

Instituto Tecnológico de Costa Rica
Área Académica Ingeniería en Computadores
Arquitectura de Computadores II
II Semestre 2023

Taller 1: OpenMP

Estudiante:

Oscar Jesús Méndez Granados

Carné:

2019150432

Profesor:

Ronald Eduardo García Fernández



Punto 3

a- Compile ambos códigos y compare sus tiempos de ejecución

Pi.c

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
oscardmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o pi pi.c
oscardmendez@mendezos:~/Documents/Taller OpenMP$ ./pi
pi with 100000000 steps is 3.141593 in 0.192778 seconds
oscardmendez@mendezos:~/Documents/Taller OpenMP$
```

Pi_loop.c

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
oscardmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o pi_loop pi_loop.c
oscardmendez@mendezos:~/Documents/Taller OpenMP$ ./pi_loop
num_threads = 1 computed pi = 3.141592653589426 in 0.165134530990766 seconds threads = 1 % error = 0.00000000020257
num_threads = 2 computed pi = 3.141592653589910 in 0.095898839001105 seconds threads = 2 % error = 0.00000000003017
num_threads = 3 computed pi = 3.141592653589570 in 0.066154154999822 seconds threads = 3 % error = -0.000000000006997
num_threads = 4 computed pi = 3.141592653589683 in 0.050188703000458 seconds threads = 4 % error = -0.00000000003421
oscardmendez@mendezos:~/Documents/Taller OpenMP$
```

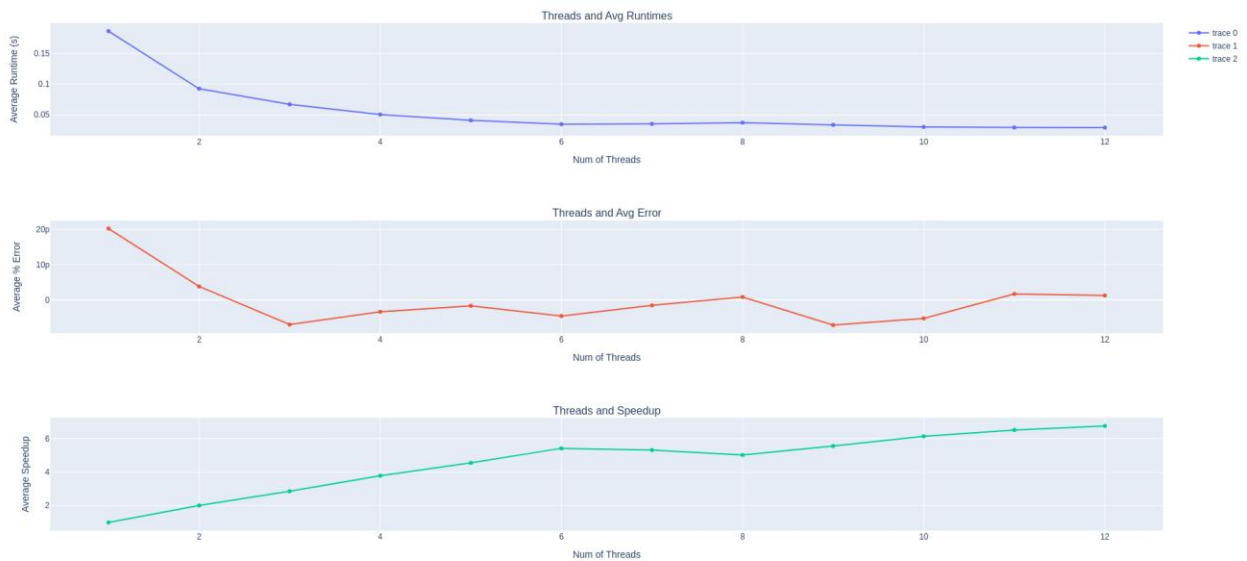
Realizando una comparación del tiempo de ejecución se puede observar que los tiempos de ejecución en Pi_loop son menores y esto es se debe a que OpenMP permite dividir la tarea en fragmentos más pequeños que pueden ser ejecutados en paralelo por diferentes hilos y en este caso el código es lo suficientemente paralelizable como para aprovechar esos beneficios.

b- Modifique pi_loop.c de forma que puede ejecutar el número de threads disponible en su sistema

```
oscardmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o pi_loop pi_loop.c
oscardmendez@mendezos:~/Documents/Taller OpenMP$ ./pi_loop
Max threads available: 12
num_threads = 1 computed pi = 3.141592653589426 in 0.205383737000408 seconds threads = 1 % error = 0.00000000020257
num_threads = 2 computed pi = 3.141592653589910 in 0.102714609998932 seconds threads = 2 % error = 0.00000000003017
num_threads = 3 computed pi = 3.141592653589570 in 0.073348905998500 seconds threads = 3 % error = -0.000000000006997
num_threads = 4 computed pi = 3.141592653589683 in 0.052280301000792 seconds threads = 4 % error = -0.000000000003421
num_threads = 5 computed pi = 3.141592653589737 in 0.043565853000473 seconds threads = 5 % error = -0.00000000001696
num_threads = 6 computed pi = 3.141592653589646 in 0.040981840000313 seconds threads = 6 % error = -0.000000000004594
num_threads = 7 computed pi = 3.141592653589740 in 0.043050276999566 seconds threads = 7 % error = -0.00000000001583
num_threads = 8 computed pi = 3.141592653589815 in 0.043037608998929 seconds threads = 8 % error = 0.000000000000792
num_threads = 9 computed pi = 3.141592653589266 in 0.040680835001074 seconds threads = 9 % error = -0.000000000007153
num_threads = 10 computed pi = 3.141592653589624 in 0.036385760000400 seconds threads = 10 % error = -0.000000000005273
num_threads = 11 computed pi = 3.141592653589842 in 0.045888730000538 seconds threads = 11 % error = 0.000000000001668
num_threads = 12 computed pi = 3.141592653589828 in 0.050030500999912 seconds threads = 12 % error = 0.00000000001216
oscardmendez@mendezos:~/Documents/Taller OpenMP$
```

c- Ejecute pi_loop.c múltiples veces (~100) y almacene los resultados en un directorio results por ejemplo pi_loop_01.txt, ..., pi_loop_100.txt

d- Utilice algún script o herramienta para graficar: tiempo de ejecución vs número de threads, porcentaje de error vs número de threads, y mejora (speedup) vs número de threads, considere el speedup como el tiempo de ejecución con un solo thread entre el tiempo de ejecución con múltiples threads puede referirse al ejemplo proporcionado `plotter.py`



Punto 4

a- Un programa que permita emplear múltiples threads para el cálculo de la constante e mediante el uso de series de Taylor con diferentes términos n (justifique cuantos términos va a emplear).

Inicialmente se hicieron varias pruebas con distintos valores de N , como se muestra a continuación:

Con $N = 4800$

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ gcc -fopenmp -o taylor_loop taylor_loop.c
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ ./taylor_loop
num_threads = 1 computed e = 2.718281828459046 in 0.052276668999866 seconds threads = 1 % error = 0.000000000000016
num_threads = 2 computed e = 2.718281828459046 in 0.039151190999974 seconds threads = 2 % error = 0.000000000000016
num_threads = 3 computed e = 2.718281828459046 in 0.029526479999959 seconds threads = 3 % error = 0.000000000000016
num_threads = 4 computed e = 2.718281828459046 in 0.023728424999945 seconds threads = 4 % error = 0.000000000000016
num_threads = 5 computed e = 2.718281828459046 in 0.019597612000098 seconds threads = 5 % error = 0.000000000000016
num_threads = 6 computed e = 2.718281828459046 in 0.016987560999951 seconds threads = 6 % error = 0.000000000000016
num_threads = 7 computed e = 2.718281828459046 in 0.014757446999965 seconds threads = 7 % error = 0.000000000000016
num_threads = 8 computed e = 2.718281828459046 in 0.013112243000023 seconds threads = 8 % error = 0.000000000000016
num_threads = 9 computed e = 2.718281828459046 in 0.011984150999979 seconds threads = 9 % error = 0.000000000000016
num_threads = 10 computed e = 2.718281828459046 in 0.011577749999978 seconds threads = 10 % error = 0.000000000000016
num_threads = 11 computed e = 2.718281828459046 in 0.011441304000073 seconds threads = 11 % error = 0.000000000000016
num_threads = 12 computed e = 2.718281828459046 in 0.020700322000039 seconds threads = 12 % error = 0.000000000000016
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$
```

Con $N = 10800$

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ gcc -fopenmp -o taylor_loop taylor_loop.c
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ ./taylor_loop
num_threads = 1 computed e = 2.718281828459046 in 0.252157481000040 seconds threads = 1 % error = 0.000000000000016
num_threads = 2 computed e = 2.718281828459046 in 0.189817185000038 seconds threads = 2 % error = 0.000000000000016
num_threads = 3 computed e = 2.718281828459046 in 0.141107559999969 seconds threads = 3 % error = 0.000000000000016
num_threads = 4 computed e = 2.718281828459046 in 0.112583419999964 seconds threads = 4 % error = 0.000000000000016
num_threads = 5 computed e = 2.718281828459046 in 0.094526764999955 seconds threads = 5 % error = 0.000000000000016
num_threads = 6 computed e = 2.718281828459046 in 0.088970048999916 seconds threads = 6 % error = 0.000000000000016
num_threads = 7 computed e = 2.718281828459046 in 0.071660211999970 seconds threads = 7 % error = 0.000000000000016
num_threads = 8 computed e = 2.718281828459046 in 0.063508063800004 seconds threads = 8 % error = 0.000000000000016
num_threads = 9 computed e = 2.718281828459046 in 0.057781430999967 seconds threads = 9 % error = 0.000000000000016
num_threads = 10 computed e = 2.718281828459046 in 0.052220785999997 seconds threads = 10 % error = 0.000000000000016
num_threads = 11 computed e = 2.718281828459046 in 0.060144893000012 seconds threads = 11 % error = 0.000000000000016
num_threads = 12 computed e = 2.718281828459046 in 0.062826710000081 seconds threads = 12 % error = 0.000000000000016
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$
```

Con $N = 24000$

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ ./taylor_loop
num_threads = 1 computed e = 2.718281828459046 in 1.217443206000098 seconds threads = 1 % error = 0.000000000000016
num_threads = 2 computed e = 2.718281828459046 in 0.919045062999985 seconds threads = 2 % error = 0.000000000000016
num_threads = 3 computed e = 2.718281828459046 in 0.693940780000048 seconds threads = 3 % error = 0.000000000000016
num_threads = 4 computed e = 2.718281828459046 in 0.570759699999826 seconds threads = 4 % error = 0.000000000000016
num_threads = 5 computed e = 2.718281828459046 in 0.457582257000013 seconds threads = 5 % error = 0.000000000000016
num_threads = 6 computed e = 2.718281828459046 in 0.388914294000039 seconds threads = 6 % error = 0.000000000000016
num_threads = 7 computed e = 2.718281828459046 in 0.340015509999830 seconds threads = 7 % error = 0.000000000000016
num_threads = 8 computed e = 2.718281828459046 in 0.304287625999995 seconds threads = 8 % error = 0.000000000000016
num_threads = 9 computed e = 2.718281828459046 in 0.273928825999974 seconds threads = 9 % error = 0.000000000000016
num_threads = 10 computed e = 2.718281828459046 in 0.248799011000084 seconds threads = 10 % error = 0.000000000000016
num_threads = 11 computed e = 2.718281828459046 in 0.229092749999836 seconds threads = 11 % error = 0.000000000000016
num_threads = 12 computed e = 2.718281828459046 in 0.231480634000036 seconds threads = 12 % error = 0.000000000000016
ooscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$
```

$N = 1200$

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
oscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$ ./taylor loop
num_threads = 1 computed e = 2.718281828459046 in 0.003218361000108 seconds threads = 1 % error = 0.000000000000016
num_threads = 2 computed e = 2.718281828459046 in 0.002677760000033 seconds threads = 2 % error = 0.000000000000016
num_threads = 3 computed e = 2.718281828459046 in 0.001967099999989 seconds threads = 3 % error = 0.000000000000016
num_threads = 4 computed e = 2.718281828459046 in 0.001667765000093 seconds threads = 4 % error = 0.000000000000016
num_threads = 5 computed e = 2.718281828459046 in 0.001331464000032 seconds threads = 5 % error = 0.000000000000016
num_threads = 6 computed e = 2.718281828459046 in 0.001062340000033 seconds threads = 6 % error = 0.000000000000016
num_threads = 7 computed e = 2.718281828459046 in 0.000973316000000 seconds threads = 7 % error = 0.000000000000016
num_threads = 8 computed e = 2.718281828459046 in 0.000865029999886 seconds threads = 8 % error = 0.000000000000016
num_threads = 9 computed e = 2.718281828459046 in 0.000773964000018 seconds threads = 9 % error = 0.000000000000016
num_threads = 10 computed e = 2.718281828459046 in 0.000719314000207 seconds threads = 10 % error = 0.000000000000016
num_threads = 11 computed e = 2.718281828459046 in 0.001936032999993 seconds threads = 11 % error = 0.000000000000016
num_threads = 12 computed e = 2.718281828459046 in 0.002166452999973 seconds threads = 12 % error = 0.000000000000016
oscarmendez@mendezos:~/Documents/GitHub/TallerOpenMP$
```

Para todas estas ejecuciones una de las que mejor comportamiento tuvo fue la de ejecución de N igual a 1200, además como se puede observar, todas las ejecuciones se realizaron con un N múltiplo de 12 y esto se debe a que el número de threads de mi equipo es 12 y, por lo tanto, un buen número N debía de ser uno divisible por 12 a fin de que se pueda distribuir de manera uniforme el trabajo entre los hilos, evitando desequilibrios en las cargas de trabajo. Por su parte, si se observan los valores de N, en la mayoría de ellos el error es bastante similar, y si varía este lo hace en el orden de los +15 decimales, es por esto que se considera que el valor de 1200 es un buen valor de N ya que no pone en riesgo la exactitud del cálculo y a su vez el valor es uno de los que mejor se comportó en una relación (exactitud/velocidad del cálculo).

b- Un programa que implemente la operación DAXPY de forma single thread y multithread (openMP), dicho programa deberá generar vectores aleatorios de tamaño definido N, considere las capacidades de su sistema para definir los valores de prueba (al menos 4 diferentes valores)

De igual forma en la que se realizó en el caso anterior con el cálculo de Euler por series de Taylor, a continuación, se presentan los resultados de DAXPY para diferentes valores de N:

N = 1200

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
oscarmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o daxpy daxpy_loop.c
oscarmendez@mendezos:~/Documents/Taller OpenMP$ ./daxpy
Computed DAXPY for vector size N = 1200
num_threads = 1 computed daxpy in 0.000002333999873 seconds threads = 1
num_threads = 2 computed daxpy in 0.000117418999223 seconds threads = 2
num_threads = 3 computed daxpy in 0.000071843000114 seconds threads = 3
num_threads = 4 computed daxpy in 0.000035377000131 seconds threads = 4
num_threads = 5 computed daxpy in 0.000024983000003 seconds threads = 5
num_threads = 6 computed daxpy in 0.000020280000299 seconds threads = 6
num_threads = 7 computed daxpy in 0.000025321000066 seconds threads = 7
num_threads = 8 computed daxpy in 0.000025049999390 seconds threads = 8
num_threads = 9 computed daxpy in 0.000022970000087 seconds threads = 9
num_threads = 10 computed daxpy in 0.000018376000033 seconds threads = 10
num_threads = 11 computed daxpy in 0.000019600000087 seconds threads = 11
num_threads = 12 computed daxpy in 0.000041099000555 seconds threads = 12
oscarmendez@mendezos:~/Documents/Taller OpenMP$
```

N = 24000

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o daxpy daxpy_loop.c
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ ./daxpy
Computed DAXPY for vector size N = 24000
num_threads = 1 computed daxpy in 0.000044960999730 seconds threads = 1
num_threads = 2 computed daxpy in 0.000064424999437 seconds threads = 2
num_threads = 3 computed daxpy in 0.000072963999628 seconds threads = 3
num_threads = 4 computed daxpy in 0.00008460999799 seconds threads = 4
num_threads = 5 computed daxpy in 0.000082765999650 seconds threads = 5
num_threads = 6 computed daxpy in 0.000037288999920 seconds threads = 6
num_threads = 7 computed daxpy in 0.000029113999517 seconds threads = 7
num_threads = 8 computed daxpy in 0.000035185000343 seconds threads = 8
num_threads = 9 computed daxpy in 0.000031649999983 seconds threads = 9
num_threads = 10 computed daxpy in 0.000029163999898 seconds threads = 10
num_threads = 11 computed daxpy in 0.000027503000638 seconds threads = 11
num_threads = 12 computed daxpy in 0.000024949999897 seconds threads = 12
● oscarmendez@mendezos:~/Documents/Taller OpenMP$
```

N = 72000

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o daxpy daxpy_loop.c
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ ./daxpy
Computed DAXPY for vector size N = 72000
num_threads = 1 computed daxpy in 0.000101709000774 seconds threads = 1
num_threads = 2 computed daxpy in 0.000168412999381 seconds threads = 2
num_threads = 3 computed daxpy in 0.000068905999797 seconds threads = 3
num_threads = 4 computed daxpy in 0.000046903000111 seconds threads = 4
num_threads = 5 computed daxpy in 0.00002112000222 seconds threads = 5
num_threads = 6 computed daxpy in 0.000055867999436 seconds threads = 6
num_threads = 7 computed daxpy in 0.000079679000009 seconds threads = 7
num_threads = 8 computed daxpy in 0.000056010999288 seconds threads = 8
num_threads = 9 computed daxpy in 0.000043445999836 seconds threads = 9
num_threads = 10 computed daxpy in 0.000042638000195 seconds threads = 10
num_threads = 11 computed daxpy in 0.000039128000026 seconds threads = 11
num_threads = 12 computed daxpy in 0.000069271000029 seconds threads = 12
● oscarmendez@mendezos:~/Documents/Taller OpenMP$
```

N = 144000

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o daxpy daxpy_loop.c
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ ./daxpy
Computed DAXPY for vector size N = 144000
num_threads = 1 computed daxpy in 0.000253797000369 seconds threads = 1
num_threads = 2 computed daxpy in 0.000223777000429 seconds threads = 2
num_threads = 3 computed daxpy in 0.000118820999887 seconds threads = 3
num_threads = 4 computed daxpy in 0.000101884000224 seconds threads = 4
num_threads = 5 computed daxpy in 0.000087569999418 seconds threads = 5
num_threads = 6 computed daxpy in 0.000097527000435 seconds threads = 6
num_threads = 7 computed daxpy in 0.000082358999862 seconds threads = 7
num_threads = 8 computed daxpy in 0.000085145999947 seconds threads = 8
num_threads = 9 computed daxpy in 0.000080570000116 seconds threads = 9
num_threads = 10 computed daxpy in 0.000127924000481 seconds threads = 10
num_threads = 11 computed daxpy in 0.000106651999886 seconds threads = 11
num_threads = 12 computed daxpy in 0.000758579999972 seconds threads = 12
● oscarmendez@mendezos:~/Documents/Taller OpenMP$
```

N = 240000

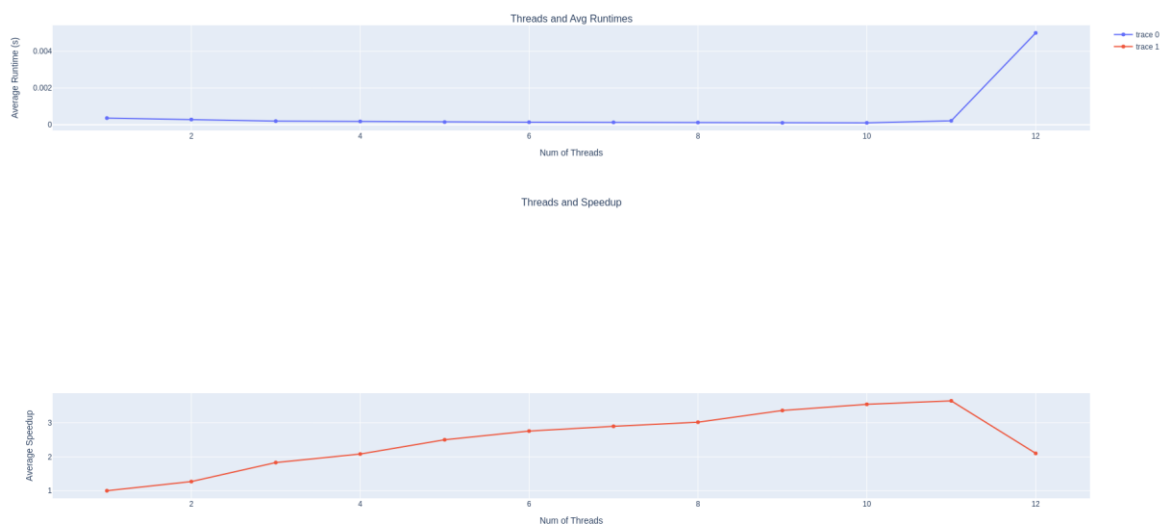
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ gcc -fopenmp -o daxpy daxpy_loop.c
● oscarmendez@mendezos:~/Documents/Taller OpenMP$ ./daxpy
Computed DAXPY for vector size N = 240000
num_threads = 1 computed daxpy in 0.000589671999478 seconds threads = 1
num_threads = 2 computed daxpy in 0.000393949000195 seconds threads = 2
num_threads = 3 computed daxpy in 0.000387566999962 seconds threads = 3
num_threads = 4 computed daxpy in 0.000287492999632 seconds threads = 4
num_threads = 5 computed daxpy in 0.000151300000653 seconds threads = 5
num_threads = 6 computed daxpy in 0.0001383930000270 seconds threads = 6
num_threads = 7 computed daxpy in 0.000135000999762 seconds threads = 7
num_threads = 8 computed daxpy in 0.000118245000522 seconds threads = 8
num_threads = 9 computed daxpy in 0.0001566290000230 seconds threads = 9
num_threads = 10 computed daxpy in 0.001243247999010 seconds threads = 10
num_threads = 11 computed daxpy in 0.002194951000092 seconds threads = 11
num_threads = 12 computed daxpy in 0.003612241999090 seconds threads = 12
● oscarmendez@mendezos:~/Documents/Taller OpenMP$
```

Como se puede observar en las capturas de pantalla anteriores, el tiempo de ejecución en la mayoría de los casos va en descenso una vez se llega a los 10 threads, sin embargo, luego de este valor, este tiende a crecer, esto puede deberse a que no todo el programa es compatible para una paralelización con esta cantidad

de hilos y, además, que esto puede deberse a la sobrecarga de la concurrencia y a la competencia por recursos que los hilos agregan.

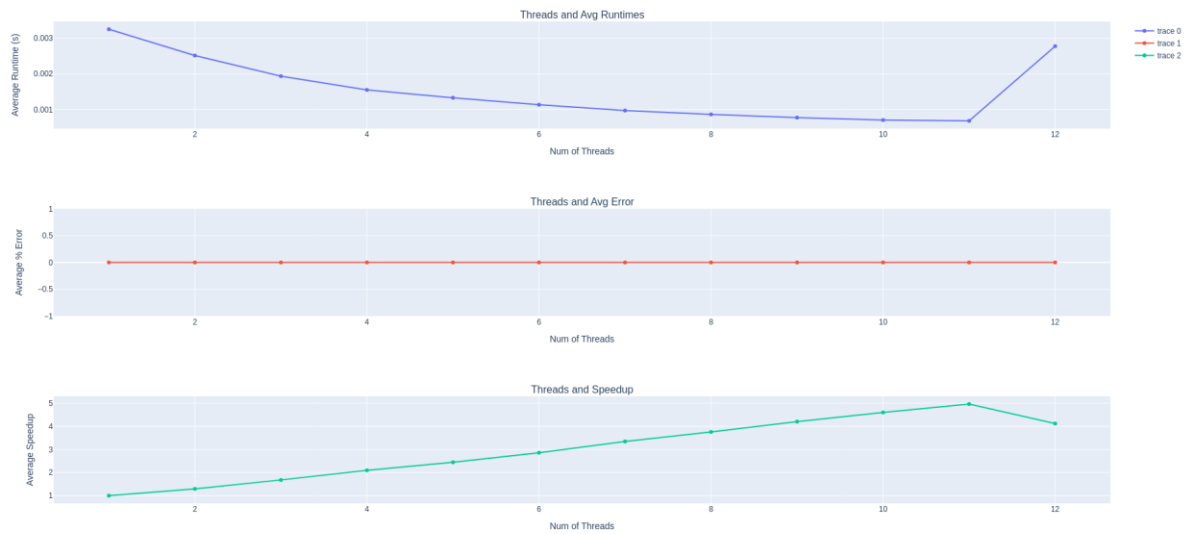
c- Un programa/script que permita graficar los resultados después de ejecutar el programa implementado en el punto a y b, de forma que se puede visualizar tiempo de ejecución, speedup, y porcentaje error (para el caso a).

Daxpy_loop.c



Inicialmente, en cuanto a la ejecución del DAXPY 100 veces, se puede observar que el comportamiento de este en cuanto a tiempo de ejecución es bastante constante, pero luego de llegar a un umbral este tiempo se dispara, esto se puede deber a como bien se mencionó en el punto anterior que existe mucha competencia por los recursos y que además la administración de hilos, la sincronización y la comunicación entre ellos, puede verse afectada a medida en que se agregan más hilos (dependiendo del programa), haciendo que la ejecución sea más lenta. Por su parte, en cuanto al speedup se puede observar cómo el mismo va en crecimiento hasta que llega a un umbral y esto puede ser fundamentado debido a la Ley de Amdahl donde el speedup máximo se puede lograr al paralelizar un programa, pero este está limitado por la parte del programa que no se puede ejecutar en paralelo. A medida que se agregan más hilos para paralelizar una task, la fracción no paralelizable se convierte en un cuello de botella, limitando el aumento del speedup.

Taylor_loop.c (Euler)



Por su parte, en la ejecución del código del cálculo de Euler se puede observar cómo al igual que con el DAXPY, el tiempo de ejecución en principio viene decreciendo y una vez llegado a un umbral, este tiempo se dispara, esto se puede deber a como bien se mencionó, a las dificultades de sincronización de los hilos y de además de las competencias que tienen estos por los recursos. Por su parte, se puede observar como el speedup tiene un comportamiento creciente, pero una vez llegado a un umbral este ya no crece más, esto puede ser producto de que el código no es completamente paralelizable para correr en un entorno con 12 hilos. Por su parte, con respecto al error obtenido, si se observa el mismo es muy cercano a cero, lo que puede ser considerado no tan crítico dependiendo la aplicación que se le dé al código, en este caso el error empieza a tener variaciones en el orden de los +15 decimales, siendo este 0.0000000000000016 por lo cual, puede ser no considerado un valor muy grande.