**UNIVERSIDAD DEL VALLE DE GUATEMALA**

CC3069 - Computación Paralela y Distribuida

Sección 21

Ing. Juan Luis García Zarceño


# Proyecto No. 2

## Fuerza Bruta - Cifrados


| | |
|---|---|
| José Pablo Orellana | 21970 |
| Diego Alberto Leiva | 21752 |


Guatemala, 11 de octubre del 2024

# DES (Data Encryption Standard)

El Data Encryption Standard (DES) es un algoritmo de cifrado simétrico, desarrollado por IBM en la década de 1970. DES utiliza una clave secreta para cifrar y descifrar bloques de datos, lo que significa que la misma clave se utiliza tanto para el cifrado como para el descifrado del mensaje.

**Características principales de DES:**

- **Tamaño del bloque:** DES trabaja con bloques de 64 bits (8 bytes) de datos.
- **Tamaño de la clave:** La clave utilizada en DES es de 64 bits, aunque solo 56 bits se utilizan efectivamente, ya que los 8 bits restantes son de paridad.
- **Algoritmo simétrico:** La misma clave se utiliza para cifrar y descifrar los datos.
- **Estructura Feistel:** DES utiliza la estructura Feistel, lo que significa que el proceso de cifrado y descifrado es muy similar, con pequeñas diferencias en el uso de la clave.

**Pasos requeridos para cifrar/descifrar un texto con DES**

1. **División del mensaje en bloques de 64 bits:** El mensaje que se desea cifrar se divide en bloques de 64 bits. Si el mensaje no es múltiplo de 64 bits, se rellena.
2. **Generación de subclaves:** La clave de 64 bits se reduce a 56 bits (eliminando bits de paridad) y luego se divide en dos mitades de 28 bits cada una. A partir de estas mitades, se generan 16 subclaves diferentes, cada una de 48 bits, mediante un proceso de permutación y desplazamientos circulares a la izquierda.
3. **Estructura Feistel de 16 rondas:** Cada bloque de 64 bits se divide en dos mitades: una mitad izquierda (L) y una mitad derecha (R), ambas de 32 bits. El cifrado se lleva a cabo en 16 rondas, en las que se realizan las siguientes operaciones:

    - **Expansión:** La mitad derecha de 32 bits se expande a 48 bits utilizando una permutación conocida como tabla E.
    - **XOR con la subclave:** El resultado de la expansión se combina con una de las subclaves generadas mediante una operación XOR.
    - **S-Box:** El resultado del XOR se divide en 8 bloques de 6 bits, y cada bloque pasa por una sustitución en las cajas S, que reducen los 6 bits a 4 bits.
    - **Permutación:** Los 32 bits resultantes pasan por P-box.
    - **Función Feistel:** La mitad izquierda (L) se combina con el resultado de la permutación usando una operación XOR, y las mitades se intercambian.

4. **Rondas finales:** Después de las 16 rondas, se combinan las mitades izquierda y derecha y se aplica una permutación final inversa, conocida como IP-1, para obtener el texto cifrado.
5. **Descifrado:** El descifrado en DES sigue los mismos pasos que el cifrado, pero las subclaves se aplican en orden inverso (de la última a la primera).

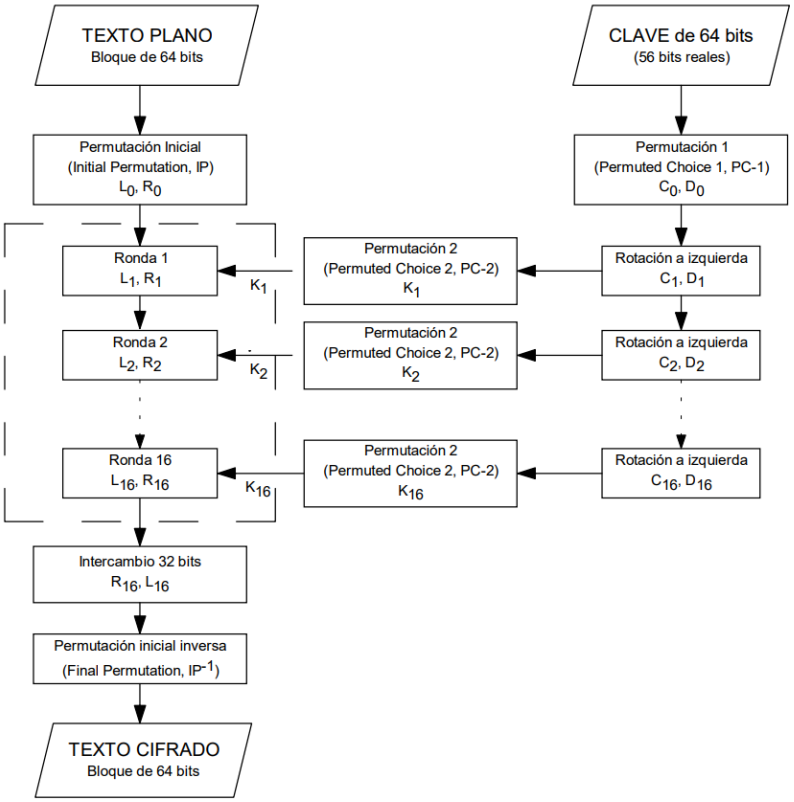



**Figura 1: Esquema general del algoritmo DES**

1. **Entrada del texto plano y la clave:**
   - El proceso comienza con un bloque de 64 bits de texto plano y una clave de 64 bits. La clave tiene 56 bits efectivos.

2. **Permutación inicial (IP):**
   - Se aplica una permutación inicial al bloque de texto plano, dividiendo el bloque en dos mitades de 32 bits: L0 (parte izquierda) y R0 (parte derecha).
3. **Generación de subclaves:**
   - La clave de 64 bits se somete a una Permutación 1 (PC-1), que reduce la clave a 56 bits y la divide en dos partes.
4. **Rondas de cifrado (16 rondas):**
   - El proceso de cifrado se lleva a cabo en 16 rondas. En cada ronda i:
     - Li es igual a Ri−1. Ri se calcula aplicando una función de sustitución y permutación (función F) a Ri−1 y a la subclave Ki.
5. **Intercambio de mitades (Swap):**
   - Después de la última ronda, se realiza un intercambio de mitades, intercambiando R16 y L16.
6. **Permutación final inversa ($IP^{-1}$):**
   - Se aplica una permutación inversa al bloque resultante.
7. **Salida:**
   - El resultado es un bloque de 64 bits de texto cifrado.

# Parte A

Para este inciso realizamos el cifrado y descifrado de un archivo de texto utilizando el algoritmo DES (Data Encryption Standard) con la biblioteca OpenSSL y realiza un ataque de fuerza bruta para encontrar la llave de cifrado, esto mediante programación secuencial. Para este apartado tuvimos estos apartados principales:

- **Adaptación de la llave:** La función adaptarLlaveNumerica convierte un número de 56 bits en una llave de 8 bytes compatible con DES.
- **Cifrado y descifrado:** Las funciones codificarTexto y descifrarTexto usan la biblioteca OpenSSL para cifrar y descifrar el texto en bloques de 8 bytes.
- **Lectura de archivos:** La función cargarArchivo carga el contenido de un archivo binario en la memoria.
- **Fuerza bruta:** fuerzaBrutaLlave intenta descifrar el texto cifrado probando todas las posibles llaves de 56 bits hasta encontrar una que produzca el texto original, midiendo el tiempo de ejecución.
- **Main:** El programa toma como argumentos un archivo de entrada y una llave, cifra el contenido del archivo, y luego realiza el ataque de fuerza bruta para intentar descubrir la llave utilizada en el cifrado.

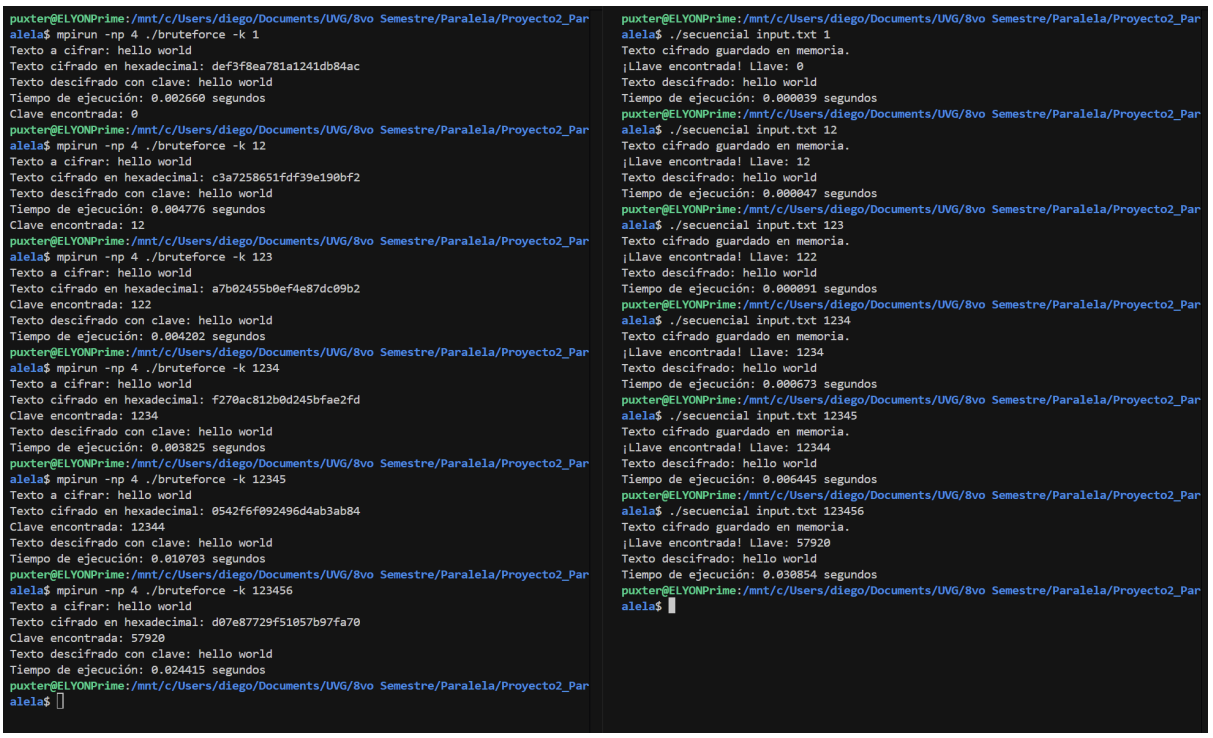
Además de realizar la modificación del código proporcionado, llamado “brutefoce.c” para este código implemenntamos un ataque de fuerza bruta para descifrar un texto cifrado con el algoritmo DES, utilizando la biblioteca OpenSSL y el entorno de programación paralela MPI (Message Passing Interface).

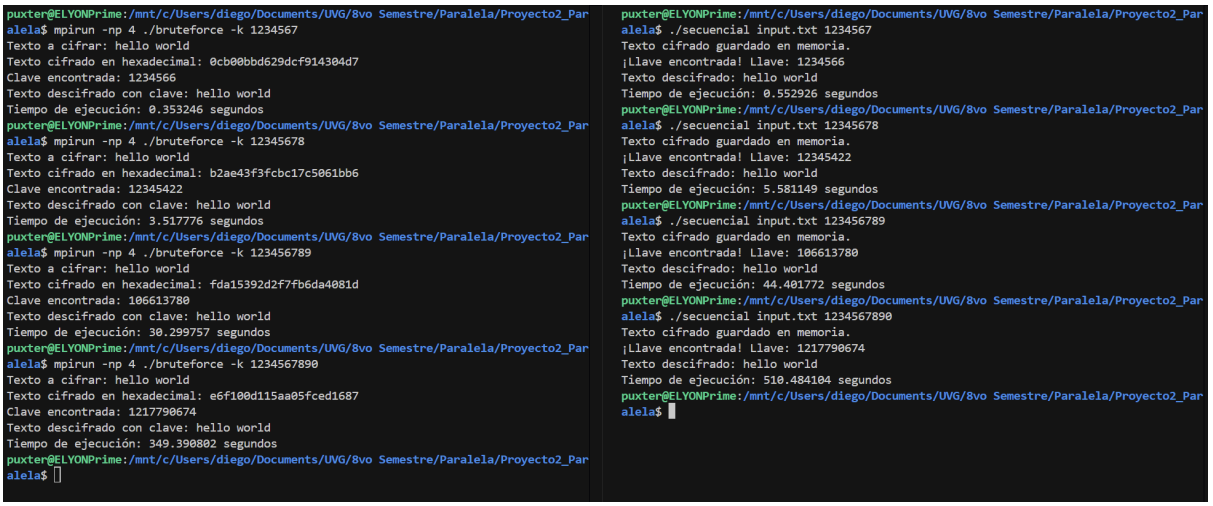
- **Cifrado y Descifrado:** Las funciones encrypt y decrypt realizan el cifrado y descifrado de un bloque de texto utilizando una clave de 56 bits.
- **Ataque de Fuerza Bruta Paralelo:** El programa distribuye el espacio de claves entre varios nodos MPI. Cada nodo prueba un rango específico de claves para descifrar el texto y busca una palabra clave. Si la clave correcta se encuentra, se notifica a todos los nodos para detener la búsqueda.
- **Lectura de Archivo y Comunicación MPI:** La función retrieveText carga el contenido de un archivo llamado "input.txt" y lo prepara para el cifrado.
- **Rango de Claves por Nodo:** Cada nodo calcula su rango de claves (mylower y myupper) y realiza un ciclo de fuerza bruta probando las claves en su rango.
- **Sincronización y Resultados:** El nodo 0 espera hasta que se reciba la clave correcta. Una vez encontrada, descifra el texto cifrado, lo imprime, y muestra el tiempo total de ejecución.
- **Argumentos y Validación:** El programa requiere que el usuario especifique una clave de cifrado mediante la opción -k en la línea de comandos. Se valida que la clave esté dentro del rango permitido para DES (1 a 2^56 - 1).

# Prueba de códigos

**Bitácora**

Para estas pruebas se realizó la ejecución de los mismos inputs con las mismas llaves en ambos algoritmos. El input utilizado fue “hello world” mientras que a lo largo de las pruebas las llaves fueron cambiando entre 1 dígito y 10 dígitos en cada prueba respectivamente.




En esta imagen podemos observar 10 pruebas tanto en el algoritmo secuencial como en el bruteforce. Donde utilizamos los siguientes datos para realizar las pruebas. **Input: *“Hello World”*** y las **keys usadas**, son las siguientes: ***1, 12, 123, 1234, 1345, 123456, 1234567, 12345678, 123456789 y 1234567890.***

## Resultados

- **Primer prueba**
  - **Secuencial:** 0.000039 seg
  - **Bruteforce:** 0.002660 seg
- **Segunda prueba**
  - **Secuencial:** 0.000047 seg
  - **Bruteforce:** 0.004776 seg
- **Tercer prueba**
  - **Secuencial:** 0.000091 seg
  - **Bruteforce:** 0.004202 seg
- **Cuarta prueba**
  - **Secuencial:** 0.000673 seg
  - **Bruteforce:** 0.003825 seg
- **Quinta prueba**
  - **Secuencial:** 0.006445 seg
  - **Bruteforce:** 0.010703 seg
- **Sexta prueba**
  - **Secuencial:** 0.030854 seg
  - **Bruteforce:** 0.024415 seg
- **Séptima prueba**
  - **Secuencial:** 0.552926 seg
  - **Bruteforce:** 0.353246 seg
- **Octava prueba**
  - **Secuencial:** 5.581149 seg
  - **Bruteforce:** 3.517776 seg
- **Novena prueba**
  - **Secuencial:** 44.401772 seg
  - **Bruteforce:** 30.299757 seg
- **Décima prueba**
  - **Secuencial:** 510.484108 sg
  - **Bruteforce:** 349.390802 sg

## SpeedUps

Utilizando la fórmula de speedup, que indica S = Ts/Tp. Donde la S es, la escalabilidad lineal. Ts es el tiempo en serie (sin paralelización). Y Tp es el tiempo en paralelo (usando N procesadores). Para este proyecto se utilizaron 4 procesadores.

- **Primer prueba**
  - **Speedup:** 0.014662
- **Segunda prueba**
  - **Speedup:** 0.009841
- **Tercer prueba**
  - **Speedup:** 0.021656
- **Cuarta prueba**
  - **Speedup:** 0.175948
- **Quinta prueba**
  - **Speedup:** 0.602168
- **Sexta prueba**
  - **Speedup:** 1.263692
- **Séptima prueba**
  - **Speedup:** 1.565187
- **Octava prueba**
  - **Speedup:** 1.587254
- **Novena prueba**
  - **Speedup:** 1.465682
- **Décima prueba**
  - **Speedup:** 1.461030

En base a los resultados obtenidos en estas pruebas podemos deducir que en las primeras cinco pruebas, los speedups son menores a 1, lo que significa que el tiempo de ejecución en paralelo es mayor que el tiempo de la versión secuencial. Y de las pruebas sexta a décima tenemos que los speedups son mayores a 1, lo que indica que la versión paralela supera a la versión secuencial, haciéndose más eficiente conforme la tarea se vuelve más grande y compleja. Esto muestra que, para tareas más grandes, la paralelización comienza a compensar la sobrecarga de coordinación entre los procesos.

# Eficiencia

- **Primer prueba**
  - **Eficiencia:** 0.003665
- **Segunda prueba**
  - **Eficiencia:** 0.002460
- **Tercer prueba**
  - **Eficiencia:** 0.005414
- **Cuarta prueba**
  - **Eficiencia:** 0.043987
- **Quinta prueba**
  - **Eficiencia:** 0.150542
- **Sexta prueba**
  - **Eficiencia:** 0.315933
- **Séptima prueba**
  - **Eficiencia:** 0.391318
- **Octava prueba**
  - **Eficiencia:** 0.396639
- **Novena prueba**
  - **Eficiencia:** 0.366354
- **Décima prueba**
  - **Eficiencia:** 0.365267

Para el caso de la eficiencia, la fórmula utiliza los datos de los speedup. Como era de esperar, los primeros speedups no fueron beneficiosos, aunado a lo anterior podemos inferir lo siguiente.

Las primeras cinco pruebas tienen eficiencias muy bajas (menores a 0.1). Esto indica que la mayoría del tiempo de ejecución en paralelo se consume en la coordinación entre los procesadores en lugar de en el procesamiento efectivo. Pero a partir de la sexta prueba, la eficiencia comienza a mejorar (superando el 0.3), lo cual sugiere que, para tareas más grandes, la paralelización empieza a ser más efectiva, con una mejor utilización de los procesadores. Aunque la eficiencia mejora en las últimas pruebas, sigue siendo inferior a 1. Esto significa que la escalabilidad no es lineal y que hay pérdidas de eficiencia debido a la sobrecarga de la comunicación entre procesos y la coordinación de la tarea paralelizada.

# Parte B

En esta parte B se buscaron alternativas paralelas para poder descifrar los textos, donde las seleccionadas fueron jump search y fb.
Para este primer algoritmo utilizamos el método de Jump Search para optimizar la búsqueda. El código utiliza la programación paralela con MPI para dividir la carga entre múltiples procesos.

- **Cifrado, descifrado y lectura:** Las funciones encrypt y decrypt emplean OpenSSL para cifrar y descifrar bloques de 8 bytes utilizando una clave DES proporcionada. Y la función retrieveText carga el contenido del archivo de texto y lo almacena en memoria
- **Distribución de rango de búsqueda:** El programa distribuye el rango de posibles claves DES (0 a 2^56 −1) entre los distintos procesos MPI, dividiendo de manera equitativa el rango de búsqueda de cada nodo para acelerar el proceso.
- **Implementación de Jump Search:** Se emplea un método de Jump Search para optimizar la búsqueda de la clave. Cada proceso realiza saltos de tamaño fijo (por ejemplo, 1000), y si no encuentra la clave en el salto, realiza una búsqueda secuencial dentro del intervalo.
- **Sincronización de procesos:** Los nodos utilizan MPI_Irecv y MPI_Send para comunicarse cuando una clave es encontrada, lo que detiene la búsqueda en los demás procesos y sincroniza la clave encontrada.
- **Ejecución y validación:** El programa toma como argumento la clave de cifrado y el archivo de entrada. Luego cifra el contenido del archivo y ejecuta el ataque de fuerza bruta.

Para el segundo algoritmo utilizamos un enfoque de programación paralela con MPI para dividir la carga de búsqueda de la clave entre múltiples procesos.

- **Cifrado y descifrado y carga del texto:** Las funciones encrypt y decrypt emplean OpenSSL para cifrar y descifrar bloques de 8 bytes con una clave DES proporcionada. Y la función loadTextFromFile carga el contenido de un archivo de texto y lo almacena en memoria.
- **Distribución del rango de búsqueda:** El programa divide el rango de posibles claves DES (0 a 2^56 −1) entre los distintos procesos MPI, asignando a cada proceso un rango específico para realizar la búsqueda de la clave, acelerando así el proceso de ataque de fuerza bruta.
- **Prueba de claves:** La función tryKey intenta descifrar el texto utilizando una clave específica y verifica si contiene la cadena "hello".
- **Sincronización de procesos:** El uso de MPI_Allreduce permite a todos los procesos sincronizarse cuando uno de ellos encuentra la clave correcta, deteniendo la búsqueda en los demás nodos para evitar trabajo innecesario.
- **Ejecución y validación:** En esta sección realiza lo mismo que en jump search.

# Prueba de códigos

**Bitácora**

## Jump Search

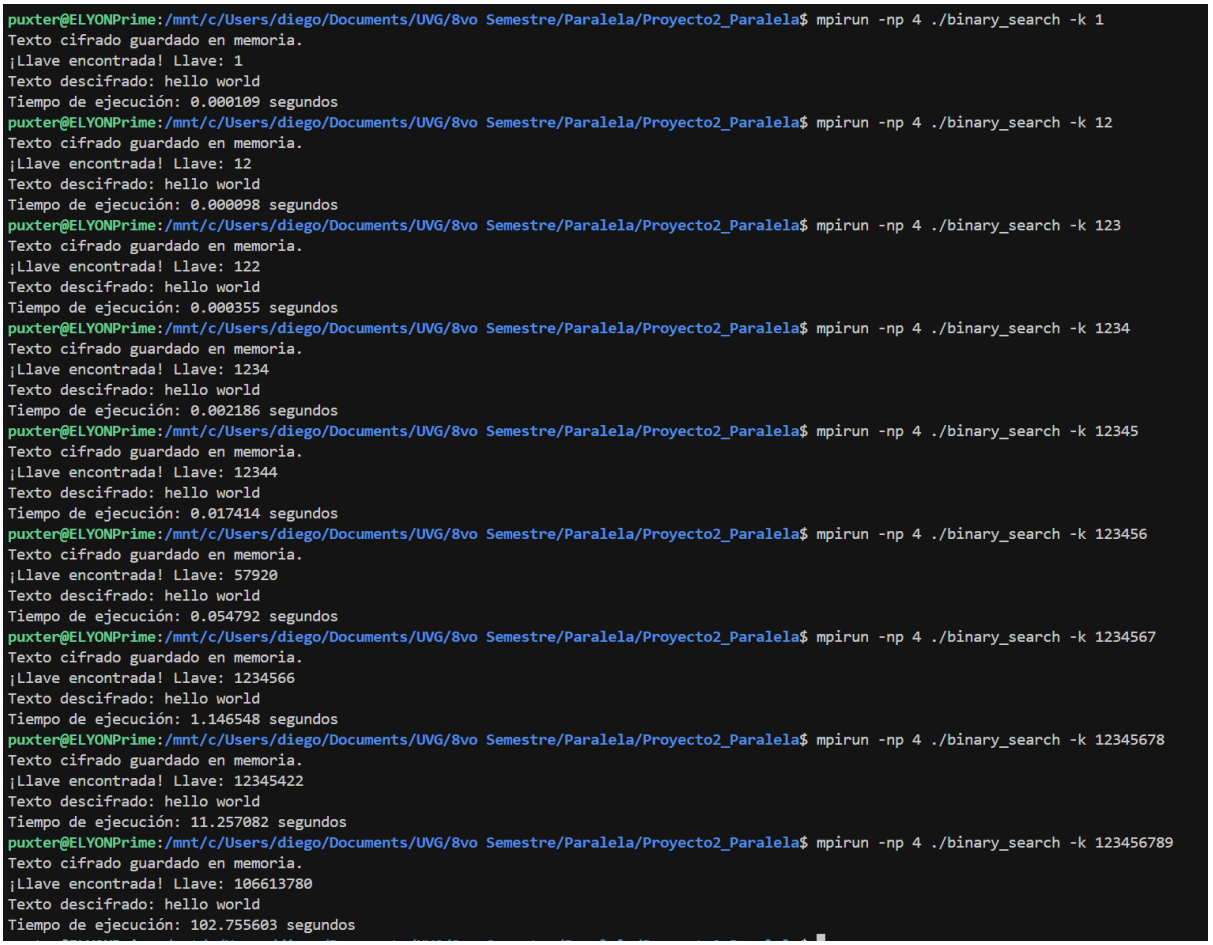
Para estas pruebas se realizó la ejecución de los mismos inputs con las mismas llaves en ambos algoritmos. El input utilizado fue “hello world” mientras que a lo largo de las pruebas las llaves fueron cambiando entre 1 dígito y 9 dígitos en cada prueba respectivamente.

```
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 1
Texto a cifrar: hello world
Texto cifrado en hexadecimal: def3f8ea781a124161c2ea
Clave encontrada: 0
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.002157 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 12
Texto a cifrar: hello world
Texto cifrado en hexadecimal: c3a7258651fdf39e89fa4d
Clave encontrada: 12
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.005364 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 123
Texto a cifrar: hello world
Texto cifrado en hexadecimal: a7b02455b0ef4e87267e60
Clave encontrada: 122
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.001784 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 1234
Texto a cifrar: hello world
Texto cifrado en hexadecimal: f270ac812b0d245bdf339d
Clave encontrada: 1234
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.016461 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 12345
Texto a cifrar: hello world
Texto cifrado en hexadecimal: 0542f6f092496d4afd7644
Clave encontrada: 12344
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.028426 segundos
```

```
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 123456
Texto a cifrar: hello world
Texto cifrado en hexadecimal: d07e87729f51057bb71999
Clave encontrada: 57920
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.022309 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 1234567
Texto a cifrar: hello world
Texto cifrado en hexadecimal: 0cb00bbd629dcf914208d0
Clave encontrada: 1234566
Texto descifrado con clave: hello world
Tiempo de ejecución: 0.360810 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 12345678
Texto a cifrar: hello world
Texto cifrado en hexadecimal: b2ae43f3fcbc17c519a64e
Clave encontrada: 12345422
Texto descifrado con clave: hello world
Tiempo de ejecución: 3.537902 segundos
puxter@ELYONPrime:/mnt/c/Users/diego/Documents/UVG/8vo Semestre/Paralela/Proyecto2_Paralela$ mpirun -np 4 ./jump_search -k 123456789
Texto a cifrar: hello world
Texto cifrado en hexadecimal: fda15392d2f7fb6dcc7043
Clave encontrada: 106613780
Texto descifrado con clave: hello world
Tiempo de ejecución: 30.469345 segundos
```

En esta imagen podemos observar 9 pruebas tanto en el algoritmo secuencial como en el jump search. Donde utilizamos los siguientes datos para realizar las pruebas. **Input:** ***“Hello World”*** y las **keys usadas**, son las siguientes: ***1, 12, 123, 1234, 1345, 123456, 1234567, 12345678, 123456789.***

Para estas pruebas se realizó la ejecución de los mismos inputs con las mismas llaves en ambos algoritmos. El input utilizado fue “hello world” mientras que a lo largo de las pruebas las llaves fueron cambiando entre 1 dígito y 9 dígitos en cada prueba respectivamente.




En esta imagen podemos observar 9 pruebas tanto en el algoritmo secuencial como en el jump search. Donde utilizamos los siguientes datos para realizar las pruebas. **Input: *“Hello World”*** y las **keys usadas**, son las siguientes: ***1, 12, 123, 1234, 1345, 123456, 1234567, 12345678, 123456789.***

## Resultados Jump Search

- **Primera prueba**
  - **Secuencial:** 0.000039 seg
  - **Paralelo:** 0.002157 seg
- **Segunda prueba**
  - **Secuencial:** 0.000047 seg
  - **Paralelo:** 0.005364 seg
- **Tercera prueba**
  - **Secuencial:** 0.000091 seg
  - **Paralelo:** 0.001784 seg
- **Cuarta prueba**
  - **Secuencial:** 0.000673 seg
  - **Paralelo:** 0.016461 seg
- **Quinta prueba**
  - **Secuencial:** 0.006445 seg
  - **Paralelo:** 0.028426 seg
- **Sexta prueba**
  - **Secuencial:** 0.030854 seg
  - **Paralelo:** 0.022399 seg
- **Séptima prueba**
  - **Secuencial:** 0.552926 seg
  - **Paralelo:** 0.360810 seg
- **Octava prueba**
  - **Secuencial:** 5.581149 seg
  - **Paralelo:** 3.537902 seg
- **Novena prueba**
  - **Secuencial:** 44.401772 seg
  - **Paralelo:** 30.469345 seg

## Resultados Paralela 2

- **Primera prueba**
  - **Secuencial:** 0.000039 seg
  - **Paralelo:** 0.0000109 seg
- **Segunda prueba**
  - **Secuencial:** 0.000047 seg
  - **Paralelo:** 0.000098 seg
- **Tercera prueba**
  - **Secuencial:** 0.000091 seg
  - **Paralelo:** 0.000355 seg
- **Cuarta prueba**
  - **Secuencial:** 0.000673 seg
  - **Paralelo:** 0.002186 seg
- **Quinta prueba**
  - **Secuencial:** 0.006445 seg
  - **Paralelo:** 0.017414 seg
- **Sexta prueba**
  - **Secuencial:** 0.030854 seg
  - **Paralelo:** 0.054792 seg
- **Séptima prueba**
  - **Secuencial:** 0.552926 seg
  - **Paralelo:** 1.146548 seg
- **Octava prueba**
  - **Secuencial:** 5.581149 seg
  - **Paralelo:** 11.257982 seg
- **Novena prueba**
  - **Secuencial:** 44.401772 seg
  - **Paralelo:** 102.755603 sg

## SpeedUps Jump Search

Utilizando la fórmula de speedup, que indica S = Ts/Tp. Donde la S es, la escalabilidad lineal. Ts es el tiempo en serie (sin paralelización). Y Tp es el tiempo en paralelo (usando N procesadores). Para este proyecto se utilizaron 4 procesadores.

- **Primera prueba**
  - **Speedup:** 0.018081
- **Segunda prueba**
  - **Speedup:** 0.008762
- **Tercera prueba**
  - **Speedup:** 0.051009
- **Cuarta prueba**
  - **Speedup:** 0.040885
- **Quinta prueba**
  - **Speedup:** 0.226729
- **Sexta prueba**
  - **Speedup:** 1.377480
- **Séptima prueba**
  - **Speedup:** 1.532333
- **Octava prueba**
  - **Speedup:** 1.578104
- **Novena prueba**
  - **Speedup:** 1.456950

## SpeedUps  Paralela 2

Utilizando la fórmula de speedup, que indica S = Ts/Tp. Donde la S es, la escalabilidad lineal. Ts es el tiempo en serie (sin paralelización). Y Tp es el tiempo en paralelo (usando N procesadores). Para este proyecto se utilizaron 4 procesadores.

- **Primera prueba**
  - **Speedup:** 3.5779817
- **Segunda prueba**
  - **Speedup:** 0.479592
- **Tercera prueba**
  - **Speedup:** 0.256338
- **Cuarta prueba**
  - **Speedup:** 0.307868
- **Quinta prueba**
  - **Speedup:** 0.370105
- **Sexta prueba**
  - **Speedup:** 0.563111
- **Séptima prueba**
  - **Speedup:** 0.482253
- **Octava prueba**
  - **Speedup:** 0.495750
- **Novena prueba**
  - **Speedup:** 0.432110

## Eficiencia Jump Search

- **Primera prueba**
  - **Eficiencia:** 0.004520
- **Segunda prueba**
  - **Eficiencia:** 0.002191
- **Tercera prueba**
  - **Eficiencia:** 0.012752
- **Cuarta prueba**
  - **Eficiencia:** 0.010221
- **Quinta prueba**
  - **Eficiencia:** 0.056682
- **Sexta prueba**
  - **Eficiencia:** 0.344370
- **Séptima prueba**
  - **Eficiencia:** 0.383083
- **Octava prueba**
  - **Eficiencia:** 0.394526
- **Novena prueba**
  - **Eficiencia:** 0.364237

## Eficiencia  Paralela 2

- **Primera prueba**
  - **Eficiencia:** 0.513158
- **Segunda prueba**
  - **Eficiencia:** 0.119898
- **Tercera prueba**
  - **Eficiencia:** 0.064085
- **Cuarta prueba**
  - **Eficiencia:** 0.076967
- **Quinta prueba**
  - **Eficiencia:** 0.092526
- **Sexta prueba**
  - **Eficiencia:** 0.140778
- **Séptima prueba**
  - **Eficiencia:** 0.120563
- **Octava prueba**
  - **Eficiencia:** 0.123938
- **Novena prueba**
  - **Eficiencia:** 0.108027

## Conclusión

### Jump Search

**Speedup**: En las primeras pruebas, el speedup de Jump Search es muy bajo, por debajo de 0.1, lo que significa que la versión paralela es más lenta que la secuencial. Sin embargo, a partir de la sexta prueba, el speedup supera el valor de 1, lo que indica una mejora en el rendimiento gracias a la paralelización.
**Eficiencia:** La eficiencia de Jump Search es consistentemente baja, especialmente en las primeras pruebas, con valores cercanos a cero. Esto sugiere que no se está utilizando de manera efectiva el poder de los 4 procesadores.

### Paralela 2

**Speedup:** Paralela 2 muestra un rendimiento mucho mejor en la primera prueba, con un speedup de 3.58, indicando una gran mejora en el tiempo de ejecución paralelo respecto al secuencial. Sin embargo, en pruebas posteriores, el speedup desciende por debajo de 1, lo que nos dice que la ganancia de rendimiento no es tan significativa en algunos casos.

**Eficiencia:** A pesar de tener un speedup más alto inicialmente, la eficiencia de Paralela 2 decrece rápidamente, con valores que caen por debajo de 0.15 en la mayoría de las pruebas. Esto nos dice que, aunque el tiempo paralelo puede ser menor, el uso de los 4 procesadores no es óptimo, lo que implica una distribución de carga que podría mejorarse.