Programación para Física y Astronomía

Departamento de Física.

Profesores: Claudia Loyola/ Joaquín Peralta
Felipe Moreno/ Fabián Gómez-Villafañe

Primer Semestre / 2023

Universidad Andrés Bello

Universidad Andrés Bello®

Resumen

Clases

Clusters de Computadores

GPU Computing

Multiprocessing

Actividades

Clases

Resumen

- Clases : Estructuras ’abstractas’ que nos sirven para crear objetos.
- Objetos : Elementos creados a partir de una clase que tienen propiedades y métodos.
- Cuando creamos clases, simplificamos nuestros programas. Reutilizamos códigos y estructuramos de mejor forma el desarrollo de un/el software.
- Revisión la guía

```python
#Este programa lee un fichero de datos y grafica
#la primera y segunda columna de esos datos (X vs Y)
import numpy as np
import matplolib.pyplot as plt

#Cargamos los datos del fichero
datos = np.loadtxt("datos.dat")

#Asignamos las columnas a x e y
x = datos[:,0]
y = datos[:,1]

#Graficamos y Desplegamos el grafico
plt.plot (x, y)
plt.show()
```

Cambiemos este simple programa, y construyamos una clase que podamos reutilizar cada vez que queramos graficar un par de columnas desde un archivo.

- Renombremos este programa a : *miclase.py*
- Y cambiemos el contenido a lo siguiente:

```
#Esta clase grafica dos columnas de un archivo
import numpy as np
import matplolib.pyplot as plt

class Grafico():
    ca, cb = 0, 1
    fichero = 'datos.txt'
    def __init__(self, a, b, f):
        self.ca, self.cb = a, b
        fichero=f
    def Show(self):
        datos = np.loadtxt(self.fichero)
        X, Y = datos[:,self.ca], datos[:,self.cb]
        plt.plot (X, Y)
        plt.show()
```

Usemos nuestra clase

Ahora que ya grabamos nuestro fichero *miclase.py* procedamos a utilizar la clase. Para ello creamos un nuevo programa llamado *analisis1.py*

```
1  #Cargamos nuestra clase
2  from miclase import Grafico
3
4  #Graficamos columnas 0 y 1 del archivo datos.txt
5  g = Grafico(0, 1, 'datos.txt')
6
7  #Mostramos el grafico
8  g.Show()
9
```

Como pueden observar, si construimos esta clase, la podemos reutilzar en todos nuestros códigos de ahora en adelante. Y podemos graficar cualquiera de las dos columnas que deseemos, es muy util aprender a utilziar clases.

Clusters de Computadores

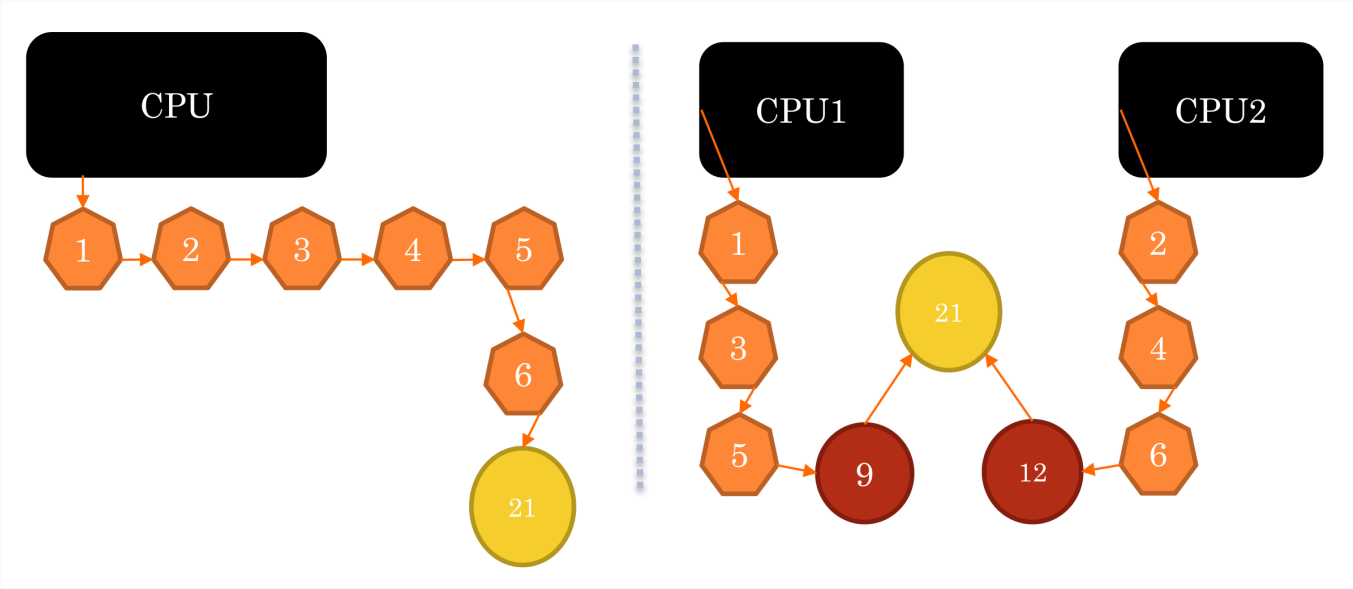
El rol de los clústers de cómputo

- Un clúster de cómputo es un grupo de computadores acoplados que trabajan juntos, de tal forma que pueden ser vistos como un solo computador.
- Usualmente son conectados a través de redes locales 1Gbps (slow) / Infiniband $\geq$ 56Gbps (fast)
- Su aplicabilidad es variada, desde *e-commerce*, HPC-*databases*, HPC-*scientific software*, etc.
- Su importancia
  - Precio/Performance
  - Disponibilidad
  - Escalabilidad

Clusters de Computadores

Software en Clusters

- Generalmente, se requieren software personalizados, y enfocados a ciertos problemas particulares, por lo que programar un código propio es “casi siempre” una excelente alternativa.
- ¿Cómo se organizan las CPU para trabajar en paralelo?
  - Sumando de uno a 6



Y El Futuro

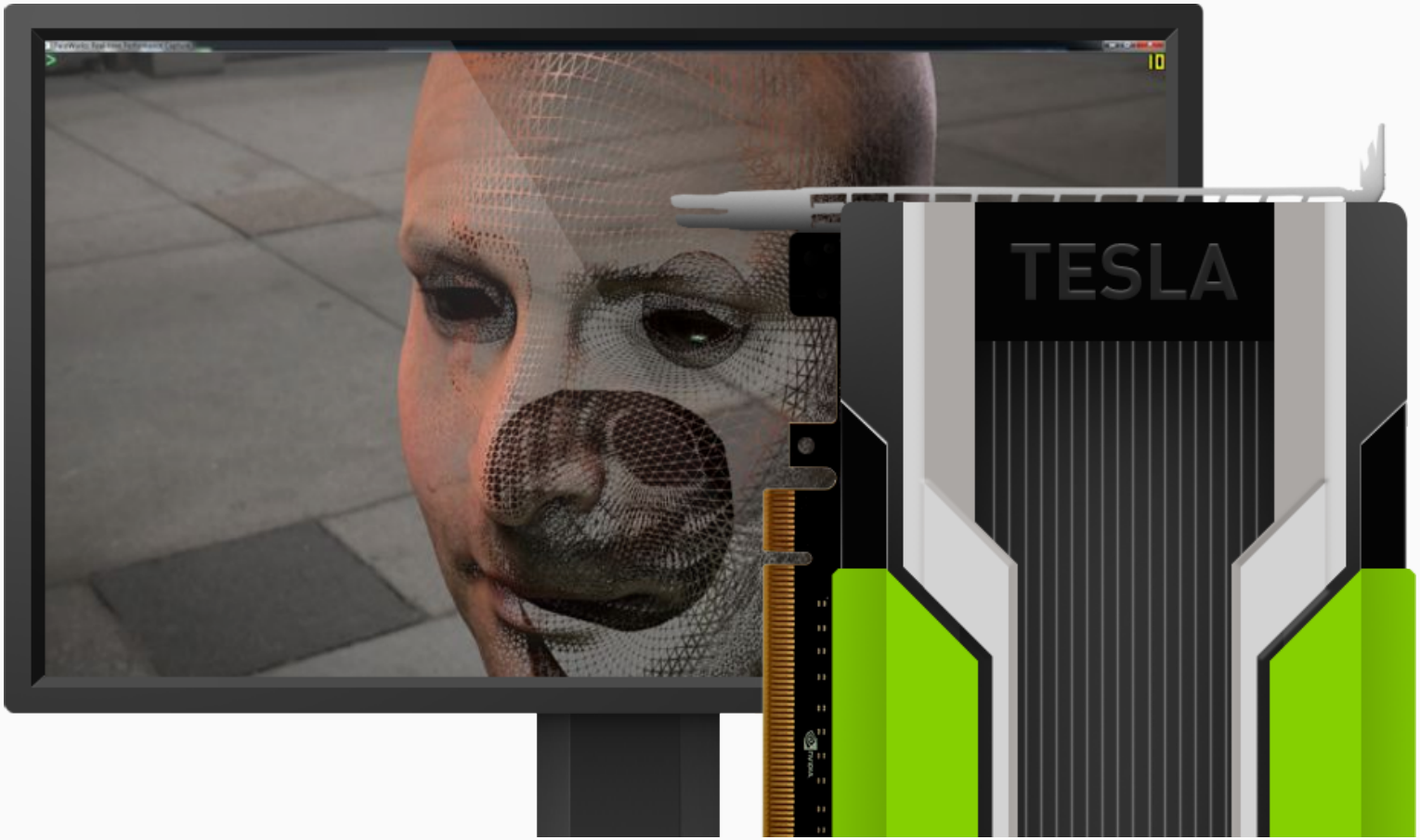
Sólo unir más y más computadores . . .

¿?

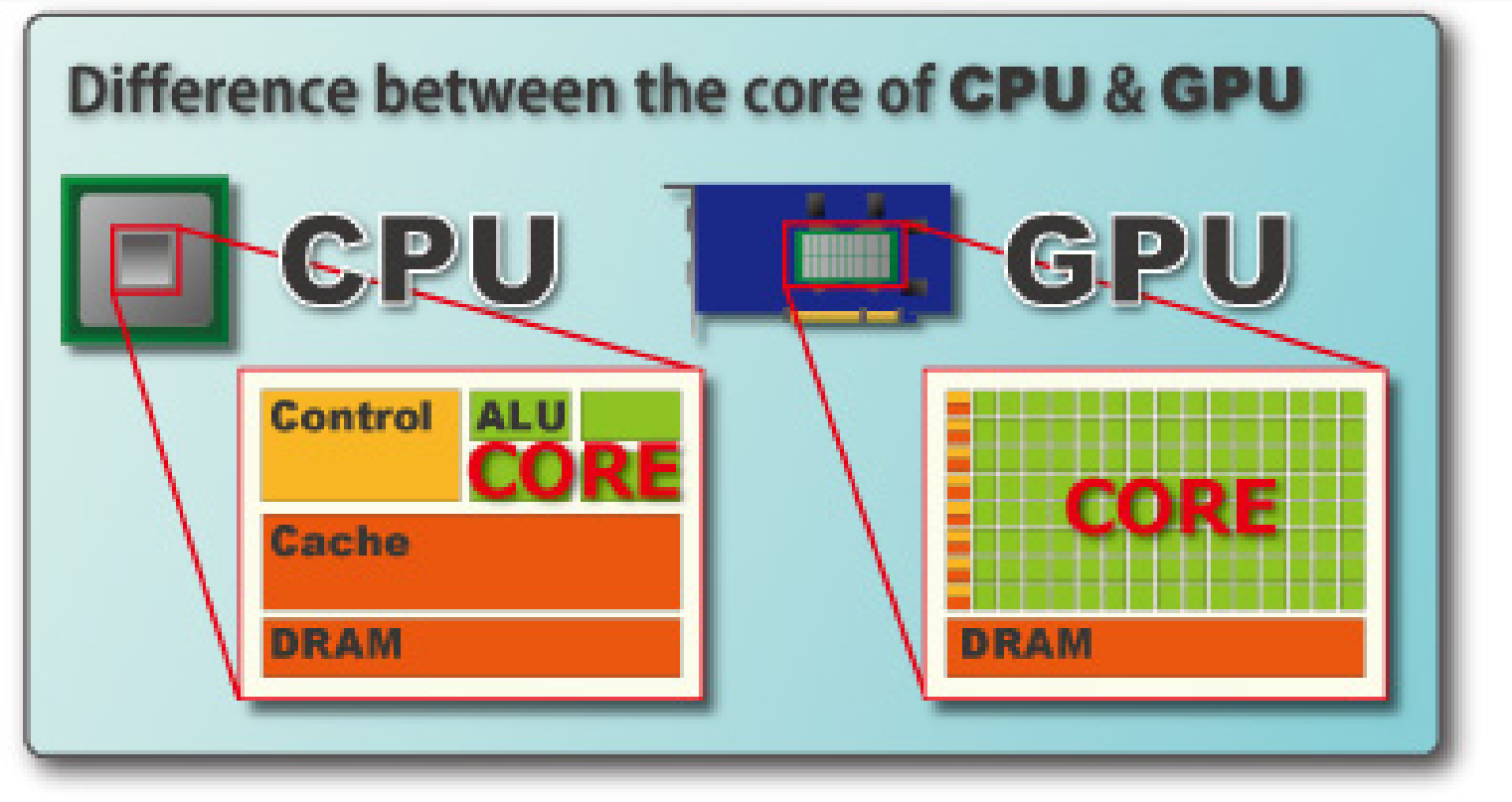
GPU Computing

Graphics Processing Units

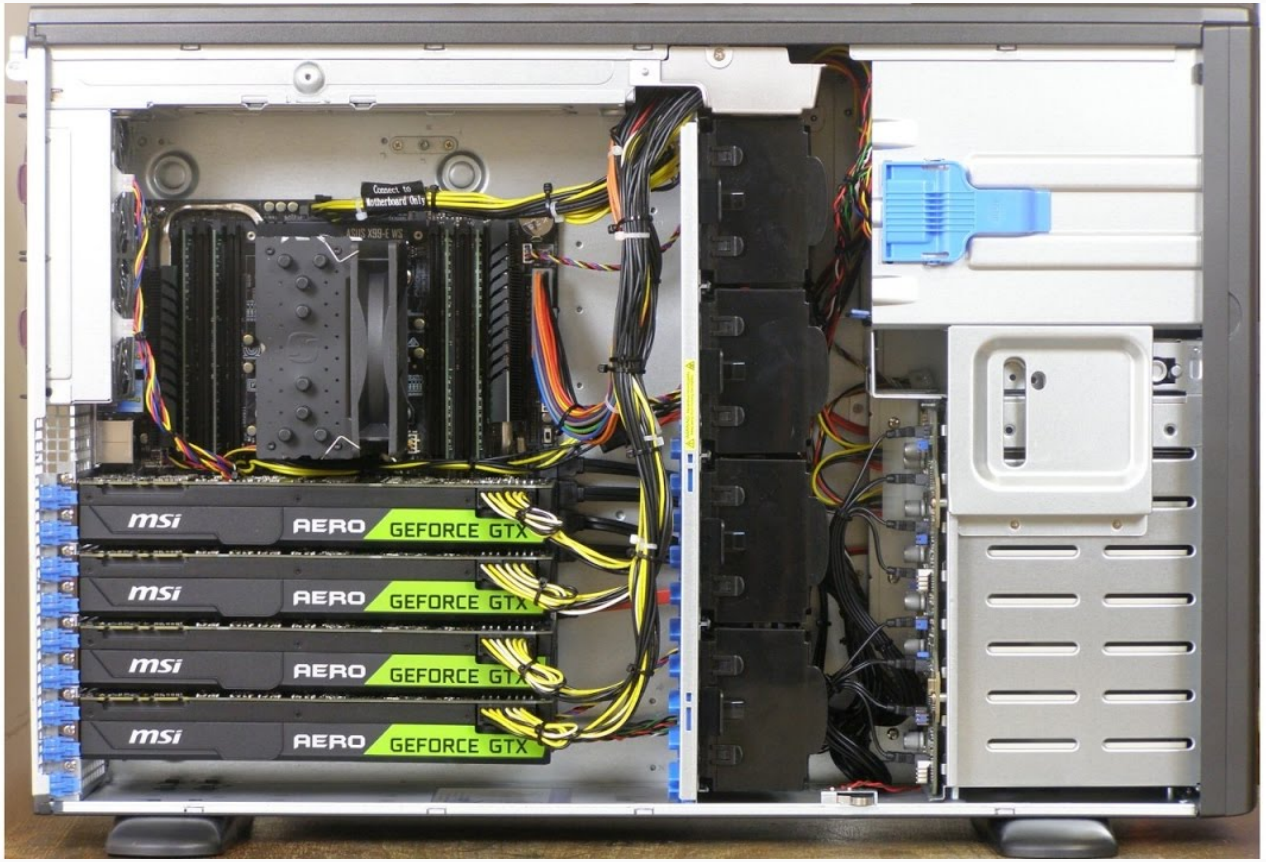
Graphics Processing Units



Graphics Processing Units



Graphics Processing Units



Graphics Processing Units

- Si bien es muy rentable (hasta 200X más rápido, para casos especiales), requiere más herramientas de programación en distintos lenguajes.
- Una ventaja es su bajo costo, pero lleva también un alto consumo eléctrico.
- Podemos convertir un PC de escritorio, en un verdadero cluster de cómputo con una sola tarjeta GPU.
- No tiene tanta madurez como MPI.

Graphics Processing Units

- Recientemente se ha revelado su potencial como arquitectura de cálculo para computación en general (GPGPU, General Purpose GPU) y por sobre todo para computación científica!
- Con las capacidades de memoria y velocidad de cómputo de las recientes generaciones de GPU, es posible conseguir ganancias en eficiencia de un orden de magnitud o más con respecto a los tiempos de CPU.
- Usos de GPU en computación científica
  - Simulaciones de elementos finitos.
  - Simulaciones Monte Carlo.
  - Dinámica Molecular
  - Calculos de Estructura Electrónica
  - Y muchos más.

Multiprocessing

Multiprocessing

La librería *`multiprocessing`* de Python es una librería de alto nivel que nos permite ejecutar código de forma paralela.

- Es una librería de alto nivel; no es necesario tener conocimiento profundo sobre como funciona internamente la ejecución en paralelo.
- Es fácil de usar: Su sintaxis aprovecha las ventajas del lenguaje de programación Python.
- Estándar: Viene por defecto en una instalación estándar de Python, por lo que no necesitamos instalar paquetes adicionales.

Process

El objeto *Process* permite crear un proceso paralelo. En el siguiente ejemplo la línea

```
1 if __name__=="__main__":
```

es necesaria para que el código principal se ejecute solo en el proceso principal.

```
1 from multiprocessing import Process
2
3 def f(x):
4     print(x**2)
5
6 if __name__=="__main__":
7     p = Process(target=f, args=(5,))
8     p.start()
9     p.join()
```

Process

Algunas cosas a considerar:

- El constructor crea el proceso y lo asocia con una función. Para que el proceso inicie hay que invocar el método *`start()`*.
- El método *`join()`* es para agrupar procesos de modo que el proceso principal espere a que todos los procesos paralelos terminen para continuar con la ejecución.

¿Cómo verificamos que el código ejecuta efectivamente en paralelo? Para ello veamos el siguiente código.

Process

```python
from multiprocessing import Process
from time import sleep
from timeit import default_timer as timer

def f(x):
    sleep(2) #Pausa de 2 segundos para simular proceso pesado
    print(x**2)

def ejecucion_en_serie():
    for i in range(1,5):
        f(i)

def ejecucion_en_paralelo():
    processes = []
    for i in range(1,5):
        p = Process(target=f, args=(i,))
        p.start()
        processes.append(p)

    for p in processes:
        p.join()

if __name__=="__main__":
    start = timer()
    ejecucion_en_serie()
    print(f"Tiempo en serie: {timer()-start:.2f}s")

    start = timer()
    ejecucion_en_paralelo()
    print(f"Tiempo en paralelo: {timer()-start:.2f}s")
```

Process

Al ejecutar este código veremos una salida como esta

```
1
4
9
16
Tiempo en serie: 8.00s
1
4
9
16
Tiempo en paralelo: 2.01s
```

Esto muestra que la segunda vez las 4 invocaciones de la función *f* fueron realizadas en paralelo.

Pipes

Sin embargo, lo anterior solo ejecuta procesos en paralelo sin entregarnos información. ¿Cómo nos comunicamos con estos procesos? Para eso utilizaremos tuberías de comunicación

```python
from multiprocessing import Process, Pipe

def f(conn, x):
    conn.send(x**2)

if __name__=="__main__":
    conn1, conn2 = Pipe()

    p = Process(target=f, args=(conn1, 5))
    p.start()
    p.join()

    value = conn2.recv()
    print(f"El valor entregado por el proceso remoto es {value}.")
```

Pool

Una forma sencilla de ejecutar una función en paralelo para distintos valores de entrada es usar *Pool*. Este objeto utiliza una sintaxis similar a la función *map* vista anteriormente, pero ejecuta cada proceso en paralelo.

```
1  from multiprocessing import Pool
2
3  def f(x):
4      return x**2
5
6  if __name__=="__main__":
7      p = Pool(4)
8      values = p.map(f, range(1,5))
9
10     print(f"Los valores entregados son {values}.")
```

Más información

https://info.gwdg.de/dokuwiki/doku.php?id=en:services:application_services:high_performance_computing:courses

Actividades

Actividades

(a) Esta primera actividad consiste en calcular el cuadrado de números en una lista. Lo haremos de dos formas:

- La forma secuencial estándar (sin multiprocesamiento).
- Usando el módulo de multiprocesamiento.

Planteamiento del problema:
Tiene una lista de números del 1 al 100.000. Su tarea es calcular el cuadrado de cada número en la lista y devolver una nueva lista con los valores cuadrados.
Las dos versiones de su programa deberían realizar la misma tarea, pero la versión de multiprocesamiento debería hacerlo más rápido (en una máquina de varios núcleos).

(b) Escriba un programa en Python3 usando multiprocesamiento para un gran set de números (por ejemplo de 1 hasta 1e6). Determine cuáles son números primos.