IMT2112: Algoritmos paralelos en computación científica 2 de 09 de 2020

Tarea 1

k-Means

Alumno: Lucas Suárez *Profesor: Elwin van't Wout*

1. Detalles previos

Para realizar las implementaciones se consideró lo siguiente:

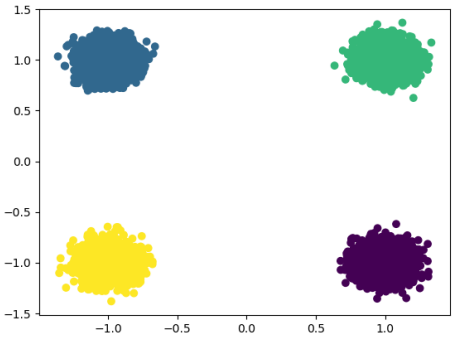
- La dimensión de los datos es $d = 20$.
- Los valores de n (tamaño del dataset) son 1000, 5000, 10000, 20000, 30000 y 40000. En el archivo adjunto **tarea1.py** puede modificar el valor de n (linea 9) para reproducir los resultados.
- Se escogió $k = 4$, es decir, buscamos 4 clusters o centros.
- Los datos fueron generados utilizando make_blobs para generar datos claramente separados. Los centros para la creación de los clusters son básicamente $[1, 1]$, $[-1, 1]$, $[-1, -1]$ y $[1, -1]$ seguidos de ceros en las otras 18 componentes. De esta manera, para plotear basta tomar las dos primeras componentes, lo que permite observar el correcto funcionamiento de los algoritmos en un plot.

2. Resultados

A continuación puede observar los resultados obtenidos con las 3 implementaciones distintas

n	For loops	Numpy	Multiprocessing
1000	0.35102 s	0.07684 s	1.63506 s
5000	2.11811 s	0.43244 s	2.48279 s
10000	4.16918 s	0.95593 s	3.58072 s
20000	7.24552 s	1.61283 s	5.05546 s
30000	13.83196 s	2.41541 s	7.92047 s
40000	15.64788 s	3.03876 s	8.45104 s

A continuación se muestra un ejemplo de k-Means con Multiprocessing y $n = 10000$, los colores representan los clusters.



Observación: de acuerdo a lo explicado anteriormente este es el plot de las dos primeras componentes.

3. Conclusiones y discusión

Tras correr los algoritmos se observa que

- La primera implementación, que sólo usa For loops abre un sólo proceso con cuatro hilos.
- La segunda implementación, que utiliza Numpy para vectorizar, abre un proceso con 4 hilos.
- La tercera implementación, que usa multiprocessing, corre 1 proceso que abre 4 subprocesos (uno para cada trabajador supongo) donde cada subproceso tiene 4 hilos.

Al parecer mi computador utiliza 4 hilos por defecto para correr los programas pues lo testié con otros códigos y siempre abre con 4 hilos.

Con respecto a la tabla de tiempos se observa que el orden de rapidez es el esperado de acuerdo a lo que vimos en clases: Numpy >Multiprocessing >For loops secuenciales. A pesar de que con pocos datos usar Multiprocessing es más lento que la ejecución secuencial, esto no es una contradicción con los resultados, pues se debe al costo fijo de generar la piscina de trabajadores en paralelo, el cual deja de ser relevante cuando se consideran más datos.

Por último, observe que entre las lineas 152 y 160 no se utilizó Multiprocessing ya que lo que se hace es sumar los resultados de la paralelización sobre los registros, que con 4 trabajadores significa sumar 4 vectores y usar Multiprocessing sería innecesario (porque el costo de usarlo excede a la ganancia).