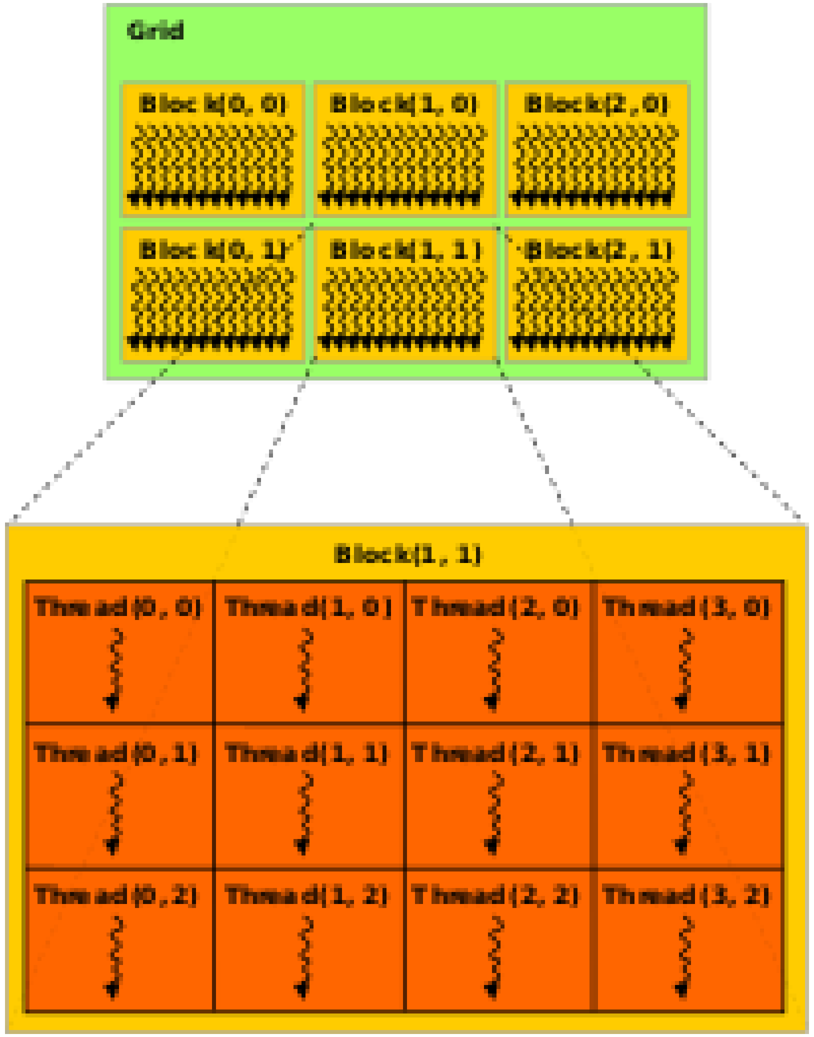
TERMINOLOGY USED IN CUDA:

HOST: CPU (PROCESSOR)

DEVICE: GPU (GRAPHICS CARD)

KERNEL: FUNCTION EXECUTED ON GPU

THREAD: MINIMUM UNIT OF CODE EXECUTION IN CUDA

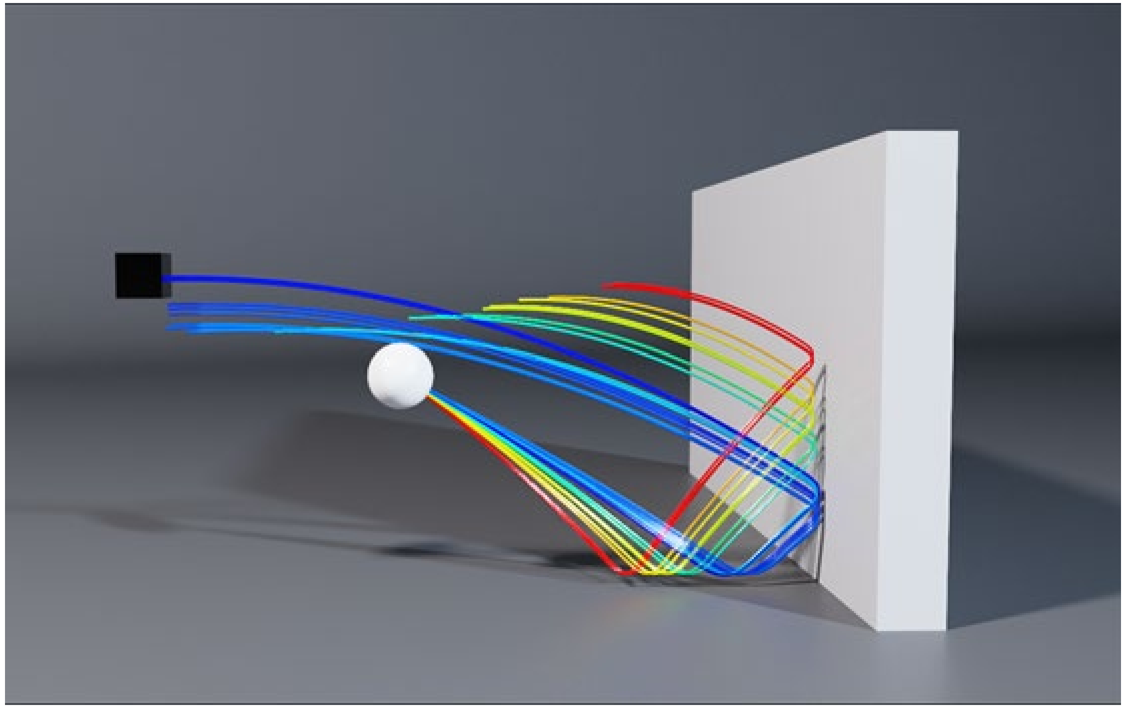


USE CASES

EXAMPLES

IT IS POSSIBLE TO USE CUDA KERNELS TO ACCELERATE CALCULATIONS IN DIFFERENT AREAS:

PHYSICAL AND NUMERICAL SIMULATIONS:
SIMULATIONS OF COMPLEX PHYSICAL SYSTEMS, SUCH AS PARTICLE DYNAMICS, FLUID MECHANICS.



USE CASES

EXAMPLES

IT IS POSSIBLE TO USE CUDA KERNELS TO ACCELERATE CALCULATIONS IN DIFFERENT AREAS:

PHYSICAL AND NUMERICAL SIMULATIONS

IMAGE PROCESSING:
IN ALGORITHMS FOR EDGE DETECTION, SEGMENTATION, FILTERS, ETC.

USE CASES

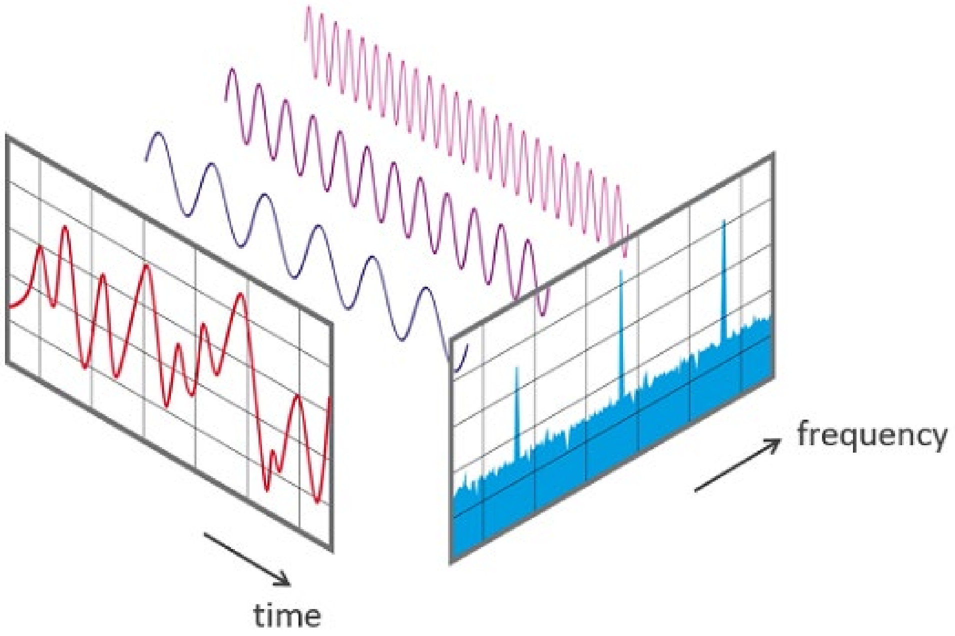
EXAMPLES

IT IS POSSIBLE TO USE CUDA KERNELS TO ACCELERATE CALCULATIONS IN DIFFERENT AREAS:

PHYSICAL AND NUMERICAL SIMULATIONS

IMAGE PROCESSING

ACCELERATION OF PARALLELIZABLE ALGORITHMS:
MONTE CARLO SIMULATION,
FAST FOURIER TRANSFORM



TIPS AND BEST PRACTICES

a

AVOID UNNECESSARY DATA TRANSFERS: TRANSFERS BETWEEN THE CPU AND GPU ARE COSTLY IN TERMS OF TIME

USE APPROPRIATE DATA TYPES: USE NUMERIC DATA TYPES THAT ARE COMPATIBLE WITH THE GPU

PERFORM INCREMENTAL TESTING: CARRY OUT INCREMENTAL TESTS INCREASING COMPLEXITY. THIS ALLOWS YOU TO IDENTIFY ERRORS AND OPTIMIZE PERFORMANCE MORE EFFECTIVELY.

¿ANY QUESTION?

JGPUQC - 2024

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

PRÁCTICA CUDA

CLICK HERE