ICNPG 2023

Python y Julia

GPUs y python

- Numba
- Cupy
- RAPIDS
- PyCUDA
- Etc...

- **Interpreted languages:**
  - **Python, Ruby, or JavaScript, etc.**
  - **Execute code directly from the source code without the need for a compilation step.**
  - **often have a simpler syntax and are easier to learn, read, and write.**
  - **better support for dynamic typing and dynamic memory management.**
  - **slower than compiled languages because they must interpret the code at runtime.**
- Compiled languages:
  - C, C++, Fortran, Julia, etc.
  - require a compilation step to translate the source code into machine code.
  - tend to have a more complex syntax and require more experience to learn, read, and write.
  - They often have better support for static typing and memory management.

Numba

https://numba.pydata.org/

Accelerate Python Functions

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

You don't need to replace the Python interpreter, run a separate compilation step, or even have a C/C++ compiler installed. Just apply one of the Numba decorators to your Python function, and Numba does the rest.

```
from numba import njit
import random

@njit
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

https://numba.pydata.org/

Portable Compilation

Ship high performance Python applications without the headache of binary compilation and packaging. Your source code remains pure Python while Numba handles the compilation at runtime. We test Numba continuously in more than 200 different platform configurations.

Numba supports Intel and AMD x86, POWER8/9, and ARM CPUs (including Apple M1), NVIDIA GPUs, Python 3.7-3.10, as well as Windows/macOS/Linux. Precompiled Numba binaries for most systems are available as conda packages and pip-installable wheels.

https://numba.readthedocs.io/en/stable/cuda/index.html

Numba

https://numba.pydata.org/

https://numba.readthedocs.io/en/stable/cuda/index.html

GPU Acceleration

NVIDIA CUDA

With support for NVIDIA CUDA, Numba lets you write parallel GPU algorithms entirely from Python.

```python
import numpy as np
from numba import cuda
```

```python
@cuda.jit
def f(a, b, c):
    # like threadIdx.x + (blockIdx.x * blockDim.x)
    tid = cuda.grid(1)
    size = len(c)

    if tid < size:
        c[tid] = a[tid] + b[tid]
```

```python
N = 100000
a = cuda.to_device(np.random.random(N))
b = cuda.to_device(np.random.random(N))
c = cuda.device_array_like(a)
```

NumPy/SciPy-compatible Array Library for GPU-accelerated Computing with Python

https://cupy.dev/

HIGHLY COMPATIBLE WITH NUMPY & SCIPY

```
>>> import cupy as cp
>>> x = cp.arange(6).reshape(2, 3).astype('f')
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]], dtype=float32)
>>> x.sum(axis=1)
array([  3.,  12.], dtype=float32)
```

```
>>> x = cp.arange(6, dtype='f').reshape(2, 3)
>>> y = cp.arange(3, dtype='f')
>>> kernel = cp.ElementwiseKernel(
...     'float32 x, float32 y', 'float32 z',
...     '''
...     if (x - 2 > y) {
...       z = x * y;
...     } else {
...       z = x + y;
...     }
...     ''', 'my_kernel')
>>> kernel(x, y)
array([[ 0.,  2.,  4.],
       [ 0.,  4.,  10.]], dtype=float32)
```

The N-dimensional array (`ndarray`)

Universal functions (`cupy.ufunc`)

Routines (NumPy)

Routines (SciPy)

CuPy-specific functions

Low-level CUDA support

Custom kernels

Distributed

Environment variables

Comparison Table

Python Array API Support

RAPIDS
GPU Accelerated Data Science

FASTER PANDAS WITH CUDF

cuDF is a near drop in replacement to pandas for most use cases and has greatly improved performance.

FASTER SCIKIT-LEARN WITH CUML

cuML brings huge speedups to ML modeling with an API that matches scikit-learn.

FASTER NETWORKX WITH CUGRAPH

cuGraph makes migration from networkX easy, accelerates graph analytics, and allows scaling far beyond existing tools.

- Dataframe processing with **cuDF** (similar API to pandas)
- Machine learning with **cuML** (similar API to scikit-learn)
- Graph processing with **cuGraph** (similar API to networkX)
- Spatial analytics with **cuSpatial** (similar API to geoPandas)
- Image processing with **cuCIM** (similar API to scikit-image)
- Signal processing **cuSignal** (similar API to scipy.signal)
- Seamless cross-filtered dashboards with **cuxfilter**
- Low level compute primitives with **RAFT**
- Apache Spark acceleration with **Spark RAPIDS**

```python
import cudf, requests
from io import StringIO

url = "https://github.com/plotly/datasets/raw/master/tips.csv"
content = requests.get(url).content.decode('utf-8')

tips_df = cudf.read_csv(StringIO(content))
tips_df['tip_percentage'] = tips_df['tip'] / tips_df['total_bill'] * 100

# display average tip by dining party size
print(tips_df.groupby('size').tip_percentage.mean())
```

Ejercicio

```python
#@title numpy
import numpy as np
import time

n = 10000000

# create x and y vectors
x = np.arange(1, n+1, dtype=np.float32)
y = np.arange(n, 0, -1, dtype=np.float32)

start_time = time.time()

# create z vector
z = x + y

# normalize z by its Euclidean norm
norm = np.linalg.norm(z)
#z_norm = z / norm

# sum the elements of the final result
result = np.sum(z)/norm

# print the result and the time taken
print("Result:", result)
print("Time taken:", (time.time() - start_time) * 1000000, "us")

tnumpy = (time.time() - start_time) * 1000000
#print(x,y,z,z_norm)
```

$$S = \frac{\sum_{i=0}^{n}(x_i + y_i)}{\sqrt{\sum_{i=0}^{n}(x_i + y_i)^2}}$$

Ejercicio: hacerlo en CUPY y NUMBA

Tasa de contagio

$$\frac{dS}{dt} = -\beta SI,$$

$$\frac{dI}{dt} = \beta SI - \gamma I,$$

$$\frac{dR}{dt} = \gamma I.$$

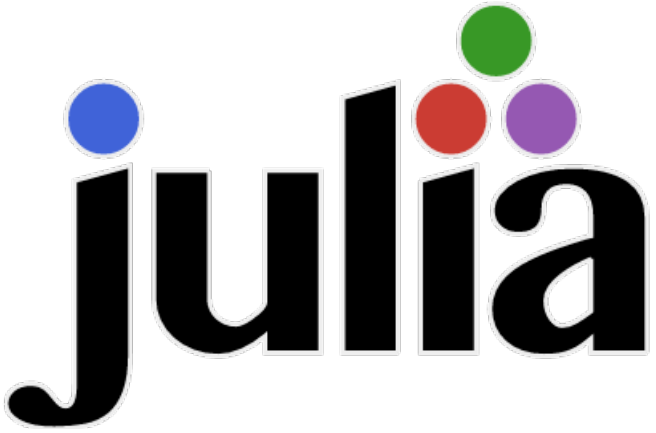
Método de Euler

$$S(t + \Delta t) = S(t) + \Delta t[-\beta I(t)S(t)]$$

$$I(t + \Delta t) = I(t) + \Delta t[\beta I(t)S(t) - \gamma I(t)]$$

$$R(t + \Delta t) = R(t) + \Delta t[\gamma I(t)]$$

Suceptible, **I**nfectado, **R**ecuperado



https://julialang.org/

Julia is a good choice for numerical computing and scientific computing tasks that require **high performance and parallelism**, while Python is a good choice for general-purpose programming, web development, and data analysis tasks that require a large ecosystem of libraries and tools.

Julia in a Nutshell

Fast

Julia was designed from the beginning for high performance. Julia programs compile to efficient native code for multiple platforms via LLVM.

Dynamic

Julia is dynamically typed, feels like a scripting language, and has good support for interactive use.

Reproducible

Reproducible environments make it possible to recreate the same Julia environment every time, across platforms, with pre-built binaries.

Composable

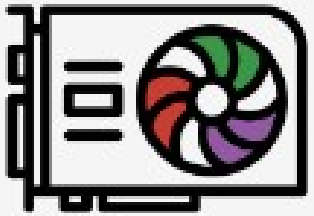
Julia uses multiple dispatch as a paradigm, making it easy to express many object-oriented and functional programming patterns. The talk on the Unreasonable Effectiveness of Multiple Dispatch explains why it works so well.

General

Julia provides asynchronous I/O, metaprogramming, debugging, logging, profiling, a package manager, and more. One can build entire Applications and Microservices in Julia.

Open source

Julia is an open source project with over 1,000 contributors. It is made available under the MIT license. The source code is available on GitHub.



CUDA.jl

CUDA programming in Julia

The CUDA.jl package is the main entrypoint for programming NVIDIA GPUs in Julia. The package makes it possible to do so at various abstraction levels, from easy-to-use arrays down to hand-written kernels using low-level CUDA APIs.

interactivo

```
Julia

julia> import Pkg; Pkg.add("CUDA")

julia> using CUDA

julia> CUDA.versioninfo()
```

```
module load julia-….

Julia myprog.jl

nvprof myprog.jl

qsub jobGPU
```

https://cuda.juliagpu.org/stable/

Mapeo logístico

$$x_{n+1} = r x_n (1 - x_n), \quad 0 \le r \le 4$$

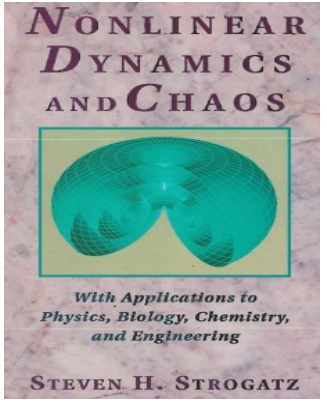
In a fascinating and influential review article, Robert May (1976) emphasized that even simple nonlinear maps could have very complicated dynamics. The article ends memorably with "an evangelical plea for the introduction of these difference equations into elementary mathematics courses, so that students' intuition may be enriched by seeing the wild things that simple nonlinear equations can do."

Robert May, Baron May of Oxford

Científico

Nacimiento: 8 de enero de 1936, Sídney, Australia

Fallecimiento: 28 de abril de 2020, Oxford, Reino Unido

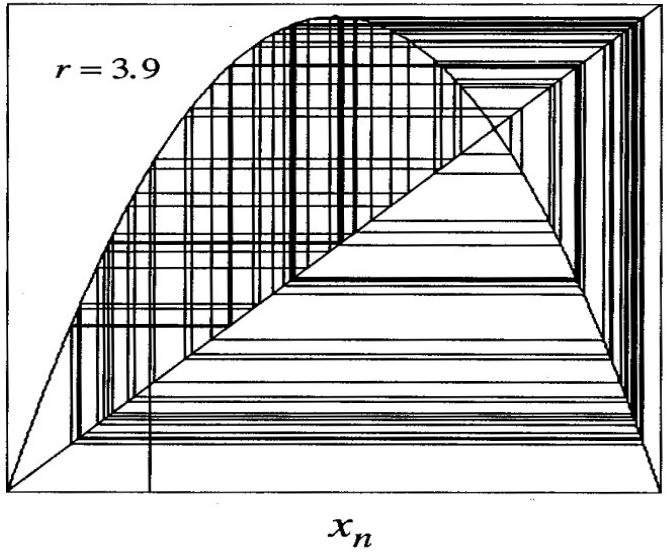


Converge a un punto fijo… ¿qué otra cosa puede pasar en general?

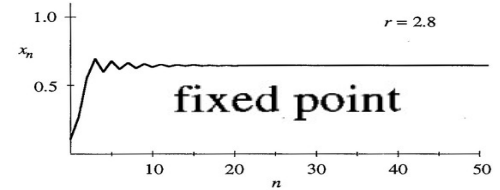
Mapeo logístico

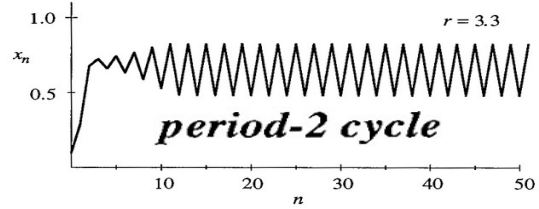
$$x_{n+1} = r x_n (1 - x_n), \quad 0 \le r \le 4$$

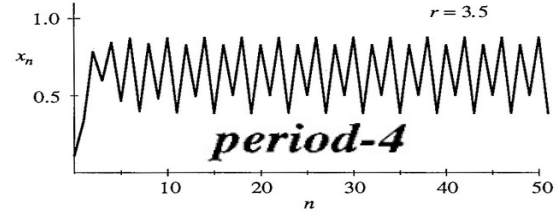
Esto tiene una interpretación biológica. Ej "reproducción de conejos"



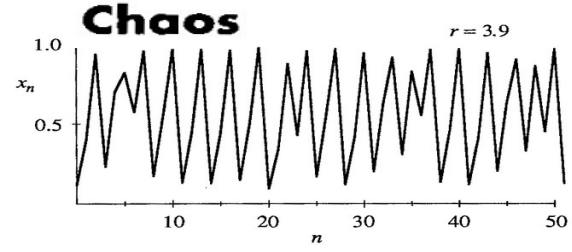
r







....



$r_1 = 3$	(period 2 is born)
$r_2 = 3.449...$	4
$r_3 = 3.54409...$	8
$r_4 = 3.5644...$	16
$r_5 = 3.568759...$	32
⋮	⋮
$r_\infty = 3.569946..$	∞

La paralelización está en hacer la cuenta para distintos valores de r

Chaos and Periodic Windows

- Manos a la obra

Liapunov Exponent

$$\lambda \approx \frac{1}{n} \ln \left| \prod_{i=0}^{n-1} f'(x_i) \right|$$

Paralelización sobre 0<= r <= 4

ICNPG_julia_logistic.ipynb

En el código de ejemplo de julia no se usa tipo notebook, sino que se escribe el archivo, se guarda, se compila y luego se ejecuta