PARALLEL PHYTON

Elección del trabajo:

En mi caso he elegido la parte práctica, debido a que me parece mas entretenido aprender la parte práctica de la programación.

Para hacer esta parte práctica, he decidido realizar una de las practicas echas anteriores en el laboratorio, en este caso, el calculo de Pi a partir de Parallel Python. He elegido esta práctica, debido a que con las demás se requería de una representación gráfica, y en mi caso, no tenía los conocimientos necesarios para dicha representación en un tiempo razonable.

Inicio del trabajo:

Una vez elegida la práctica a realizar, he procedido a la documentación de la página oficial de Parallel Python para aprender loa principios básicos necesarios para utilizar su librería.

En mi caso he utilizado el ID de PyCharm, debido a que ya estoy acostumbrado a su utilización y además facilita la programación en Python. Esta ID te ayuda a crear un espacio virtual para instalar las librerías necesarias para la ejecución del programa, en este caso, he instalado los siguientes paquetes:

Package	Versión
Pip	19.3.1
Pp	1.6.5
Setuptools	40.0.0

En primera instancia intente utilizar Python 3.7 pero al intentar instalar el paquete pp me daba distintos errores y debido a ello pase a la versión de Python 2.7 y no tuve ningún problema es su instalación, debido a ello me quede con la versión antigua de Python.

Programación:

Una vez ya está toda lista, ya podremos proceder a la programación del código. En mi caso he hecho un programa para ejecutar con la consola.

código

```
import pp
import sys

'''
Recibe por parámetro :
- myId -> número id del trabajador.
- howmany -> cuantos trabajadores hay.
- num_steps -> numero de pasos para calcular pi (cuantos más pasos, mas
preciso es el resultado)
- steps la precisión de cada step

Función:
- Cada trabajador calcula la suma dependiendo de su ID, es decir que cada
trabajador solo recorre una parte de num_steps y luego devuelve la suma
obtenida.

Devuelve:
- Devuelve la suma obtenida en forma de float.
'''
def pi_parts(myid, howmany, num_steps, step):
    sum = 0.0
    for i in range(myid, num_steps, howmany):
        x = (i + 0.5) * step
        sum += 4.0 / (1.0 + x * x)
    return sum


print """Usage: python get_pi.py [ncpus] [nsteps]
    [ncpus] -El numero de trabajadores que se ejecutan en paralelo,
    si se omite el valor, será igual al numero de cores disponibles en el
sistema.
    [nsteps] - El numero de pasos para la precisión del numero pi,
    si se omite el valor será igual a 100000. (Para introducir el numero de
steps se tiene que introducir primero el numero de cpus)
"""

ppservers = ()
'''
Dependiendo de cómo ejecutemos el programa cojera un tipo de parámetros u
otros.
'''
if len(sys.argv) == 1:
    job_server = pp.Server(ppservers=ppservers)
    ncpus = job_server.get_ncpus()
    num_steps = 100000
elif len(sys.argv) == 2:
    ncpus = int(sys.argv[1])
```

```
    job_server = pp.Server(ncpus, ppservers=ppservers)
    num_steps = 100000
elif len(sys.argv) == 3:
    ncpus = int(sys.argv[1])
    job_server = pp.Server(ncpus, ppservers=ppservers)
    num_steps = int(sys.argv[2])
else:
    print "El numero máximo de valores son 2, el número de cpus y el número
           de steps, en este orden."
    exit(0)

print "Empezando pp con", ncpus, "trabajadores:"

step = 1.0 / float(num_steps)
'''
Se crean tantas tareas como numero de procesadores hayamos determinado.
'''
jobs = [(cpu_id, job_server.submit(pi_parts, (cpu_id, ncpus, num_steps,
         step), (pi_parts,), ())) for cpu_id in range(ncpus)]

sum = 0.0
'''
Esperamos a que cada trabajador termine y obtenemos su valor para sumarlos
todos y obtener el numero Pi.
'''
for input, job in jobs:
    sum += job()
    print "Suma de pi del trabajador", input, "es", job()

pi = step * sum
print("\nNumero pi calculado: %.15f\n" % round(pi, 15))

#Imprimimos la información generado por el ParallelPython
job_server.print_stats()
```

Como ejecutar el programa:

Para poder ejecutar el programa sin la necesidad de instalar ningún paquete hemos creado un espacio virtual al que podremos acceder fácilmente des de nuestra terminal con virtualenv.

Pero primero nos tenemos que asegurar de que nuestro Python 2.7 para estar totalmente seguros de que se ejecute correctamente.

Windows

Entramos con la consola Power Shell dentro de la carpeta del programa y ejecutamos el siguiente comando para poder entrar en el Virtualenv:

.\venv\Scripts\activate.ps1

Una vez dentro del entorno virtual ya podremos ejecutar el programa sin ningún problema utilizando el siguiente comando:

Python .\get_pi.py [ncpus] [nsteps]

```
PS C:\Users\tonix\Desktop\ParallelPython2> .\venv\Scripts\activate.ps1
(venv) PS C:\Users\tonix\Desktop\ParallelPython2> python .\get_pi.py
Usage: python get_pi.py [ncpus] [nsteps]
    [ncpus] -El numero de trabajadores que se ejecutan en paralelo,
    si se omite el valor, sera igual al numero de cores disponibles en el sistema.
    [nsteps] - El numero de pasos para la precision del numero pi,
    si se omite el valor sera igual a 100000. (Para introducir el numero de steps s
ero de cpus)

Empezando pp con 4 trabajadores:
Suma de pi del trabajador 0 es 78540.5663375
Suma de pi del trabajador 1 es 78540.0663425
Suma de pi del trabajador 2 es 78539.5663425
Suma de pi del trabajador 3 es 78539.0663375

Numero pi calculado: 3.141592653598117

Job execution statistics:
 job count | % of all jobs | job time sum | time per job | job server
         4 |        100.00 |       0.0400 |     0.010000 | local
Time elapsed since server creation 0.0299999713898
0 active tasks, 4 cores

(venv) PS C:\Users\tonix\Desktop\ParallelPython2>
```

Linux

Para entrar en el entorno virtual creado solo necesitaremos en siguiente comando:

source venv\Scripts\activate

y ya podremos proceder a ejecutar el programa sin ningún problema igual que con Windows.

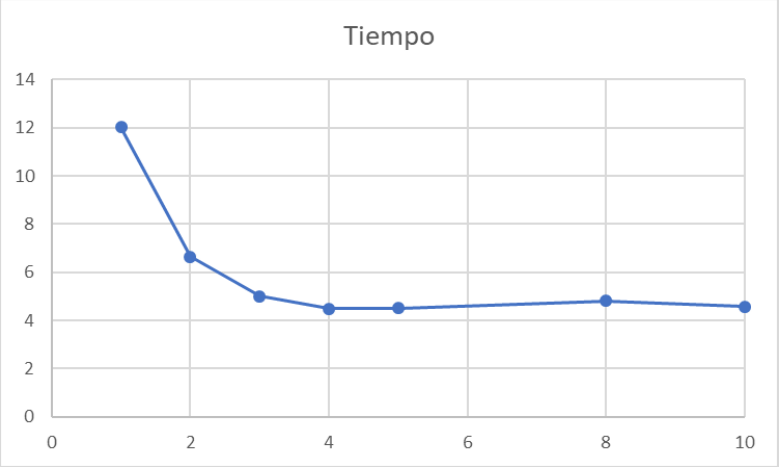
Otras opciones:

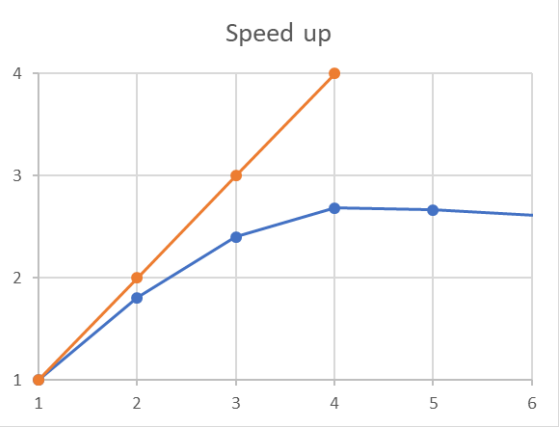
Si no deseamos instalar virtualenv también podemos proceder a instalar todos los paquetes necesarios para ejecutarlo directamente des de nuestro sistema.

Resultados de la ejecución:

Una vez ya está toda lista podemos proceder a la ejecución del programa. Teniendo en cuanta que mi ordenador solo tiene 4 procesadores, podemos observar el resultado de las pruebas de ejecución con nsteps igual a 50000000:

nCores	Tiempo	speed up
1	12,0149	1
2	6,6510	1,80648023
3	5,0039	2,40110714
4	4,4759	2,68435398
5	4,5120	2,66287677
8	4,7999	2,50315632
10	4,5699	2,62913849





Los resultados pueden variar dependiendo de la carga del ordenador en un momento determinado, pero por lo general, no varía demasiado.

En este caso podemos deducir que, a partir de 4 procesadores, ya no es paralelismo sino concurrencia, por lo que ya no genera una mejora en el tiempo.