

Método Jacobi de manera secuencial, con hilos y procesos

Presentado por:

Brandon David Palacio Alvarez
Johan Esteban Restrepo Ladino

Presentado a:

Ramiro Andrés Barrios Valencia

Programa de Ingeniería en Sistemas y Computación
HPC (High Performance Computing)
Universidad Tecnológica de Pereira
agosto 2022

Tabla de contenidos

Tabla de contenidos.....	2
Resumen	3
Introducción.....	4
Marco conceptual.....	5
Métodos numéricos.....	5
Método de Jacobi	5
Algoritmo método Jacobi.....	6
HPC (High Performance Computing)	7
¿Por qué es tan importante la HPC?.....	7
¿Cómo funciona la HPC?.....	7
Procesamiento Paralelo.....	7
¿Cómo funciona el procesamiento paralelo?.....	7
Hilos (Threads)	8
CPU Clock Time	8
Speedup	8
Marco contextual.....	9
Pruebas realizadas	9
Características de la máquina.....	9
Desarrollo.....	9
Análisis de los datos – Tablas.....	13
Análisis de datos – Gráficas	19
Observaciones:	21
Conclusiones	22
Bibliografía	23

Resumen

En el siguiente informe, se observará la implementación y comportamiento del método Jacobi, realizado en el lenguaje de programación C, de forma secuencial y de forma paralela con implementaciones de hilos y procesos. Esto nos mostrará y determinará con cuál de las opciones ya mencionadas el algoritmo se desempeña con un mejor rendimiento. Los datos serán tomados a través de fórmulas y tablas en Excel, las cuales serán tabuladas y promediadas, esto con el fin de poder sacar datos más exactos en su medición además de la realización de sus respectivas gráficas comparativas con los diferentes tipos de ejecución. El proceso debe ser realizado repetidas veces en diferentes iteraciones para obtener tiempos más aproximados a su tiempo de ejecución real.

Introducción

Los sistemas de ecuaciones lineales no solamente se pueden resolver analíticamente, también se pueden recurrir a los métodos numéricos. Si estamos hablando de sistemas de ecuaciones lineales tenemos lo que es el método Jacobi, el cual, a través de una pequeña definición, utiliza las aproximaciones sucesivas (o en otras palabras iteraciones) para encontrar los valores aproximados de un sistema lineal propuesto. Ahora, si vemos esto desde el punto de vista tecnológico y autónomo, esto dicho al ser iteraciones y algo matemático se puede llevar a la programación, lo cual nos simplificaría mucho la cantidad estimada de tiempo cuando se tienen pequeñas iteraciones.

Si pensamos en un sistema lineal con muchas iteraciones, ya se complica un poco, esto debido a que todas las maquinas no están diseñadas para correr tiempos de ejecución muy grandes o incluso esto podría llegar a topar la mayoría de su memoria RAM o incluso parar el procesador por la cantidad de procesos ejecutándose simultáneamente.

Gracias a las investigaciones del HPC, se puede determinar el alcance de una máquina, que tan eficaz es, y cuando soporta en cuanto a su velocidad y sus características generales.

En el Reto de Jacobi, vamos a hacer un breve análisis de cómo llevamos matemáticamente el método numérico de Jacobi a la programación en C, y a su vez, se determinará que tan eficiente es en la máquina que se usará para el experimento.

Marco conceptual

Métodos numéricos

Los métodos numéricos corresponden a aquellos procedimientos en los que se realizan cálculos aritméticos que permitan dar solución a problemas. Existen diversos tipos de métodos numéricos o algoritmos que pueden ser utilizados para dar una solución aproximada a problemas planteados que no pueden ser resueltos de otra manera, sin embargo, cabe resaltar que se pueden presentar errores.

Método de Jacobi

Es un método iterativo, usado para resolver sistemas de ecuaciones lineales del tipo. El algoritmo toma su nombre del matemático alemán Carl Gustav Jakob Jacobi. El método de Jacobi consiste en usar fórmulas como iteración de punto fijo.

Un método iterativo con el cual se resuelve el sistema lineal $Ax = b$ comienza con una aproximación inicial $x(0)$ a la solución x y genera una sucesión de vectores $x(k)$ que converge a x . Los métodos iterativos traen consigo un proceso que convierte el sistema $Ax = b$ en otro equivalente de la forma $x = Tx + c$ para alguna matriz fija T y un vector c .

Luego de seleccionar el vector inicial $x(0)$ la sucesión de los vectores de la solución aproximada se genera calculando:

$$X(k) = Tx(k-1) + c$$

Para cada $k = 1, 2, 3, \dots$

El método se escribe en la forma $x(k) = T x(k-1) + c$ separando A en sus partes diagonal D y fuera de la diagonal. Sea D la matriz diagonal cuya diagonal es la misma que A , sea $-L$ la parte estrictamente triangular inferior de la parte A y sea $-U$ la parte estrictamente triangular superior de A .

Con esta notación $A = D - L - U$, entonces transformamos la ecuación $Ax = b$, o $(D - L - U)x = b$, en

$$Dx = (L + U)x + b$$

Y, si D^{-1} existe, es decir, si a_{ii} es distinto de cero para cada i , entonces

$$x = D^{-1}(L + U)x + D^{-1}b$$

Esto da origen a la forma matricial del método iterativo de Jacobi:

$$X(k) = D^{-1}(L + U)x(k-1) + D^{-1}b, k = 1, 2, \dots$$

Al introducir la notación $T_j = D^{-1}(L + U)$ y c esta técnica tiene la forma:

$$X(k) = T x(k-1) + c$$

Es de mencionar el siguiente teorema: "Si A es estrictamente diagonal dominante, entonces con cualquier elección de la aproximación inicial, el método de Jacobi da una sucesión que converge a la solución única de $Ax = b$ "

La sucesión se construye descomponiendo la matriz del sistema A en la forma siguiente:

$$A = D + L + U$$

Donde, D es una matriz diagonal.

Partiendo de $Ax=b$, podemos reescribir dicha ecuación como:

$$Dx(L+U)x=b$$

Luego

$$x = D^{-1} [b - (L + U) x]$$

Donde k es el contador de iteración, finalmente tenemos:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), i = 1, 2, 3, \dots$$

Algoritmo método Jacobi

Función Jacobi ($A * X^0$)

" X^0 es una aproximación inicial a la solución //

Para $K \leftarrow 1$ hasta convergencia hacer

Para $i \leftarrow 1$ hasta n hacer

$O \leftarrow 0$

Para $j \leftarrow 1$ hasta n hacer

¡Si $J! = i$ entonces|

$O = o + a_{ij} x_j$

Fin para

$$x_i = \frac{(b_i - o)}{A_{ii}}$$

HPC (High Performance Computing)

La informática de alto rendimiento (HPC) es un tipo de aplicaciones y cargas de trabajo que realizan operaciones intensivas desde el punto de vista informático. La demanda de HPC para impulsar casos de uso ricos en datos y habilitados por la IA en entornos académicos e industriales es cada vez mayor. Los clústeres HPC se diseñan sobre procesadores de alto rendimiento con memoria y almacenamiento de alta velocidad, además de otros componentes avanzados.

¿Por qué es tan importante la HPC?

En los últimos años, la cantidad de datos ha proliferado rápidamente y muchas aplicaciones nuevas han podido aprovechar la potencia de la HPC, es decir, de la capacidad para realizar operaciones de proceso intensivo con los distintos recursos compartidos, para lograr resultados en menos tiempo y a un menor coste en comparación con la informática tradicional. Al mismo tiempo, el hardware y el software HPC se han vuelto más asequibles y están más extendidos.

¿Cómo funciona la HPC?

Mientras que la HPC puede ejecutarse en un único nodo, su verdadera potencia proviene de la conexión de varios nodos HPC en un clúster o superordenador con capacidad de procesamiento paralelo de los datos. Los clústeres HPC pueden procesar simulaciones a escala extrema, inferencias de IA y análisis de datos que podrían no ser factibles en un sistema único.

Procesamiento Paralelo

El procesamiento paralelo es un método de computación para ejecutar dos o más procesadores (CPU) para manejar partes separadas de una tarea general. Dividir diferentes partes de una tarea entre varios procesadores ayudará a reducir la cantidad de tiempo para ejecutar un programa. Cualquier sistema que tenga más de una CPU puede realizar procesamiento paralelo, así como procesadores de múltiples núcleos que se encuentran comúnmente en las computadoras hoy en día.

Los procesadores multinúcleo son chips IC que contienen dos o más procesadores para un mejor rendimiento, menor consumo de energía y procesamiento más eficiente de múltiples tareas. Estas configuraciones de múltiples núcleos son similares a tener varios procesadores separados instalados en la misma computadora. La mayoría de las computadoras pueden tener entre dos y cuatro núcleos; aumentando hasta 12 núcleos.

¿Cómo funciona el procesamiento paralelo?

Por lo general, un informático dividirá una tarea compleja en varias partes con una herramienta de software y asignará cada parte a un procesador, luego cada procesador resolverá su parte y una herramienta de software volverá a ensamblar los datos para leer la solución o ejecutar la tarea.

Hilos (Threads)

Un Hilo es simplemente una tarea que puede ser ejecutada al mismo tiempo con otra tarea. En los sistemas operativos tradicionales cada proceso tiene un espacio de direcciones y un hilo de control. Un hilo es una secuencia de código en ejecución dentro del contexto de un proceso no pueden ejecutarse ellos solos, requieren la supervisión de un proceso padre para correr.

Cuando se agrupan varios hilos son denominados Multihilos, los cuales pueden ser manipularlos todos de una vez, es una característica que permite a una aplicación realizar varias tareas a la vez.

CPU Clock Time

La CPU Clock time, o velocidad del reloj de la CPU, en general, una velocidad de reloj más alta significa una CPU más rápida. Sin embargo, hay muchos otros factores que desempeñan un rol. La CPU procesa muchas instrucciones (cálculos de bajo nivel como los aritméticos) a partir de diferentes programas a cada segundo. La velocidad de reloj mide la cantidad de ciclos que ejecuta tu CPU por segundo, medida en GHz (gigahertz). Un “ciclo” es técnicamente un pulso sincronizado por un oscilador interno, pero, para nuestros fines, es una unidad básica que ayuda a comprender la velocidad de una CPU. Durante cada ciclo, miles de millones de transistores dentro del procesador se abren y cierran. La frecuencia es más operaciones dentro de una cantidad dada de tiempo, como se representa más arriba. Una CPU con una velocidad de reloj de 3,2 GHz ejecuta 3200 millones de ciclos por segundo. (Las CPU más antiguas tenían velocidades medidas en megahertz, o millones de ciclos por segundo.) A veces, se completan múltiples instrucciones en un solo ciclo de reloj. En otros casos, una instrucción podría ser manejada a lo largo de múltiples ciclos de reloj. Dado que los distintos diseños de CPU manejan las instrucciones de manera diferente, lo mejor es comparar las velocidades de reloj dentro de la misma marca y generación de CPU.

Speedup

Es un proceso realizado para mejorar el rendimiento de un sistema que procesa un problema determinado. Más técnicamente, es la mejora en la velocidad de ejecución de una tarea ejecutada en dos arquitecturas similares con diferentes recursos.

Ejemplos:

Usando tiempos de ejecución: Estamos probando la efectividad de un predictor de saltos en la ejecución de un programa. Primero, ejecutamos el programa con el predictor de saltos estándar en el procesador, lo que consigue un tiempo de ejecución de 2,25 segundos. A continuación, ejecutamos el programa con nuestro predictor de saltos modificado (y esperamos mejorado) en el mismo procesador, lo que produce un tiempo de ejecución de 1,50 segundos. En ambos casos la carga de ejecución es la misma. Usando nuestra fórmula de speedup, sabemos que:

$$S_{\text{latencia}} = \frac{L_{\text{antigua}}}{L_{\text{nueva}}} = \frac{2,25 \text{ s}}{1,50 \text{ s}} = 1,5.$$

Nuestro nuevo predictor de saltos ha conseguido una speedup de 1,5x sobre el original.

Marco contextual

Pruebas realizadas

Se realizaron las respectivas pruebas del algoritmo del método Jacobi con los respectivos valores, primeramente, ejecutándolo de forma secuencial. Así mismo para su forma paralela con Hilos y su forma paralela con procesos. Se analizaron los diferentes métodos utilizados, además de tomar nota de sus respectivos tiempos y su evolución al aplicarle el SpeedUp. A continuación, se verá un poco todos los datos que se mostraron en este análisis.

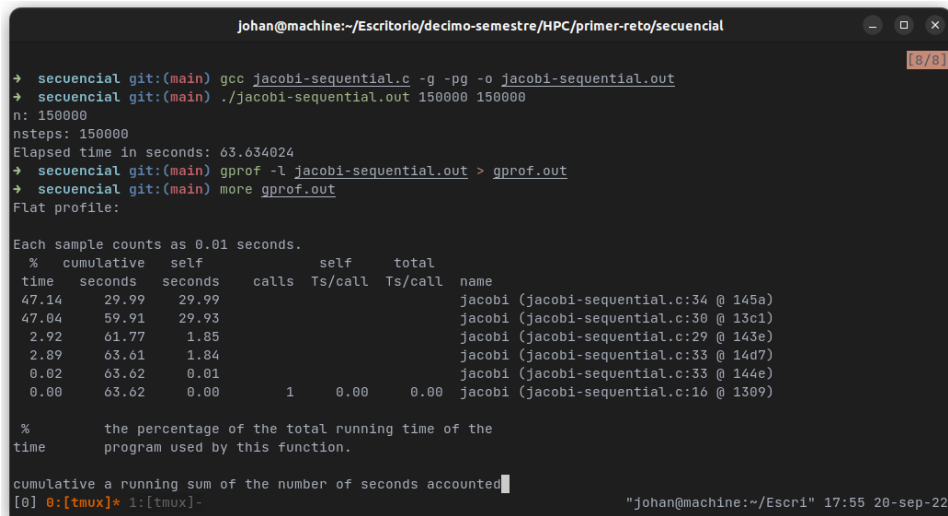
Características de la máquina

Las pruebas fueron realizadas en un equipo portátil con las siguientes características:

- OS: Ubuntu 22.04 jammy
- Kernel: x86_64 Linux 5.15.0-47-generic
- Resolution: 2726x768
- CPU: Intel Core i5 – 1035G4 @ 8x 3.7 GHz
- GPU: Intel Corporation Iris Plus Graphics G4 (Ice Lake)
- RAM: 3719 MiB

Desarrollo

Para el desarrollo del presente reto, como primer paso se realizó el análisis de la implementación secuencial del algoritmo de Jacobi que fue entregada por el profesor utilizando gprof como herramienta de análisis para aplicaciones en sistemas operativos basados en Unix, obteniendo los siguientes resultados:



```
Johan@machine:~/Escritorio/decimo-semester/HPC/primer-reto/secuencial
+ secuencial git:(main) gcc jacobi-sequential.c -g -pg -o jacobi-sequential.out
+ secuencial git:(main) ./jacobi-sequential.out 150000 150000
n: 150000
nsteps: 150000
Elapsed time in seconds: 63.634024
+ secuencial git:(main) gprof -l jacobi-sequential.out > gprof.out
+ secuencial git:(main) more gprof.out
Flat profile:
Each sample counts as 0.01 seconds.
%   cumulative   self           calls Ts/call   Ts/call  name
time  seconds    seconds                   Ts/call   Ts/call  name
47.14   29.99     29.99                1 0.00    0.00 jacobi (jacobi-sequential.c:34 @ 145a)
47.04   59.91     29.93                1 0.00    0.00 jacobi (jacobi-sequential.c:30 @ 13c1)
2.92   61.77      1.85                1 0.00    0.00 jacobi (jacobi-sequential.c:29 @ 143e)
2.89   63.61      1.84                1 0.00    0.00 jacobi (jacobi-sequential.c:33 @ 14d7)
0.02   63.62      0.01                1 0.00    0.00 jacobi (jacobi-sequential.c:33 @ 144e)
0.00   63.62      0.00                1 0.00    0.00 jacobi (jacobi-sequential.c:16 @ 1309)

%           the percentage of the total running time of the
time        program used by this function.
cumulative a running sum of the number of seconds accounted
[0] 0:[tmux]* 1:[tmux]- "Johan@machine:~/Escr1" 17:55 20-sep-22
```

```

Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self           self      total
time  seconds    seconds   calls   Ts/call  Ts/call  name
47.74    29.53    29.53           1      0.00    0.00  jacobi (jacobi-sequential.c:34 @ 145a)
46.36    58.20    28.68           1      0.00    0.00  jacobi (jacobi-sequential.c:30 @ 13c1)
 3.03    60.08     1.88           1      0.00    0.00  jacobi (jacobi-sequential.c:29 @ 143e)
 2.85    61.84     1.76           1      0.00    0.00  jacobi (jacobi-sequential.c:33 @ 14d7)
 0.02    61.85     0.01           1      0.00    0.00  jacobi (jacobi-sequential.c:33 @ 144e)
 0.00    61.85     0.00           1      0.00    0.00  jacobi (jacobi-sequential.c:16 @ 1309)

%           the percentage of the total running time of the
time        program used by this function.

cumulative  a running sum of the number of seconds accounted
seconds     for by this function and those listed above it.

self        the number of seconds accounted for by this
seconds     function alone. This is the major sort for this
            listing.

calls       the number of times this function was invoked, if
            this function is profiled, else blank.

self        the average number of milliseconds spent in this
ms/call     function per call, if this function is profiled,
            else blank.

total       the average number of milliseconds spent in this
ms/call     function and its descendents per call, if this
            function is profiled, else blank.

name        the name of the function. This is the minor sort
            for this listing. The index shows the location of
            the function in the gprof listing. If the index is
            in parenthesis it shows where it would appear in
            the gprof listing if it were to be printed.

```

El anterior resultado evidenció que la parte del código que se debía optimizar para obtener un mejor rendimiento era la función con firma `jacobi(int nsweeps, int n, double *u, double *f)`. Cabe resaltar que al código que fue entregado por el profesor se le hicieron pequeñas modificaciones en materia de la toma de tiempo, ya que el código original usaba un archivo extra el cual se tenía que compilar junto que el código fuente del algoritmo en cuestión, mientras que en la presente implementación se hizo uso de la función `gettimeofday` que ofrece la librería de `c sys/time`. Finalmente, en la implementación secuencial, se compiló el código fuente y se ejecutó un script para correr el programa 10 veces para tamaños de arreglos de 5000 25000 50000 100000 150000 200000 250000 300000 y número de pasos igual al tamaño del arreglo, guardando los respectivos resultados en un archivo separado por coma (.csv) para su posterior análisis.

Para la implementación del algoritmo utilizando hilos, se empleó la librería `pthread` que incluye las funciones necesarias para la creación y funcionamiento de hilos con el estándar Posix. Teniendo como premisa que la función que se debía atacar era la función `jacobi`, se separó la lógica principal de la función `jacobi` que consistía en calcular los valores para los arreglos `utmp` y `u` en ciclos `for` separados y se movió dicha lógica a una función de subrutina llamada `thread_subroutine` que sería posteriormente ejecutada por cada uno de los hilos. Del mismo modo, se creó una estructura llamada `thread_info` que contenía los campos: `from`, `to`, `h2`. Lo anterior con el fin de pasar la información necesaria para el correcto funcionamiento de cada uno de los hilos. La función `jacobi` fue responsable de la creación de hilos y la

creación de la información de los hilos que consistió en calcular que partes de cada arreglo le correspondía a cada hilo, para ello se emplearon las siguientes fórmulas para la distribución de carga de los hilos:

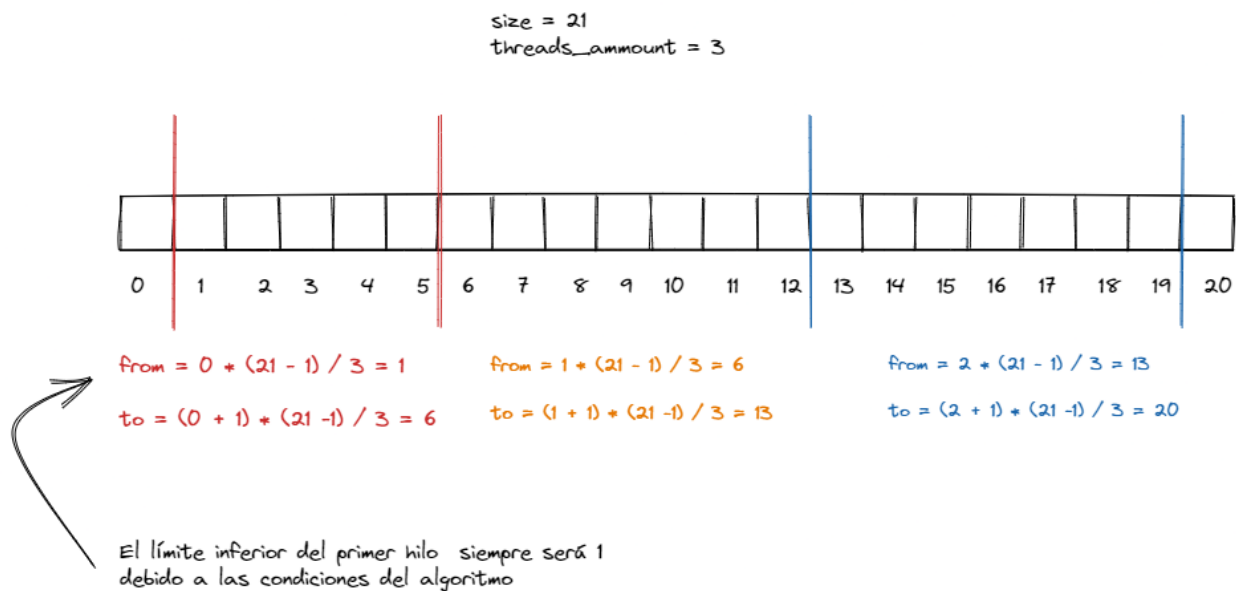
$$from = \frac{thread_idx * (size - 1)}{threads_ammount}$$

$$to = \frac{(thread_idx + 1) * (size - 1)}{threads_ammount}$$

Donde:

- **from**: límite inferior de trabajo del hilo (Parte entera de la división).
- **to**: límite superior de trabajo del hilo no inclusivo (Parte entera de la división).
- **thread_idx**: índice del hilo al que se le está calculando la carga. Esto debido a que la carga se calcula dentro de un ciclo for.
- **threads_ammount**: cantidad de hilos entre los que se va a distribuir la carga.
- **size**: tamaño del arreglo en el que trabajan los hilos.

Dichas formulas garantizan la correcta distribución de carga para cada uno de los hilos, independientemente de si el arreglo tiene un tamaño par o impar. Teniendo en cuenta que el algoritmo siempre trabaja desde la posición 1 del arreglo u se hace la asignación especifica de ese límite de trabajo para el primer hilo que corresponde al hilo con thread_idx = 0. Por otra parte, también se tuvo en cuenta que los limites superiores “to” son no inclusivos.



Una vez establecidos los límites, se construyeron los hilos y se les pasó la información correspondiente para trabajar con los arreglos globales u, utmp y f, cómo también la referencia a la subrutina thread_subroutine que ejecuta cada uno de los hilos. Posteriormente, solo fue cuestión de esperar a que todos los hilos terminaran su ejecución, acción que se realizó en la función jacobi. Del mismo modo que en la implementación secuencial, se compiló el código fuente y se ejecutó un script para correr el

programa 10 veces para tamaños de arreglos de 5000 25000 50000 100000 150000 200000 250000 300000 y número de pasos igual al tamaño del arreglo, guardando los respectivos resultados en un archivo separado por coma (.csv) para su posterior análisis.

Finalmente, para la implementación de procesos, se siguió en el mismo enfoque que se utilizó para la implementación de hilos. Sin embargo, al tratarse de una implementación con procesos, los arreglos con los que trabaja el algoritmo fueron creados utilizando la función mmap de c la cual genera una especie de “buffer” que puede ser accedido por todos los procesos que desprendan del mismo proceso padre que utilizó la función mmap. Dicha función también implica que después de que se hayan utilizados los arreglos, se libere la memoria utilizando la función munmap. Del mismo modo que en la implementación secuencial y con hilos, se compiló el código fuente y se ejecutó un script para correr el programa 10 veces para tamaños de arreglos de 5000 25000 50000 100000 150000 200000 250000 300000 y número de pasos igual al tamaño del arreglo, guardando los respectivos resultados en un archivo separado por coma (.csv) para su posterior análisis.

Análisis de los datos – Tablas

1. Resultados al ejecutar el algoritmo de forma secuencial, utilizando uno de sus núcleos:

Tabla 1. Forma secuencial

SECUENCIAL								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,070518	1,660448	6,839659	27,427966	60,952717	111,354166	168,269115	244,131413
2	0,070567	1,772878	6,613491	26,972229	59,457476	107,83176	166,787743	244,865804
3	0,06901	1,657812	6,612312	26,596308	61,974235	105,901168	168,013353	243,478388
4	0,069134	1,726419	6,594557	26,569873	64,268995	108,455123	166,140947	251,135133
5	0,069193	2,021203	7,094523	27,738105	68,217536	105,26737	171,356909	304,228165
6	0,068983	1,702576	6,634458	27,779457	59,116138	105,780392	167,135808	243,512121
7	0,067923	1,694974	6,866084	27,454497	59,258784	105,609437	168,207896	245,448065
8	0,069504	1,665885	6,676619	26,55546	58,649381	105,489293	168,14339	242,925408
9	0,06908	1,832738	6,591834	26,441471	60,418955	105,653748	168,049035	244,311493
10	0,069766	1,747546	6,602481	26,366694	60,099327	105,686957	166,936086	244,13406
Promedio	0,0693678	1,7482479	6,7126018	26,990206	61,2413544	106,702941	167,904028	250,817005

2. Resultados al ejecutar el algoritmo de forma paralela usando 2 hilos con su respectivo speedup, el cual muestra un incremento o un decremento en su tiempo de ejecución:

Tabla 2. Forma paralela usando 2 hilos

PARALELO 2 HILOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,119735	1,259821	4,415376	17,54734	43,715109	73,15996	116,594983	170,146736
2	0,101631	1,264117	4,441099	17,525298	53,319305	73,520866	117,636898	168,895415
3	0,101825	1,281504	4,409558	18,075434	53,058183	72,585044	118,48107	169,791823
4	0,101941	1,387073	4,396124	17,591784	50,80057	74,074386	116,827505	172,702174
5	0,099289	1,265476	4,415396	17,453549	46,781226	72,679665	116,421567	169,482367
6	0,100917	1,266078	4,407603	17,417208	40,67566	73,483077	116,791715	171,0188
7	0,099598	1,258913	4,412599	18,088009	41,011657	74,085809	118,381051	173,379146
8	0,101636	1,263201	4,40568	17,612186	40,863117	73,778389	114,807564	168,867852
9	0,099522	1,260778	4,487693	18,414875	40,539884	73,012902	115,112549	169,799697
10	0,100549	1,254228	4,405754	17,498659	42,05035	74,379608	114,699593	171,35174
Promedio	0,1026643	1,2761189	4,4196882	17,7224342	45,2815061	73,4759706	116,57545	170,543575
Speedup	0,67567597	1,36997258	1,51879533	1,52294012	1,35245842	1,45221547	1,4403035	1,47069161

3. Resultados al ejecutar el algoritmo de forma paralela usando 4 hilos:

Tabla 3. Forma paralela usando 4 hilos

PARALELO 4 HILOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,133087	1,100565	4,175299	13,562245	28,672452	50,225952	88,233468	128,589024
2	0,126218	1,106102	4,10664	13,293863	28,552299	49,955059	81,591917	130,717745
3	0,123303	1,08299	3,918477	13,457374	28,269495	50,026879	79,289199	133,675982
4	0,124966	1,149229	4,020273	14,267855	28,508449	49,95851	79,807078	132,02043
5	0,121785	1,26055	3,80241	13,421937	28,493673	49,841577	83,596178	130,288916
6	0,12336	1,207781	4,213437	13,33614	28,908786	50,519741	80,729658	132,777193
7	0,127778	1,186311	3,535538	13,845864	28,779008	50,107204	80,688223	135,488274
8	0,122783	1,434789	3,566448	13,633521	28,55339	50,117916	79,602395	163,048274
9	0,122787	1,559004	4,190458	14,139242	31,517995	50,214398	79,667105	134,206828
10	0,125395	1,372214	4,462098	13,521416	28,513781	49,936871	80,21096	134,80298
Promedio	0,1251462	1,2459535	3,9991078	13,6479457	28,8769328	50,0904107	81,3416181	135,561565
Speedup	0,5542941	1,40314057	1,67852484	1,97760209	2,12077075	2,13020696	2,06418353	1,85020736

4. Resultados al ejecutar el algoritmo de forma paralela usando 8 hilos:

Tabla 4. Forma paralela usando 8 hilos

PARALELO 8 HILOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,256055	1,642135	4,27573	14,672464	32,830089	55,741678	88,689531	168,791663
2	0,250251	2,054815	4,322309	15,911861	33,191703	56,097162	89,721601	165,913439
3	0,247097	1,74761	4,197255	17,165038	32,66311	55,841917	89,086697	170,615588
4	0,246673	1,640348	4,181418	15,912776	32,642154	55,823789	89,636318	166,104906
5	0,251477	1,602558	4,317085	15,626595	32,201501	55,375414	89,072876	165,636696
6	0,25455	1,602078	4,374915	15,622987	32,529336	55,256024	88,348265	169,789595
7	0,247028	1,569359	4,792966	16,18046	32,048638	55,659164	88,542753	169,723453
8	0,250506	1,627511	5,119356	16,179709	32,091583	55,87152	89,918904	172,567026
9	0,255272	1,606031	5,677363	16,055077	32,137376	56,481203	88,542497	170,820352
10	0,250919	1,63767	5,179406	16,193817	32,028454	55,653034	88,470025	168,320858
Promedio	0,2509828	1,6730115	4,6437803	15,9520784	32,4363944	55,7800905	89,0029467	168,828358
Speedup	0,27638468	1,04497064	1,44550374	1,69195545	1,88804445	1,91292163	1,88649965	1,48563315

5. Resultados al ejecutar el algoritmo de forma paralela usando 16 hilos:

Tabla 5. Forma paralela usando 16 hilos

PARALELO 16 HILOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,496062	2,911952	7,458961	21,338266	50,293054	112,681542	211,433162	299,674736
2	0,505106	3,280597	7,361768	25,151644	75,780654	114,288423	212,585186	297,399763
3	0,499378	3,030872	8,050331	26,286325	78,543859	111,326398	214,031714	271,451668
4	0,495739	3,004964	8,396143	19,172675	35,471472	107,996096	203,280707	263,114193
5	0,486892	2,79641	8,67164	19,874363	63,144934	108,373706	211,057537	300,739298
6	0,504575	2,756444	7,885073	19,557564	72,108168	108,220491	169,06387	241,746888
7	0,495502	2,865569	7,298788	19,363201	35,633135	105,840035	211,463021	230,219986
8	0,576115	3,101082	7,257934	19,086387	67,976954	112,470574	213,340872	296,227776
9	0,643477	2,6851	7,255163	19,356378	68,988064	110,591417	170,223684	298,785756
10	0,498264	2,978504	7,252166	19,143696	48,848122	105,081137	211,386989	234,492853
Promedio	0,520111	2,9411494	7,6887967	20,8330499	59,6788416	109,686982	202,786674	273,385292
Speedup	0,13337115	0,59440976	0,87303671	1,29554751	1,02618202	0,97279494	0,82798354	0,91744879

6. Resultados al ejecutar el algoritmo de forma paralela usando 32 hilos:

Tabla 6. Forma paralela usando 32 hilos

PARALELO 32 HILOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	1,019096	6,375862	14,652305	34,573072	50,871404	79,85534	123,035471	180,098622
2	1,01332	6,462651	15,159297	29,940346	50,906718	80,860218	123,428694	180,317037
3	1,018812	6,480895	16,985545	29,800508	51,044403	80,64476	122,821963	180,469769
4	1,009772	6,354384	15,502787	29,68339	51,338215	79,664754	123,12328	180,728692
5	1,014908	6,466584	14,384547	29,804785	50,994251	79,463348	140,112086	180,319101
6	1,019166	6,248705	15,93608	29,766263	51,721061	79,571253	122,785001	179,052858
7	1,00118	6,638062	17,060301	29,769612	51,137875	79,689786	122,450686	180,154761
8	1,010303	6,772912	15,923848	29,920892	51,96867	79,693258	123,331458	180,014394
9	1,012062	6,06122	15,667135	29,793829	51,165347	79,899387	122,805298	180,187174
10	1,015857	6,194555	16,139094	29,781961	51,238633	79,543697	123,260141	202,335462
Promedio	1,0134476	6,405583	15,7410939	30,2834658	51,2386577	79,8885801	124,715408	182,367787
Speedup	0,06844735	0,27292565	0,42643808	0,89125222	1,19521777	1,33564699	1,34629739	1,37533612

7. Resultados al ejecutar el algoritmo de forma paralela usando 2 procesos con su respectivo speedup en las siguientes tablas:

Tabla 7. Forma paralela usando 2 procesos

PARALELO 2 PROCESOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,332976	2,656792	8,163897	27,705125	60,227934	103,545187	167,542264	246,900896
2	0,327405	2,679908	8,23258	28,496163	60,874102	105,422741	166,925184	244,87278
3	0,358002	2,768842	8,238476	28,238057	59,614715	104,451328	167,441784	244,203044
4	0,321534	2,774694	8,221472	28,286827	60,402905	103,454005	167,423422	242,487356
5	0,33158	2,680733	8,802743	28,763067	63,779103	105,990693	166,872865	245,938318
6	0,32219	2,703684	8,618326	27,984385	59,359928	104,484665	168,413789	281,664435
7	0,32928	2,69084	8,251016	27,835981	59,432827	104,65069	166,356165	248,045792
8	0,331968	2,768856	8,063935	27,73788	59,930805	103,909621	168,514889	246,381347
9	0,320378	2,773385	8,183051	28,259206	59,258719	119,228632	166,982602	245,130342
10	0,325865	2,695884	8,048853	28,004801	60,160165	103,832463	166,633596	241,864117
Promedio	0,3301178	2,7193618	8,2824349	28,1311492	60,3041203	105,897003	167,310656	248,748843
Speedup	0,21013044	0,64288904	0,81046237	0,959442	1,01554179	1,00761059	1,00354653	1,00831426

8. Resultados al ejecutar el algoritmo de forma paralela usando 4 procesos:

Tabla 8. Forma paralela usando 4 procesos

PARALELO 4 PROCESOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,447156	2,792339	6,90911	21,78319	47,194325	81,315522	136,372205	209,438812
2	0,442894	2,845681	7,126978	21,738015	46,969885	80,692426	136,940758	208,669745
3	0,438621	2,821014	6,97933	21,826312	47,848298	81,608975	134,704044	204,98651
4	0,443682	2,817379	7,008647	21,97567	47,01344	83,108457	136,124304	226,109722
5	0,443528	2,789326	6,987693	21,722109	47,462359	91,253147	159,367995	204,161779
6	0,439878	2,819521	7,237765	22,304743	47,021469	123,882927	135,520765	206,524306
7	0,433254	2,852609	7,696989	23,837239	47,49789	114,494243	131,634453	209,117266
8	0,430526	2,781579	7,619619	23,602504	48,040871	87,943922	136,020112	203,408075
9	0,44954	2,828173	7,507467	22,635917	48,112212	82,846385	132,807553	204,810314
10	0,442461	2,785239	7,544872	22,775363	47,02871	82,933767	137,346079	204,889327
Promedio	0,441154	2,813286	7,261847	22,4201062	47,4189459	91,0079771	137,683827	208,211586
Speedup	0,15724169	0,62142559	0,92436563	1,20383935	1,29149548	1,17245702	1,21948984	1,20462559

9. Resultados al ejecutar el algoritmo de forma paralela usando 8 procesos:

Tabla 9. Forma paralela usando 8 procesos

PARALELO 8 PROCESOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	0,735613	4,148634	8,904632	29,90269	55,732089	96,639557	164,730131	260,016218
2	0,743945	4,053419	10,113949	29,14624	55,246532	96,263379	167,198787	256,633621
3	0,718342	4,112523	10,275567	29,408021	70,582668	97,293718	166,096055	260,743275
4	0,7087	4,032773	10,276978	28,712287	72,673982	97,39292	165,182133	257,741952
5	0,705447	4,193134	10,927193	29,059779	56,263077	98,755433	164,616389	257,060941
6	0,74748	4,06013	11,183457	29,221747	55,656484	100,054135	167,205689	259,346199
7	0,710437	4,247082	10,947451	29,468696	55,668934	95,930097	165,045221	256,341979
8	0,717664	3,974424	11,249954	28,760468	56,699053	97,273491	165,96936	279,51475
9	0,715464	4,064923	11,101714	29,596129	56,292176	96,640252	168,403471	258,816589
10	0,718619	4,02727	10,824219	29,001689	55,945421	97,341344	165,40305	259,75644
Promedio	0,7221711	4,0914312	10,5805114	29,2277746	59,0760416	97,3584326	165,985029	260,597196
Speedup	0,09605452	0,42729495	0,63443075	0,92344376	1,03665298	1,09598048	1,01156128	0,96247008

10. Resultados al ejecutar el algoritmo de forma paralela usando 16 procesos:

Tabla 10. Forma paralela usando 16 procesos

PARALELO 16 PROCESOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	1,216872	6,927194	16,264662	41,510589	75,144455	122,76185	194,399454	306,60221
2	1,217757	6,871702	16,312386	41,498901	72,408039	120,127066	199,364512	307,594529
3	1,228165	6,95713	16,126786	41,60138	73,907716	119,902571	199,626781	308,30104
4	1,231149	7,223437	16,551894	41,867819	73,700774	119,516441	205,589162	307,822704
5	1,274348	6,885648	16,461757	41,436362	73,875166	120,316497	197,812518	304,283539
6	1,284923	6,879758	16,751522	41,948654	72,667034	121,658927	201,437146	305,26745
7	1,256509	6,935293	16,529059	41,508324	73,951771	121,199742	194,157553	308,0539
8	1,240689	7,067801	16,292067	43,928084	74,84874	121,694996	203,825203	305,39972
9	1,265831	7,770791	16,853482	41,918389	73,239659	122,067568	201,285379	308,017068
10	1,263511	7,538589	17,540304	41,611182	72,940953	120,57841	200,041616	309,108795
Promedio	1,2479754	7,1057343	16,5683919	41,8829684	73,6684307	120,982407	199,753932	307,045096
Speedup	0,05558427	0,24603339	0,40514504	0,64441961	0,8313107	0,88197073	0,84055431	0,81687351

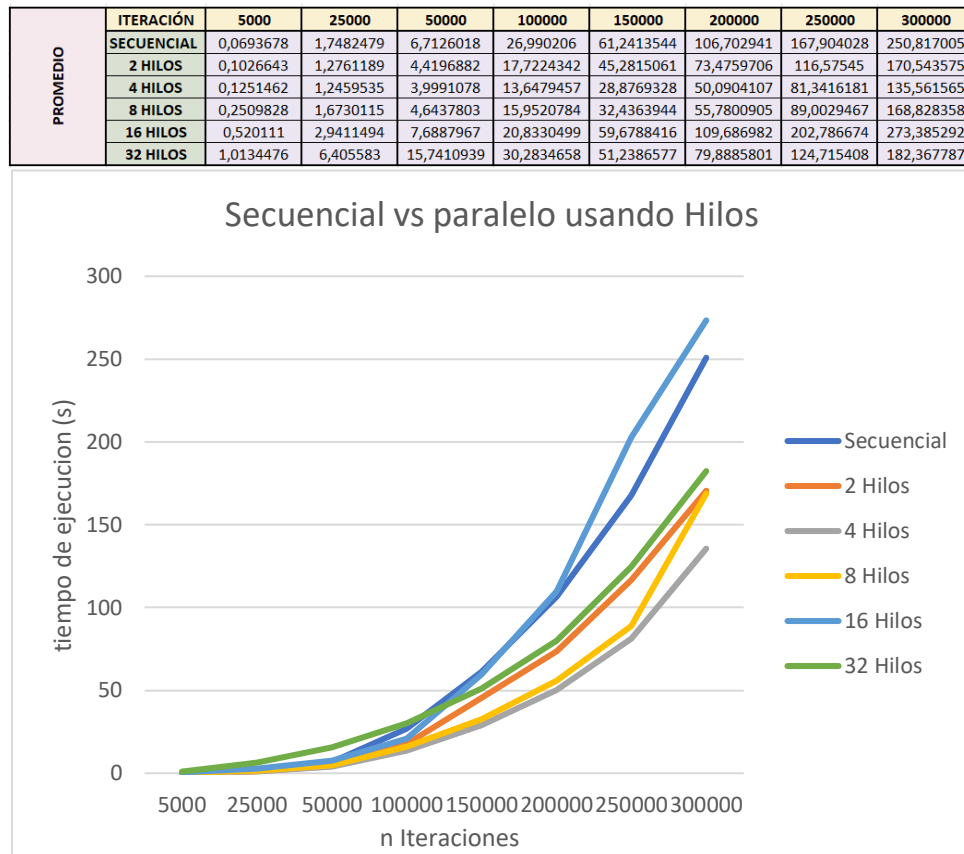
11. Resultados al ejecutar el algoritmo de forma paralela usando 32 procesos:

PARALELO 32 PROCESOS								
Valor contador de iteracion n	n = 5000	n = 25000	n = 50000	n = 100000	n = 150000	n = 200000	n = 250000	n = 300000
1	2,301973	12,766022	32,422937	71,312973	112,635757	165,823578	231,02488	316,482006
2	2,368016	12,995712	31,964375	71,273172	117,778035	165,858017	228,556129	306,823913
3	2,496684	14,087275	32,800203	69,996258	114,679677	167,103931	225,837847	301,761549
4	2,322195	14,765464	31,796799	70,849944	114,356929	165,32365	229,74349	309,774889
5	2,292931	14,9491	31,511077	71,66153	118,736422	166,850246	230,035737	301,417199
6	2,354132	15,251833	31,66366	70,321613	114,562554	167,141822	227,819848	311,967038
7	2,358383	15,528023	32,352126	72,356168	114,578661	166,590943	230,705983	312,771095
8	2,541709	14,92423	32,059125	71,142977	116,834791	167,275247	227,831501	306,297407
9	2,400996	15,574745	32,901834	71,661675	114,721368	169,377661	232,590811	303,69662
10	2,393286	15,282105	31,242844	71,245621	113,880511	171,631966	231,414358	314,298337
Promedio	2,3830305	14,6124509	32,071498	71,1821931	115,276471	167,297706	229,556058	308,529005
Speedup	0,02910907	0,11964098	0,20930116	0,37917076	0,53125633	0,63780278	0,7314293	0,81294465

Análisis de datos – Gráficas

Después de analizar los datos con las tablas sacándoles el promedio en cada una de ellas y su speedup, se procedió a juntar todos los datos encontrados para así hacerles un análisis más a fondo mostrando su comportamiento a través de gráficas para determinar si en realidad se estaba cumpliendo con el objetivo:

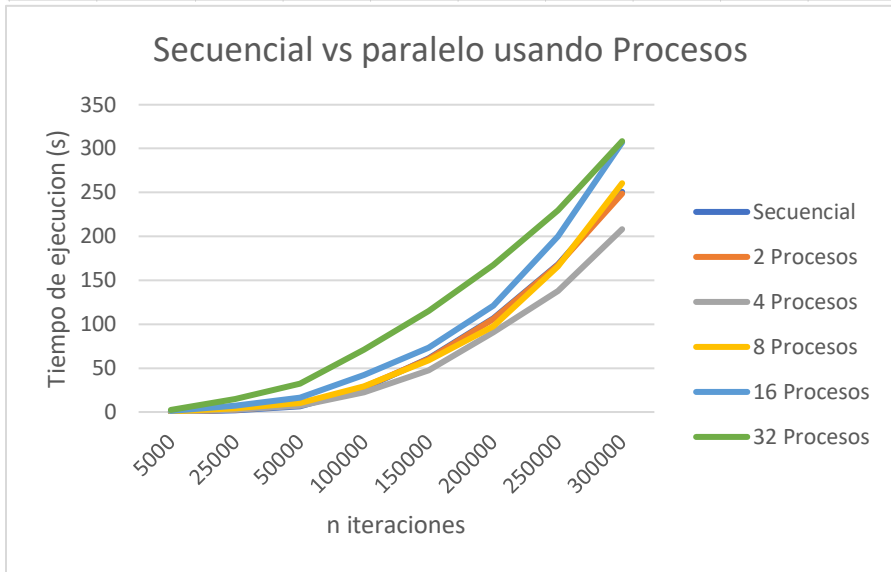
1. Resultados gráficos al comparar el promedio de forma secuencial con el promedio de forma paralela usando hilos sin su Speedup:



Gráfica 1. Secuencial vs paralelo usando Hilos

2. Resultados gráficos al comparar el promedio de forma secuencial con el promedio de forma paralela usando procesos sin su Speedup:

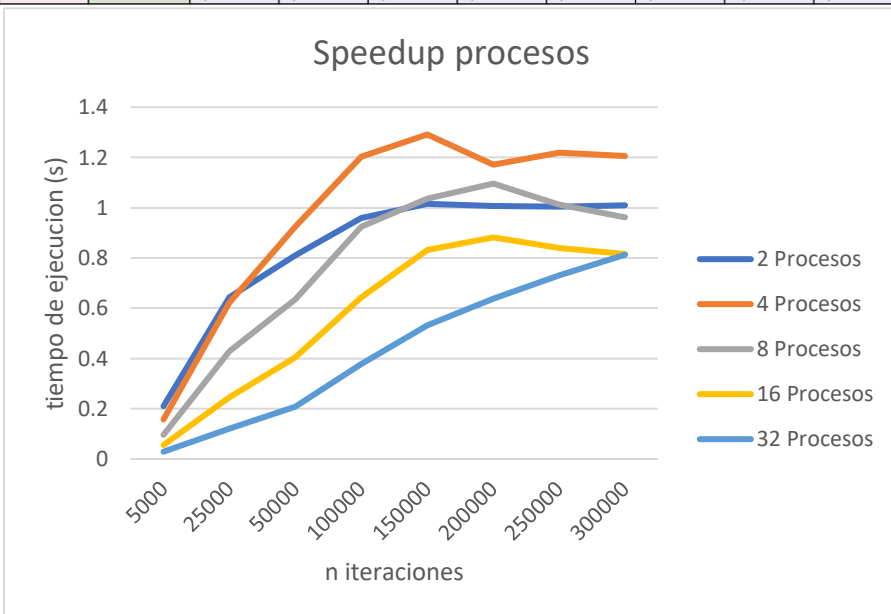
	ITERACIÓN	5000	25000	50000	100000	150000	200000	250000	300000
PROMEDIO	SECUENCIAL	0,0693678	1,7482479	6,7126018	26,990206	61,2413544	106,7029414	167,9040282	250,817005
	2 PROCESOS	0,3301178	2,7193618	8,2824349	28,1311492	60,3041203	105,8970025	167,310656	248,7488427
	4 PROCESOS	0,441154	2,813286	7,261847	22,4201062	47,4189459	91,0079771	137,6838268	208,2115856
	8 PROCESOS	0,7221711	4,0914312	10,5805114	29,2277746	59,0760416	97,3584326	165,9850286	260,5971964
	16 PROCESOS	1,2479754	7,1057343	16,5683919	41,8829684	73,6684307	120,9824068	199,7539324	307,0450955
	32 PROCESOS	2,3830305	14,6124509	32,071498	71,1821931	115,2764705	167,2977061	229,5560584	308,5290053



Gráfica 2. Secuencial vs paralelo usando Procesos

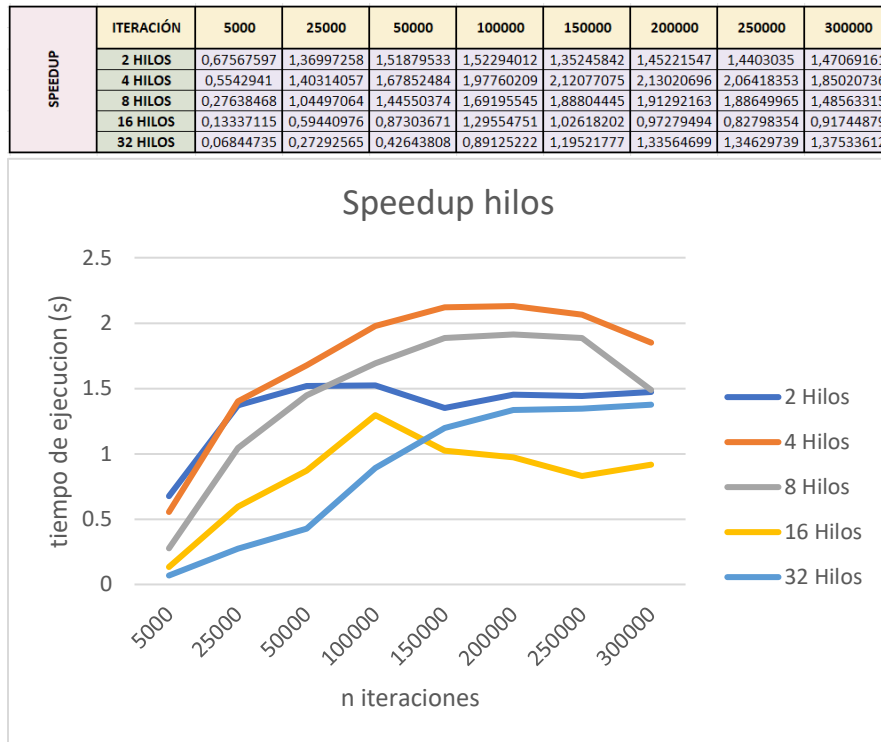
3. Resultados gráficos al comparar el speedup usando procesos:

	ITERACIÓN	5000	25000	50000	100000	150000	200000	250000	300000
SPEEDUP	2 PROCESOS	0,21013044	0,64288904	0,81046237	0,959442	1,01554179	1,00761059	1,00354653	1,00831426
	4 PROCESOS	0,15724169	0,62142559	0,92436563	1,20383935	1,29149548	1,17245702	1,21948984	1,20462559
	8 PROCESOS	0,09605452	0,42729495	0,63443075	0,92344376	1,03665298	1,09598048	1,01156128	0,96247008
	16 PROCESOS	0,05558427	0,24603339	0,40514504	0,64441961	0,8313107	0,88197073	0,84055431	0,81687351
	32 PROCESOS	0,02910907	0,11964098	0,20930116	0,37917076	0,53125633	0,63780278	0,7314293	0,81294465



Gráfica 3. Speedup procesos

4. Resultados gráficos al comparar el speedup usando hilos:



Observaciones:

- Durante el desarrollo del presente reto/laboratorio, se presentó un inconveniente que afectó el resultado de los cálculos realizados por el algoritmo de jacobi para sus versiones en hilos y procesos y fue que, durante la programación de dichos algoritmos, no se tuvo en cuenta que, para la distribución de la carga, el ultimo proceso/hilo debía tener como límite superior el valor que se obtenía en las fórmulas e incluirlo o en su defecto sumarle 1 para que a la hora que fueran ejecutados en los ciclos for se tomarán todos los valores de los arreglos u, utmp y f, causando así que las últimas posiciones de los arreglos no se tuvieran en cuenta y generando salidas ligeramente diferentes a la versión secuencial. El problema no se pudo atender a tiempo debido a que se percató del error muy tarde, sin tener oportunidad de volver a correr los algoritmos para un nuevo análisis.

Conclusiones

1. Una vez analizados los datos obtenidos en cada una de las implementaciones, se pudo concluir que la implementación con hilos es la mejor opción para paralelizar el algoritmo de jacobi. En base a lo anterior, el mejor rendimiento lo obtuvo la implementación de 4 hilos con un speedup de 2.130206958 para un tamaño de arreglo de 200000, siendo superior al mejor resultado obtenido por los procesos que tuvo un speedup de 1.291495482 utilizando 4 procesos en un arreglo de tamaño 150000, de manera que la implementación de procesos apenas pudo mejorar la versión secuencial.
2. En el caso específico de los hilos, se concluyó que el punto óptimo para paralelizar un algoritmo utilizando hilos va a depender de la cantidad de núcleos con la que cuente el procesador de la computadora, ya que en el caso de la máquina en la que se corrieron los programas era un procesador Intel Core i5 con 4 núcleo físico, coincidiendo con la versión en la que se obtuvo mayor rendimiento (versión de 4 hilos).
3. Enlazado al punto anterior, su pudo notar que, en general para arreglos de tamaño menor a 25000, el rendimiento de la implementación de hilos empeora. Con lo cual, se puede intuir que debido a la implementación de código y a la división de la carga de los hilos se pudo generar un peor rendimiento comprado a su contraparte secuencial para arreglos de un tamaño considerablemente pequeño. Sin embargo, a partir de valores mayores o iguales a arreglos de tamaño 25000 para las versiones de 2, 4 y 8 hilos, el rendimiento mejora, situación que no es el caso para la implementación de 32 hilos, ya que solo mejora a partir de valores mayores a 150000.
4. Para el caso de la implementación a través de procesos, se evidenció que no vale la pena paralelizar el algoritmo con valores diferentes a 4 procesos, ya que, en las versiones de 2, 8, 16 y 32 el desempeño en general era peor que la implementación secuencial. Incluso, en la versión de 4 el rendimiento solo mejoraba para tamaños de arreglos mayores a 50000. Por otra parte, el no tan buen rendimiento de los procesos de pudo deber en gran parte al hecho de que los procesos tenían que acceder a un área de memoria que no les pertenecía generando así un cambio de contexto muy costoso que hizo que la implementación a través de procesos fuera, de manera general, peor que las implementaciones secuenciales y por hilos.
5. Por último, cabe resaltar que, los resultados del presente reto/laboratorio no indican que los procesos sean necesariamente peores, sino que los procesos debería ser empleados en otro tipo de problemas ya que estos presentan algunas características interesantes que los hilos no, por ejemplo, los procesos tienen su propio espacio de memoria que es independiente de los demás procesos creados por un mismo proceso padre. También está el hecho de que, si el proceso padre termina, no lo harán los procesos hijos, característica que no presentan los hilos y la cual puede ser una propiedad interesante a tener en cuenta dependiendo del tipo de problema que se esté enfrentando.

Bibliografía

- *Método de jacobi - Ensayos - 848 Palabras.* (n.d.). Retrieved September 19, 2022, from <https://www.buenastareas.com/ensayos/Metodo-De-Jacobi/2123409.html>
- *¿Qué son los métodos numéricos?* (n.d.). Retrieved September 19, 2022, from <https://metodosvistos.blogspot.com/2020/11/que-son-los-metodos-numericos.html>
- *Método de Jacobi.* (n.d.). Retrieved September 19, 2022, from <https://www.metodosnumericosunitec.com/post/la-vida-bajo-el-microscopio-lo-que-no-se-puede-ver>