

a) Se implemento el programa con el nombre correspondiente de manera sincrona y se adjunta el tiempo de ejecución

```
TERMINAL OUTPUT PROBLEMS 1 DEBUG CONSOLE JUPYTER
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> python.exe .\norm_vectorial_sincrono.py
Tiempo de ejecucion serial: 0.0005016399998567067 segundos
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> █
```

Se realizó 5 medidas consecutivas del calculo para luego sacar el promedio de estas y tener una medida mas significativa para poder comparar con los otros códigos.

b) Se implementó el programa nuevamente pero ahora en multihilo y se adjunta tiempo de ejecución

```
Tiempo de ejecucion serial: 0.0005016399998567067 segundos
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> python.exe .\norm_vectorial_multihilo.py
Tiempo de ejecucion multihilo: 0.0010408000001916663 segundos
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> █
```

c) Se implemento el programa ahora en multiprocessing y se adjunta el tiempo

```
Tiempo de ejecucion multiproceso: 0.2839405000000743 segundos
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> python.exe .\norm_vectorial_multiprocessing.py
Tiempo de ejecucion multiproceso: 0.28465661999994413 segundos
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> █
```

d) Ahora se haya el speedup correspondiente de las implementaciones, el síncrono con respecto a al multihilo y al multiprocessing

```
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> python.exe .\speedup.py
Tiempo de ejecucion serial: 0.0005230399998254143 segundos
Tiempo de ejecucion multihilo: 0.0010256199995637872 segundos
Tiempo de ejecucion multiproceso: 0.32912089999736055 segundos
Speed up síncrono con multihilo: 0.5099744545229926
Speed up síncrono con multiprocess: 0.0015892032375628801
PS C:\Users\djver\OneDrive\Documentos\Catolica\2023-1\ARQUITECTURA DE COMPUTADORAS (1IEE14-0621)\final> █
```

e) Comentarios y Conclusiones

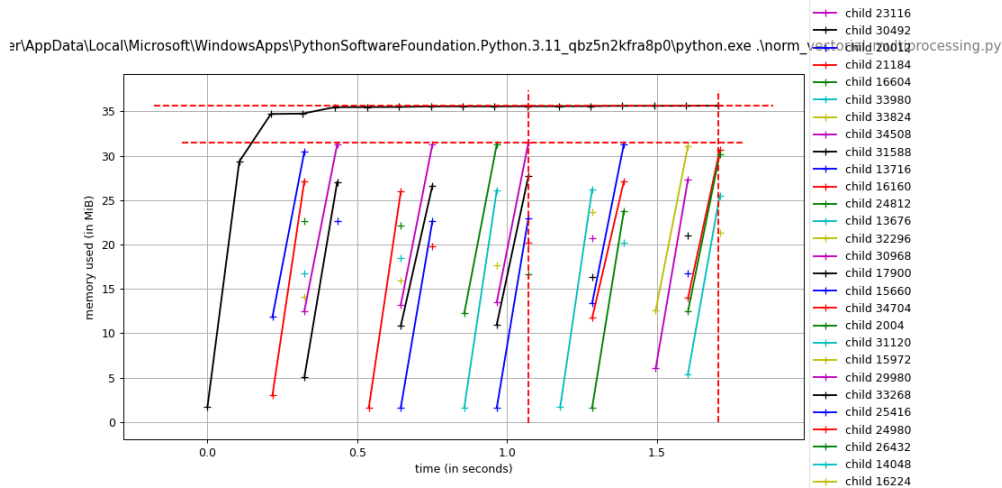
Después de hacer respectivas mediciones al pc utilizado para el código se puede observar mediante el uso de -m mprof run que la cantidad óptima para la CPU de 24 núcleos y 32 hilos es de 8 ya que mayores a estos se ve que el tiempo incrementa considerablemente. Esto se puede lograr al aplicar iteraciones cambiando el pool process principalmente para el multiprocessing e iterando un par de veces para sacar la media.

Lastimosamente en mis pruebas no se ha podido apreciar una mejora al pasar del código síncrono al multihilo y multiprocessing . Esto se debe principalmente por 2 posibles causas:

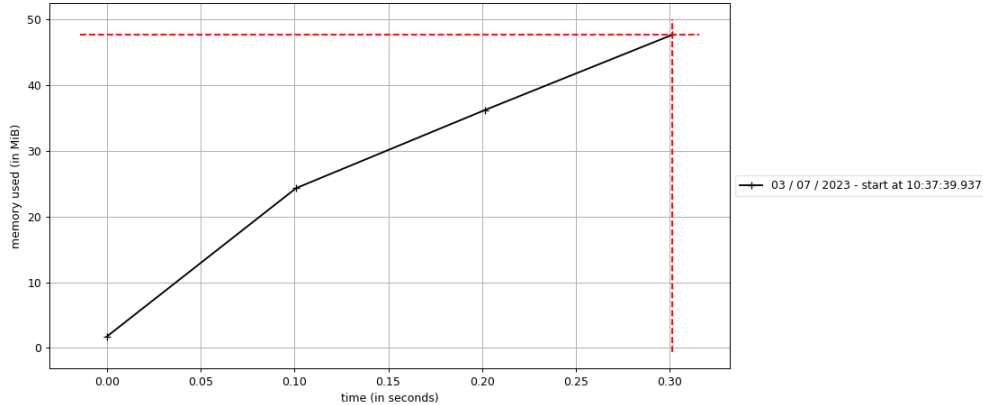
1. Por mala implementación de esto, en la parte del threading se utilizo varios for en vez de utilizar un pool el cual directamente podría dar los valores de manera vectorial para no estar iterando ya que estas iteraciones generan más tiempos de espera.

2. Por ser un calculo relativamente sencillo para el cpu que poseo, anteriormente con otros ejemplos del profesor también pasó algo similar ya que al llamar mas núcleos o hilos termina aumentando el tiempo por cargar variables y abrir otros interpretes

Se adjuntan imágenes de los Consumos de memoria footprint para cada una de las implementaciones. Podemos observar que el que mas consumió memoria es el multiprocessing incluso limitándolo a solo 8 procesos ya que tiene que cargar su propia memoria y el intérprete de Python.



Path: d:\jver\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe .\norm_vectorial_sincrono.py



Path: d:\jver\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe .\norm_vectorial_multihilo.py

