Proyecto No.2:
OpenMPI

Oliver Graf 17190
Estuardo Ureta 17010
Kristen Brandt 171482

Índice

Antecedentes conceptuales

GitHub

GitHub es una herramienta para el control de versiones de proyectos, normalmente este se utiliza para proyectos de código. Esta herramienta permite que haya varios colaboradores en el desarrollo del proyecto y se mantenga un orden en el trabajo de los mismos (Github, 2022).

OpenMPI

OpenMPI es una biblioteca estándar para realizar procesamiento en paralelo haciendo uso de un modelo de memoria distribuida. Esta biblioteca es open source y se puede correr en Linux, OS, Windows (Open MPI, 2022).

Cifrado/Decifrado

El cifrado es el proceso que traduce los datos sin formato especial (“plaintext") a algo que no parece tener sentido y parece ser aleatorio (“ciphertext”). Descifrado es el proceso para pasar de los datos “sin sentido” de regreso a un texto sin formato especial. La idea de hacer un proceso de cifrado es que solo las personas autorizadas a ver los datos los puedan ver mientras que personas no autorizadas no puedan tener acceso a la información inicial. Ahora este proceso de cifrado y descifrado se utiliza en las computadoras y las comunicaciones que se tienen a través de ellas (Fox, 2022).

Bruteforce

Un ataque de brute force o fuerza bruta es un método de hacking que utiliza el método de prueba y error para poder descifrar todo tipo de documentos o información (Fortinet, 2020). En el caso del proyecto se utilizará un programa de bruteforce para poder descubrir la llave privada usada para cifrar un texto. Dicho programa tendrá iteraciones hasta encontrar la clave deseada.

Algoritmo DES

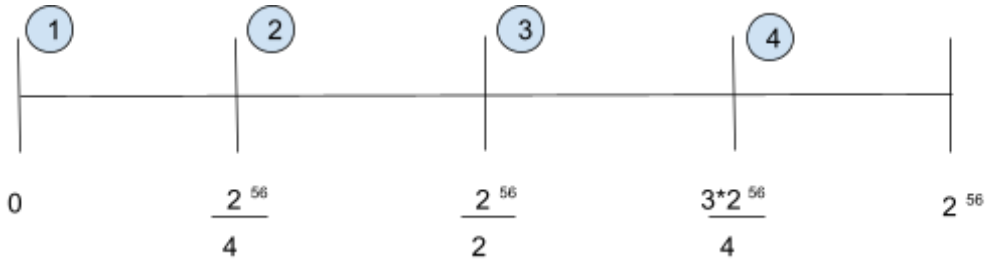
Data Encryption Standard, también conocido como DES, es un algoritmo de cifrado para data digital. Aunque este algoritmo ya no es considerado como uno seguro por su clave de solamente 56 bits este ha influenciado un gran avance en la criptografía que se utiliza hoy en día. Este algoritmo fue desarrollado en los 1970 en International Business Machines Corporation (IBM) y fue seleccionado por el National Security Agency (NSA) para la protección de datos gubernamentales electrónicos confidenciales

y no clasificados. Aunque este algoritmo era reforzado contra el criptoanálisis diferencial no era muy bueno en contra de ataque de fuerza bruta. En 1977 este algoritmo se volvió el estándar federal de procesamiento de información (FIPS) para los Estados Unidos (DES, 2020).

Antecedentes numéricos

Speedups inconsistentes

Se pueden dar speedups inconsistentes y dependientes de la llave elegida dado a que se recorre el espacio de datos de forma incremental y en orden, aparte se dividen equitativamente los espacios.



Posibles claves y cómo los speedups pueden ser inconsistentes

Los speedups dependen mucho de la llave que se elige.

Si se corre el programa de manera paralela con 4 distintos procesos, si se escoge una llave de $\frac{2^{56}}{2} + 1$ se encontrara la llave en la primera iteración por el 3er proceso pero si se utiliza la misma clave de $\frac{2^{56}}{2} + 1$ en el programa secuencial el programa se toma $\frac{2^{56}}{2} + 1$ iteraciones para encontrar la respuesta. Aquí se daría un speedup extremadamente alto.

Otro ejemplo de las inconsistencia es si se escoge una llave de $\frac{2^{56}}{4}$ se encontrará la llave en exactamente la misma iteración sin importar si el programa es paralelo o no. En este caso no se daría ningún speedup en el programa.

Diagrama de flujo del algoritmo DES

DES se fundamenta de dos atributos fundamentales de la criptografía. La primera es la sustitución (también llamada confusión) y la segunda es la transposición (también llamada difusión). El algoritmo en cuestión consta de 16 pasos, cada uno de los cuales se les llaman una ronda. Cada ronda realiza los pasos de sustitución y transposición. Hablando de los pasos tenemos los siguientes: En el primer paso, el bloque de texto sin formato de 64 bits se transfiere a una función de permutación en el inicial (PI) . La permutación inicial se realiza en texto sin formato. Luego, la PI produce dos mitades del bloque permutado. Una de las mitades denominándose LPT o permutación izquierda de texto por su siglas en inglés. La otra mitad denominada permutación derecha de texto sin formato (RPT). Luego, cada LPT y RPT pasan por 16 rondas del proceso de encriptación. Al final, LPT y RPT se vuelven a unir y se realiza una permutación final (FP) en el bloque combinado el resultado de este proceso produce texto cifrado de 64 bits.

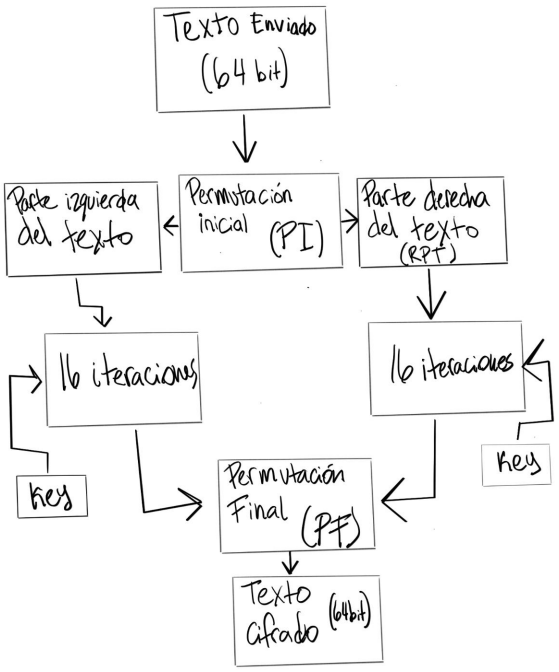


Diagrama 1. Flujo del algoritmo DES (Data Encryption Standard)

El algoritmo DES es usado como un cifrado de bloque de clave simétrica. El algoritmo toma el texto sin formato en bloques de 64 bits y los convierte en texto cifrado utilizando claves de 48 bits. Se puede utilizar para prácticamente cualquier tipo de cifrado. Pero siendo prácticos y estando actualizados con nuevas tecnologías podemos saber que el único uso práctico actual de DES es como bloque de construcción para llegar a Triple DES. Al momento de ser usado ya hace décadas se utilizaba para todo tipo de cifrado, como se menciona anteriormente. En ese momento, este era similar y usado con igual o parecida frecuencia a AES, por ejemplo.

Introducción

En el proyecto No.2 de computación paralela y distribuida se utilizó OpenMPI, para diseñar un programa que sea capaz de encontrar una llave privada con la cual fue cifrado un texto sin formato (“plain text”). Este proyecto se hizo con el objetivo de diseñar programas para la paralelización de procesos con memoria distribuida utilizando la librería de OpenMPI además de optimizar el uso de recursos distribuidos y mejorar el speedup del programa paralelo.

Se utilizó el algoritmo de DES mencionado en los antecedentes conceptuales para poder cifrar y luego descifrar un texto y descubrir la llave privada utilizada. Este algoritmo se escribió utilizando C + +.

Metodología

- Se modificó el programa Bruteforce00.c para utilizar una librería que reemplace a res/des _crypt.h.
- Se utilizó el código encontrado en https://www.techiedelight.com/des-implementation-c/ como una base pasa las funciones decrypt y encrypt.
- Se realizó un programa secuencial para encontrar claves con fuerza bruta utilizando C + + .
- Utilizando el programa secuencial como base se realizó el programa paralelo utilizando OpenMPI.
- Se hicieron varias corridas con claves de distinta dificultad y se calcularon los speedups
- Se desencriptaron los textos de otros grupos.

Diagrama de flujo del programa

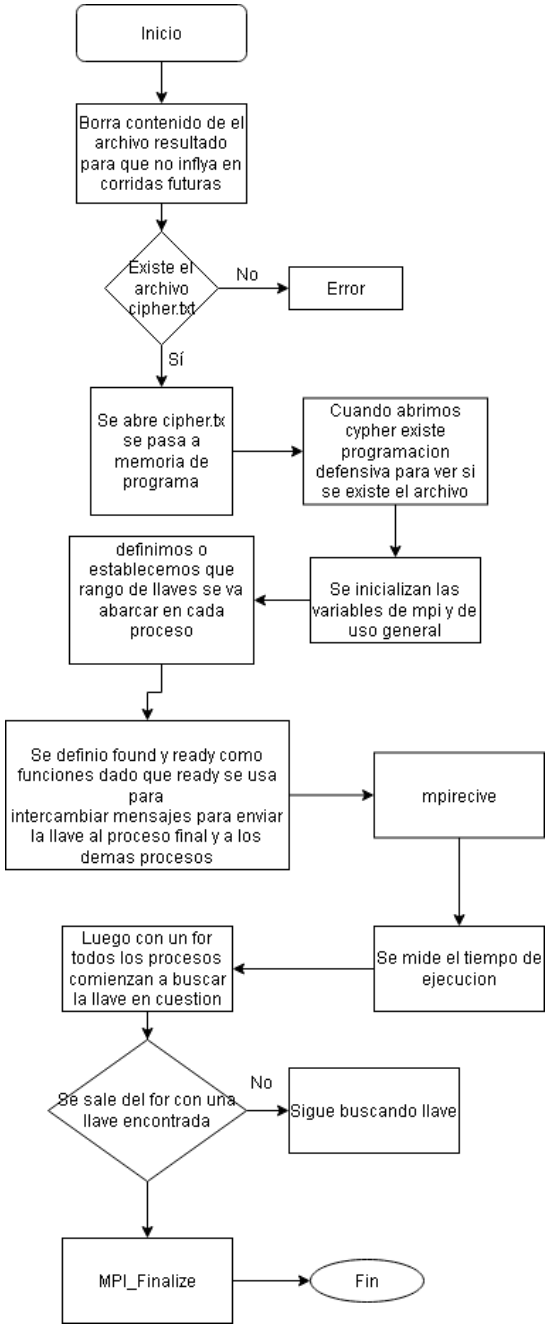
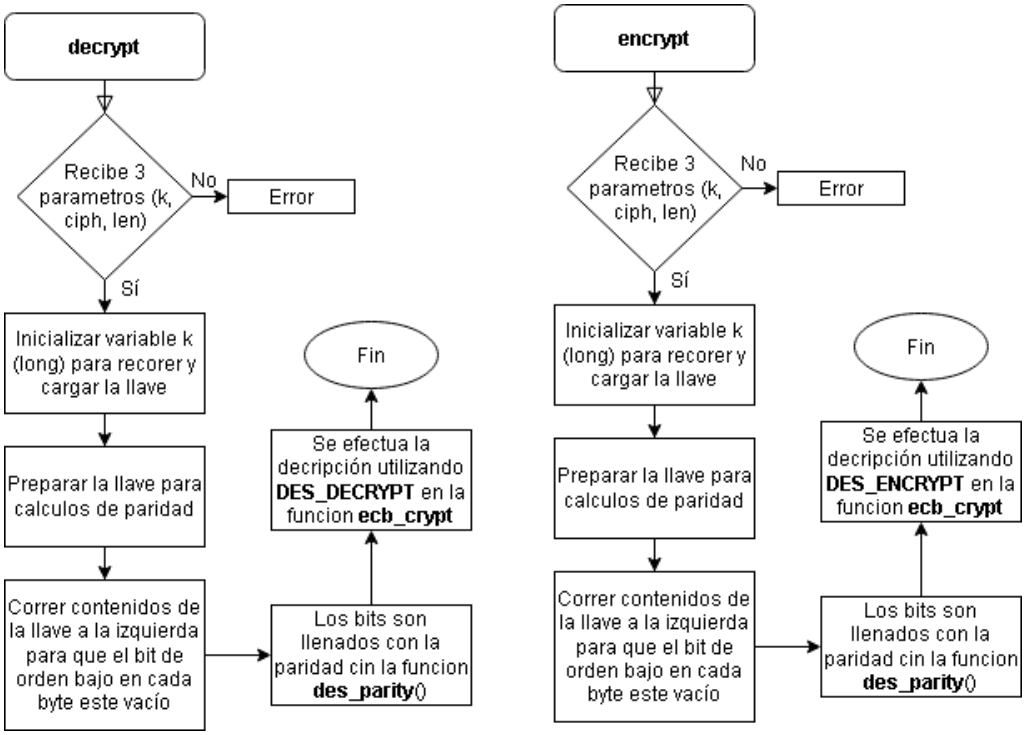
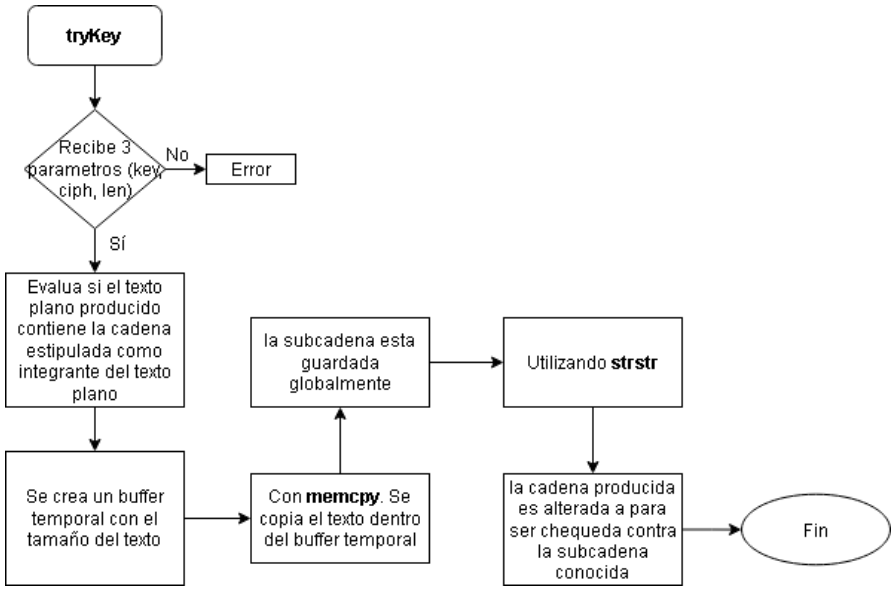

Inicio
Borra contenido de el archivo resultado para que no inflya en corridas futuras
Existe el archivo cipher.txt
No
Error
Sí
Se abre cipher.tx se pasa a memoria de programa
Cuando abrimos cypher existe programacion defensiva para ver si se existe el archivo
definimos o establecemos que rango de llaves se va abarcar en cada proceso
Se inicializan las variables de mpi y de uso general
Se definio found y ready como funciones dado que ready se usa para
intercambiar mensajes para enviar la llave al proceso final y a los demas procesos
mpirecive
Luego con un for todos los procesos comienzan a buscar la llave en cuestion
Se mide el tiempo de ejecucion
Se sale del for con una llave encontrada
No
Sigue buscando llave
MPI_Finalize
Fin

Diagrama de rutinas

decrypt (key, *ciph, len) y encrypt (key, *ciph, len)



tryKey (key, *ciph, len), memcpy, strstr



Funciones primitivas de MPI

MPI_Irecv

Esta función es para comenzar el recibimiento de un mensaje. Lo que hace es bloquear el proceso hasta que se le notifique la llegada de un mensaje. Cuando esto suceda, pedirá que se comience a recibir el mensaje, a la vez que continúa la ejecución del resto del proceso. Una vez nos es necesario utilizar el mensaje, es obligatorio utilizar alguna directiva de MPI para detener la ejecución (como MPI_Wait), o bien comprobar el estado del recibo (por ejemplo con MPI_Test).

MPI_Send

Función de envío de mensaje bloqueante de un proceso de origen a uno de destino. Al ser bloqueante significa que hasta que el mensaje no haya sido enviado (que salga del buffer de salida) no se continúa la ejecución.

MPI_Wait

Este método bloquea el proceso que lo invoca hasta que la operación indicada en request se complete. Una vez se completa, se rellena la variable en el parámetro de salida status con los datos propios del objeto de tipo MPI_Status.

Resultados

Programa paralelo sin la optimización (llave fácil 2200)

Corrida	1 proceso	2 procesos	4 procesos	6 procesos
1	3.128	3.624	4.377	1.191
2	3.145	3.937	4.251	1.177
3	3.381	3.715	4.634	1.242
4	3.729	3.681	4.289	1.261
5	3.156	3.731	4.418	1.154
6	3.134	3.831	4.389	1.149
7	3.147	4.154	4.461	1.111
8	3.153	3.699	4.293	1.103
9	3.356	3.941	4.469	1.122

10	3.139	3.956	4.566	1.173
Promedio	3.247	3.827	4.415	1.168

$$speedup = \frac{t_{seq}}{t_{par}}$$

	Secuencial	2 procesos	Speedup
Promedio 10 corridas	3.247	3.827	0.8484151663

	Secuencial	4 procesos	Speedup
Promedio 10 corridas	3.247	4.415	0.7354520126

	Secuencial	6 procesos	Speedup
Promedio 10 corridas	3.247	1.168	2.779080716

Programa paralelo sin la optimización (llave mediana 4578)

Corrida	1 proceso	2 procesos	4 procesos	6 procesos
1	6.177	7.602	3.623	2.183
2	6.167	7.362	3.571	2.102
3	6.067	7.402	3.661	2.182
4	6.229	8.078	3.594	2.206
5	6.827	7.539	3.595	2.125
6	6.301	7.451	3.783	2.121
7	6.274	7.571	3.797	2.191
8	6.272	7.587	3.716	2.072
9	6.267	7.512	3.512	2.354
10	6.223	8.125	3.584	2.167
Promedio	6.280	7.623	3.644	2.170

$$speedup = \frac{t_{seq}}{t_{par}}$$

	Secuencial	2 procesos	Speedup

Promedio 10 corridas	6.280	7.623	0.8238859227

	Secuencial	4 procesos	Speedup
Promedio 10 corridas	6.280	3.644	1.723679877

	Secuencial	6 procesos	Speedup
Promedio 10 corridas	6.280	2.170	2.893793485

Discusión

La encripción y decriptación siempre ha sido un tema fascinante en el sector de la computación. En el estudio presente, aunque podría ser categorizado como superficial, es un ejemplo claro de cómo es posible sincronizar y balancear la carga de procesos de MPI. El algoritmo DES es un método de encriptación que utiliza una llave secreta de 56 bits para los procesos de encriptación y decriptación de la data, en este caso un mensaje.

Inicialmente contamos con un programa en el lenguaje de programación (bruteforce.c) el cual utilizaba una librería no funcional (res/des_crypt.h). Por lo tanto utilizamos un recurso diferente para manejar ciertas funciones necesarias. Con esto en mente se utilizó C + + para implementar dicho algoritmo en una clase por aparte. Nuestro programa, en esencia, utilizó MPI para diseñar un programa que encuentre la llave privada con la que fué cifrado un texto plano. A partir de un texto y su conversión a binario DES es implementado por la función Encryption. La llave previamente mencionada es utilizada al correr sus contenidos a la izquierda para que el bit de orden bajo en cada byte sea mantenido vacío. Estos bits son llenados con su paridad. Se debe mencionar que para decidir si el texto producido, usando la llave, sea correcto debemos asumir que parte del mensaje es conocido. Por lo que no se tomaron en cuenta llaves que no produzcan el texto que contiene una subcadena dada. En el peor de los escenarios tendríamos que probar 2^(56) llaves posibles. Por lo que entra la interrogante sobre cómo podemos particionar el espacio de búsqueda la llave a manera de minimizar el tiempo de ejecución. Una manera es haciendo la división de la carga de trabajo a priori. Podemos tener particiones tanto iguales como desiguales en función de las capacidades de los nodos que componen la plataforma de ejecución.

La única pregunta persistente que debe responderse es qué debería suceder una vez que se haya encontrado una solución candidata. Se podría continuar hasta

agotar todas las claves o terminar el programa. La terminación anticipada es un problema más difícil de abordar y, por esta razón, es el enfoque elegido de partición estática. Una solución fácil al problema de partición fue hacer que cada proceso MPI calcule de qué parte del espacio de búsqueda clave será responsable. Con esto mente nuestros resultados mostraron que en el programa secuencial los cambios significativos en el speedup se notaron al utilizar una llave fácil o sencilla con un speedup de 3.247 y con la llave mediana un speedup de 6.280. Mientras que en el programa paralelo con 6 procesos se obtuvo el mejor speedup, de 2.78, para una clave de dificultad fácil y hablamos de un speedup de 2.89 con dificultad media. Cabe destacar que los speedups adquiridos deben su inconsistencia directamente a la llave escogida. Dado que recorremos el espacio de datos de una manera ordinal y equitativa (también conocido como recorrer el espacio de manera *naive*).

Conclusiones/Recomendaciones

- Ya no es recomendable utilizar el algoritmo DES por la facilidad que tienen los ataques de fuerza bruta de conseguir la llave.
- Se recomienda mantener un orden el el código cuando se está trabajando con paralelismo.
- El cifrado se puede hacer con distintos encoding, unos ejemplos son binario, hexadecimal y caracteres.
- El speedup podría ser nulo si la clave se encontraba dentro de las claves de 0 a $\frac{2^{56}}{4}$. En estos casos el paralelo se tarda más por tener que inicializar los procesos.
- Según la Tabla No. 1 el programa paralelo con 6 procesos obtuvo el mejor speedup de 2.78 para una clave de dificultad fácil.
- Según la Tabla No. 2 el programa paralelo con 6 procesos obtuvo el mejor speedup de 2.89 para una clave con dificultad media.

Anexos

- Cronograma de actividades

UNIVERSIDAD DEL VALLE DE GUATEMALA
Proyecto 2 MPI
CRONOGRAMA DE ACTIVIDADES

FASE	ACTIVIDADES / ESTRATEGIAS	RESPONSABLE	Abril			Mayo	
			2	3	4	1	2
Etapa 1	Planificación de actividades						
	Inicializar GIT						
	Documentación preliminar						
	Bosquejo de Informe						
	Código parcial						
	Mediciones de tiempo sobre código base						
Etapa 2	Programación defensiva						
	Antecedentes numéricos						
	Codigo (inicial)						
	Diagrama de flujo de su programa						
	catálogo de las funciones						
	Revision						
	Mediciones de tiempo						
	Modificaciones						
	Calculo de Speed up						
	Correcciones de parametros						
FINAL	Reporte						
	Retos encontrados para la implementación						
	Bitacora						
	Recomendaciones						
	Resultados						
	Discusión y conclusiones						

- Catálogo de funciones
  - Para programa de descifrado con fuerza bruta.

	Entradas	Salidas	Descripción
findFileSize	n/a	Long int	Devuelve el tamaño del archivo cifrado. Se usa para que la función decrypt pueda alojar el espacio correcto en sus variables internas

decrypt	Long int n Long int plain[]	string	Recibe el texto cifrado en bits y lo descifra, devolviéndolo en un string. n es utilizado para alojar el espacio correcto en memoria para las variables temporales
generateKey	Long int	Unsigned int *	Recibe un entero y devuelve una llave de 64 bits.
tryKey	Long int inkey Long int plain[] Long int n	bool	Recibe un entero (que es una posible llave), el texto cifrado en bits y lo descifra, devolviendo verdadero si el texto contiene una palabra clave y falso si no contiene la palabra. n es utilizado para alojar el espacio correcto en memoria para las variables temporales.

- Bitácora de pruebas
- Descifrado de texto de 2 grupos
    - Grupo de Majo
        - Se pasó a binario antes de descifrar

```
oligraf@MSI:~/Desktop/cparalela/proyecto2/programa$ mpic++ -fopenmp des_alg
_par.cpp -o des_alg_par -lstdc++
oligraf@MSI:~/Desktop/cparalela/proyecto2/programa$ mpirun -np 6 ./des_alg_
par
Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'bubble' en el texto

El tiempo de ejecucion fue: 0.043879
La llave que descifra el texto es: 8
oligraf@MSI:~/Desktop/cparalela/proyecto2/programa$
```

Corrida	1 proceso	2 procesos	4 procesos	6 procesos
1	0.031	0.032	0.031	0.044
2	0.026	0.028	0.032	0.034
3	0.027	0.034	0.032	0.031
4	0.032	0.027	0.029	0.033
5	0.029	0.027	0.035	0.045
6	0.025	0.031	0.031	0.033
7	0.031	0.028	0.037	0.033
8	0.033	0.023	0.027	0.036
9	0.028	0.027	0.033	0.049
10	0.025	0.029	0.031	0.046

Promedio	0.0287	0.0286	0.0318	0.0384

	Secuencial	2 procesos	Speedup
Promedio 10 corridas	0.029	3.827	0.007499542711

	Secuencial	4 procesos	Speedup
Promedio 10 corridas	0.029	0.032	0.9025157233

	Secuencial	6 procesos	Speedup
Promedio 10 corridas	0.029	0.038	0.7473958333

- Grupo Augusto
    - Se pasó a binario antes de descifrar
-

```
oligraf@MSI:~/Desktop/cparalela/proyecto2/programa$ mpirun -np 6 ./des_alg_
par
Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

Leyendo del archivo 'cipher.txt' el texto cifrado en bits
Buscando la palabra 'systems' en el texto

El tiempo de ejecucion fue: 30.191812
La llave que descifra el texto es: 34464
oligraf@MSI:~/Desktop/cparalela/proyecto2/programa$
```

Corrida	1 proceso	2 procesos	4 procesos	6 procesos
1	21.236	24.214	26.919	33.216
2	23.252	24.205	28.849	32.294
3	24.244	25.602	27.512	35.632
4	21.002	23.698	25.426	34.235
5	20.909	21.453	25.381	31.524
6	24.531	24.891	27.583	33.537
7	23.012	26.239	25.474	37.754
8	23.411	22.453	26.753	34.462
9	21.041	23.522	28.742	33.453
10	22.054	24.573	28.525	32.353
Promedio	22.4692	24.085	27.1164	33.846

	Secuencial	2 procesos	Speedup
Promedio 10 corridas	22.469	24.085	0.9329126012

	Secuencial	4 procesos	Speedup
Promedio 10 corridas	22.469	27.116	0.8286203183

	Secuencial	6 procesos	Speedup
Promedio 10 corridas	22.469	33.846	0.6638657448

Literatura Citada

Fox, P. (2022). *Cifrado, Descifrado y cracking (artículo)*. Khan Academy. Retrieved from https://es.khanacademy.org/computing/ap-computer-science-principles/x2d2f703b37b450a3:online-data-security/x2d2f703b37b450a3:data-encryption/a/encryption-decryption-and-code-cracking

Lecture 4 data encryption standard (DES) - LRI. (2020). Retrieved from https://www.lri.fr/~fmartignon/documenti/systemesecurite/4-DES.pdf

Open MPI: Open source high performance computing. Open MPI: Open Source High Performance Computing. (2022). Retrieved from https://www.open-mpi.org/

What is a brute force attack?: Definition, Types & How It Works. Fortinet. (2020). Retrieved from https://www.fortinet.com/resources/cyberglossary/brute-force-attack

Where the world builds software. GitHub. (2022). Retrieved from http://github.com/