Universidad del Valle De Guatemala

Facultad de Ingeniería

Computación Paralela y Distribuida

Proyecto 2

Programación Paralela con OpenMPI

Elean Rivas 19062
Diego Ruiz 18761

Guatemala, octubre de 2023

Introducción

En informática, el paso de mensajes es un modelo de programación ampliamente utilizado en el software moderno. Sus aplicaciones cubren un área grande y se puede utilizar para garantizar que los diferentes objetos que componen un programa de computadora puedan trabajar sincronizados entre sí, para permitir una sola tarea sincronizada entre varias computadoras. Es uno de los conceptos clave de los paradigmas de concurrencia, programación distribuida y programación orientada a objetos.

De manera abstracta, un mensaje es información que un proceso emisor envía a un receptor (que puede ser otro proceso, un agente o un objeto). El modelo de paso de mensajes es un modelo que define los métodos y funciones para realizar la entrega de un mensaje desde un proceso de envío a un receptor. Esto supone un enfoque opuesto al paradigma tradicional, donde los procesos, funciones y subrutinas solo pueden llamarse directamente por su nombre.

MPI (Message Passing Interface) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores. El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua. Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

Con este proyecto se pretende demostrar la implementación de Open MPI, esta es una implementación de interfaz de paso de mensajes de código abierto desarrollada y mantenida por un consorcio de socios académicos, de investigación y de la industria. Por lo tanto, Open MPI puede combinar la experiencia, las tecnologías y los recursos de toda la comunidad informática de alto rendimiento para crear la mejor biblioteca MPI disponible. Open MPI ofrece ventajas para los proveedores de sistemas y software, desarrolladores de aplicaciones e investigadores de informática.

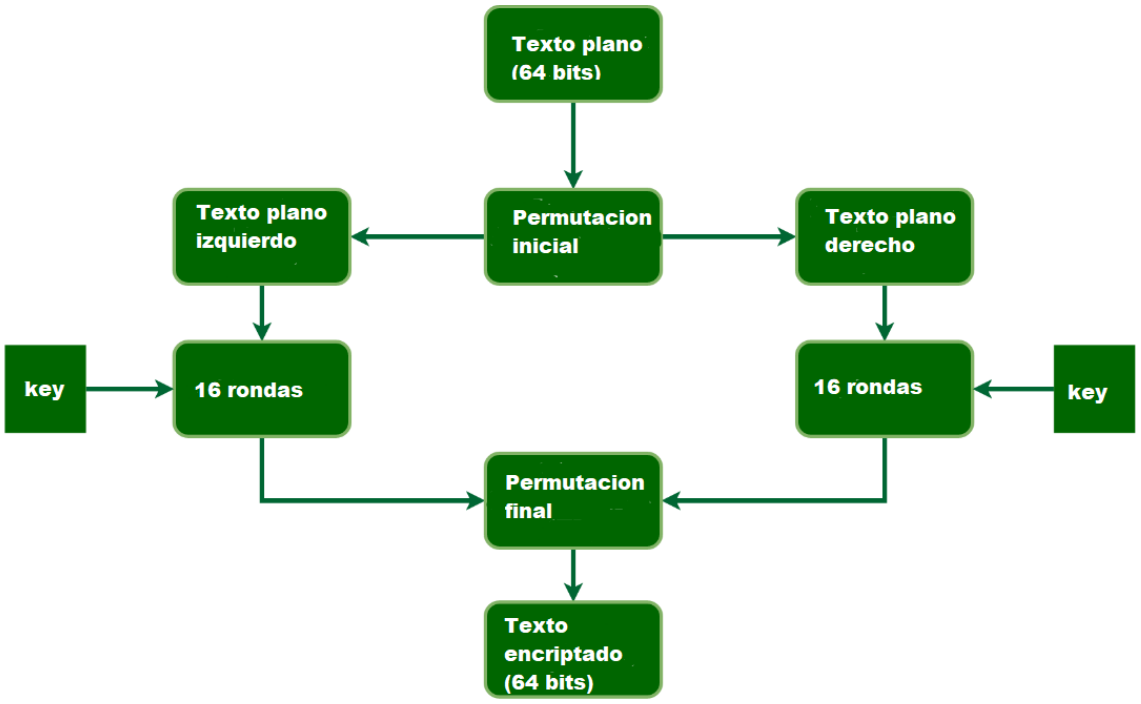
DES

Des o Data Encryption Standard, es un una vieja práctica de cifrado de información, en esta se cifra la información en bloques de 56 bits pues el octavo bit de cada combinación es eliminado para generar la clave de 56 bits, es decir la cadena original consta de 64 bits donde el bit 8, 16, 24, 32, 40, 48, 56 y 64 son descartados.

El DES se basa en dos conceptos básicos, sustitución y desplazamiento. El DES consta de 16 pasos, cada uno de los cuales se llama una ronda. Cada ronda realiza los pasos de sustitución y transposición. Ahora, discutamos los pasos a un nivel más general en el DES.

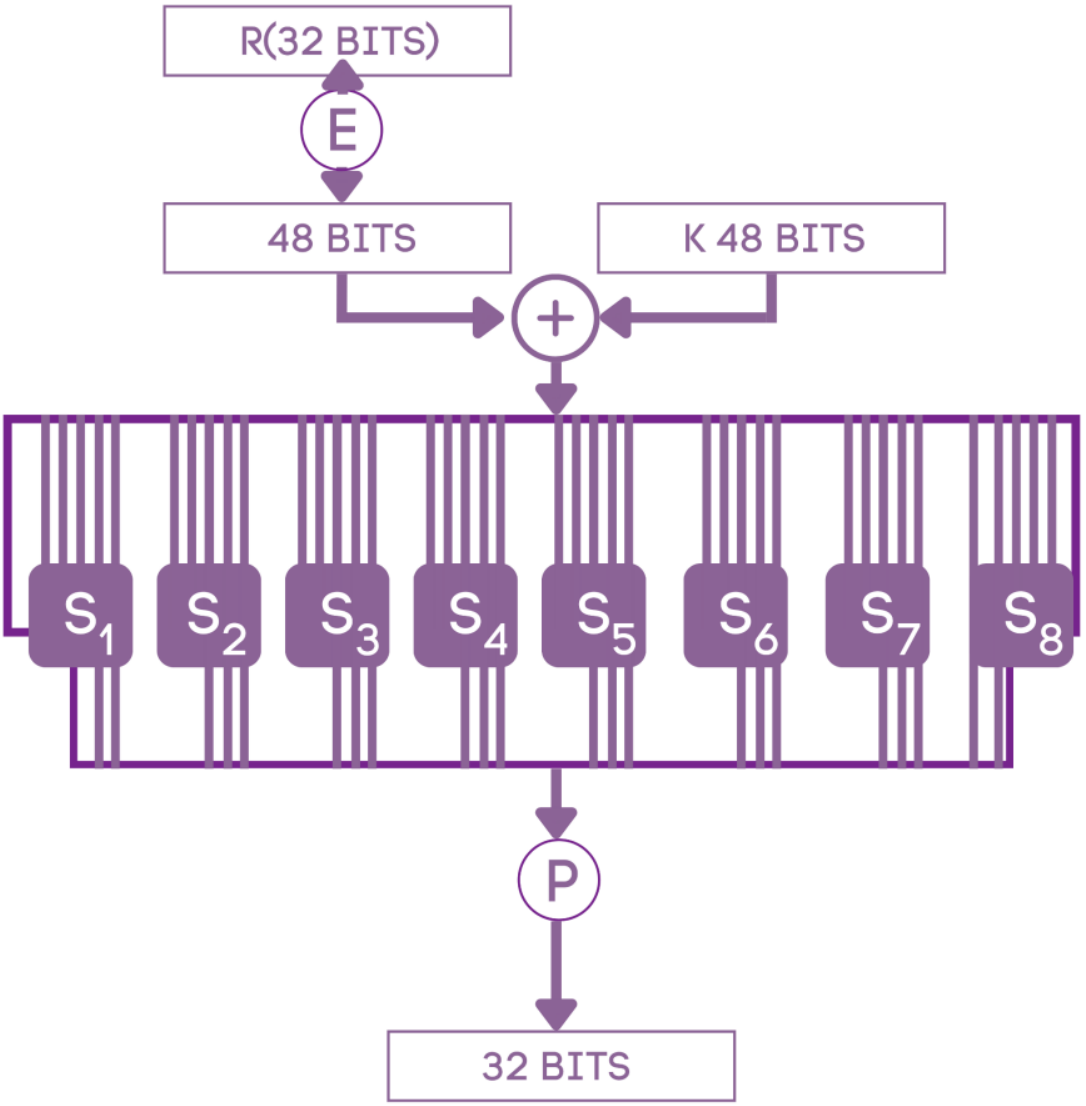
En el primer paso, el bloque de texto plano de 64 bits se entrega a una función de Permutación Inicial (IP).

La permutación inicial se realiza en el texto plano.
A continuación, la permutación inicial (IP) produce dos mitades del bloque permutado; llamadas Texto Plano Izquierdo (LPT) y Texto Plano Derecho (RPT).
Ahora, cada LPT y RPT pasan por 16 rondas del proceso de cifrado.
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 un cifrado de 64 bits.



DES, es característico por ser el primer proceso de encriptación comercial usado, hoy día no se usa pues es muy vulnerable precisamente a algoritmos como bruteforce, que son capaces de generar muchas combinaciones para obtener la información, por tanto bruteforce es el algoritmo perfecto para paralelizar y optimizar si se quiere romper con el paradigma de encriptación DES

Función cypher:



Rutinas:

main: Función principal del programa.
Inicializa MPI y determina el rango (rank) y el tamaño (size) del comunicador.
El proceso de rango 0 lee un archivo "message.txt" y verifica si contiene la cadena "combinaciones".
Cifra el texto usando AES con una clave de ejemplo y transmite el texto cifrado a todos los otros procesos.
Cada proceso intenta descifrar el texto usando diferentes claves, distribuyendo la carga de trabajo entre ellos.
Si un proceso encuentra la clave correcta, imprime la clave.
El proceso de rango 0 mide e imprime el tiempo total de ejecución.

encrypt: Cifra el texto plano usando AES.
Configura la clave AES para cifrar y luego cifra el texto plano.

decrypt: Descifra el texto cifrado usando AES.Configura la clave AES para descifrar y luego descifra el texto cifrado.

tryKey: Intenta descifrar el texto cifrado con una clave y verifica si el texto descifrado contiene una cadena específica.
Descifra el texto cifrado usando una clave.
Comprueba si el texto descifrado contiene la cadena especificada.

my_memcpy: Una función de copia de memoria personalizada.
Copia 'n' bytes desde la fuente 'src' a la destinación 'dest'.

my_strstr: Una función que busca una cadena (needle) dentro de otra cadena (haystack).
Itera a través de 'haystack' y verifica si 'needle' está presente en cada posición.
Si encuentra la cadena 'needle', devuelve 1; de lo contrario, devuelve 0.

Primitivas de MPI:

- MPI_SEND:

    Realiza el envío de un mensaje de un proceso fuente a otro destino.
    Parametros
    - buf Dirección inicial del buffer de envío. Esto significa que requiere un puntero. Si solo se pretende enviar un elemento, se puede enviar el puntero a este (&elemento).
    - count Numero de elementos a enviar (Debe ser un entero no negativo).
    - datatype Tipo de dato de cada elemento que se va a enviar. Acepta constantes definidas por MPI (Véase el apartado de constantes). Por ejemplo MPI_INT.
    - dest Rango del proceso destino. (Véase MPI_Comm_rank)
    - tag Entero que representa la etiqueta del mensaje. El significado de la etiqueta queda en manos del usuario, durante el proceso de envío no es modificado.
    - comm Comunicador utilizado para la comunicación.
    Descripción
    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:

    Bloquea al proceso hasta que termine la operación de envío o de recibo especificada.
    Request: Dato de tipo MPI_Request, en el se guarda una etiqueta que identifica una operación no bloqueante. (Véase por ejemplo MPI_Isend).
    Status: Objeto de tipo MPI_Status, contiene datos relevantes sobre el mensaje (como son el origen (MPI_SOURCE), la etiqueta (MPI_TAG) y el

tamaño (size)).
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.

Posibles Errores

MPI_SUCCESS → No ha habido error, la función MPI se ha realizado satisfactoriamente.
MPI_ERR_REQUEST → Objeto no válido de MPI_Request. Generalmente sucede porque tiene valor nulo.
MPI_ERR_ARG Ha recibido un argumento incorrecto no especificado por un error específico de clase (como MPI_ERR_RANK).

- MPI_Irecv:

- Operación de recibo de mensaje no bloqueante.
- 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).
Parámetros de entrada
count: Entero que indica el número máximo de elementos que se espera recibir en el buffer de entrada.
datatype: Tipo de dato de cada elemento que se va a recibir. Acepta constantes definidas por MPI (Véase el apartado de constantes). Por ejemplo

MPI_INT.

source: Rango del proceso de origen esperado, sólo se recogen mensajes cuyo origen sea el especificado. Se acepta el valor MPI_ANY_SOURCE , el cual recoge de cualquier proceso origen. (Véase también MPI_Comm_rank)
tag: Entero que representa la etiqueta del mensaje. Solo se recogerá un mensaje con la etiqueta especificada. Se acepta el valor MPI_ANY_TAG , que recoge con cualquier etiqueta. El significado de la etiqueta queda en manos del usuario, durante el proceso de envío no es modificado.
comm: Comunicador utilizado para la comunicación. Solo se recogerán mensajes que han sido enviados por el comunicador seleccionado.
Parámetros de salida
buf → Buffer de entrada en el que se guarda el contenido del mensaje enviado. Recibe un puntero al comienzo del buffer.
request → Dato de tipo MPI_Request, en el se guarda una etiqueta que identifica la operación no bloqueante. Esto se utiliza para poder consultar, por ejemplo, si la operación ha finalizado, o poder esperar a que finalice en un momento dado. (Véase MPI_Wait).

Resultados:

Prueba 1:

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: key
Tiempo total de ejecución: 0.083212 segundos.
```

a : 123456L

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: 123456L
Tiempo total de ejecución: 0.061398 segundos.
```

b: 18014398509481983L

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: 180143985094819 8@A5@@@ @@d@
```

c: 18014398509481984L

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: 180143985094819 8@A5@@@ @@d@
Tiempo total de ejecución: 1.002319 segundos.
```

d: El tiempo de ejecución en un ataque de fuerza bruta no está directamente relacionado con el valor de la clave en sí, sino con su posición dentro del espacio de claves que se está explorando. Dada esta situación, ya que las llaves se encuentran en la mitad del rango de ejecución de la exploración de los bits el tiempo tiende a ser mucho mayor

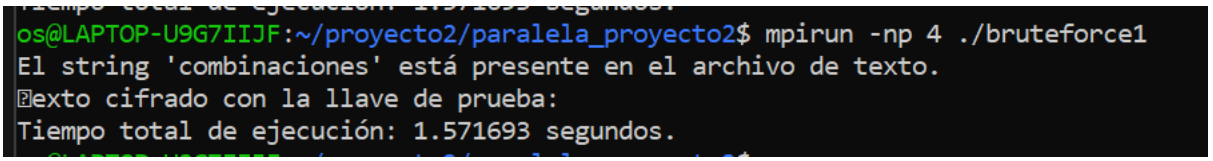
e. Una llave fácil de encontrar, por ejemplo, con valor de (2^56) / 2 + 1

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: @
Tiempo total de ejecución: 0.100051 segundos.
```

f. Una llave medianamente difícil de encontrar, por ejemplo, con valor de (2^56) / 2 + (2^56) / 8

```
os@LAPTOP-U9G7IIJF:~/proyecto2/paralela_proyecto2$ mpirun -np 4 ./bruteforce1
El string 'combinaciones' está presente en el archivo de texto.
Texto cifrado con la llave de prueba: `
Tiempo total de ejecución: 0.192451 segundos.
```

g. Una llave difícil de encontrar, por ejemplo, con valor de (2^56) / 7 + (2^56) / 13 aproximados al entero superior



Demostración de la ecuación:

$E[tPar(n, k)] = \sum \Pi$
$i\ xipi = 255$
$n + 1/2$

Donde:
$E[tPar(n, k)]$, es el tiempo esperado de búsqueda paralela para un problema con n elementos y k procesos.
xi, es el tiempo que se necesita para probar la clave i.
pi, es la probabilidad de que la clave sea i
$n + 1/2$, el tiempo esperado para buscar un elemento en una lista no ordenada
Distribución uniforme de claves: Suponemos que cualquier clave es igualmente probable. Por lo tanto, la probabilidad pi es $1/n$ para todas las claves en un espacio de clave de tamaño n en este caso n=256

División igual del espacio de claves entre procesos: Si hay k procesos trabajando en paralelo, cada proceso verificará n/k claves.

El tiempo esperado para encontrar la clave correcta en un único proceso es el tiempo medio. Si la clave puede ser cualquiera de las n con igual probabilidad, el tiempo medio es probar la mitad de las claves. Por lo tanto, el tiempo esperado en un proceso sería $n + 1/2 * x$

Acercamientos naive:

Acercamiento 1:

```
funcion fuerzaBrutaParalelo(númeroDeProcesos, espacioDeClaves, mensajeCifrado):

    claveEncontrada = null
    tamañoPorProceso = espacioDeClaves / númeroDeProcesos

    paralelizar como proceso(p = 0 hasta númeroDeProcesos - 1):
        inicio = p * tamañoPorProceso
        final = inicio + tamañoPorProceso

        para clave en rango(inicio, final):
            if esLaClaveCorrecta(clave, mensajeCifrado):
                claveEncontrada = clave
                detener todos los procesos
                salir

    return claveEncontrada

funcion esLaClaveCorrecta(clave, mensajeCifrado):
```

```
    mensajeDescifrado = descifrar(mensajeCifrado, clave)
    if contieneTextoEspecifico(mensajeDescifrado):
        return verdadero
    else:
        return falso
```

Acercamiento 2:

```
const TAMAÑO_UNIDAD = ... // tamaño adecuado según experimentación

funcion fuerzaBrutaAdaptativo(númeroDeProcesos, espacioDeClaves, mensajeCifrado):

    claveEncontrada = null
    unidadesTotales = espacioDeClaves / TAMAÑO_UNIDAD
    unidadesProcesadas = 0

    paralelizar como proceso(p = 0 hasta númeroDeProcesos - 1):
        mientras unidadesProcesadas < unidadesTotales:
            unidadActual = obtenerNuevaUnidadAtomica(unidadesProcesadas)
            inicio = unidadActual * TAMAÑO_UNIDAD
            final = inicio + TAMAÑO_UNIDAD

            para clave en rango(inicio, final):
                if esLaClaveCorrecta(clave, mensajeCifrado):
                    claveEncontrada = clave
                    detener todos los procesos
                    salir

    return claveEncontrada

funcion obtenerNuevaUnidadAtomica(unidadesProcesadas):
    // Esta función incrementa atomica y seguramente el contador y retorna la unidad actual
    unidadActual = unidadesProcesadas
    unidadesProcesadas += 1
    return unidadActual

funcion esLaClaveCorrecta(clave, mensajeCifrado):
    mensajeDescifrado = descifrar(mensajeCifrado, clave)
    if contieneTextoEspecifico(mensajeDescifrado):
        return verdadero
    else:
        return falso
```

$$E[Tadapt] = (\frac{k}{2U})x\frac{Tunit+TComm}{n}$$

n: Número de procesos trabajando en paralelo.
K: Tamaño total del espacio de claves.
U: Tamaño de una unidad de trabajo (cantidad de claves que un proceso intenta en una sola iteración).

T unit: Tiempo promedio que tarda un proceso en verificar una unidad de trabajo (es decir, probar U claves).

T comm: Tiempo promedio que tarda un proceso en solicitar y recibir una nueva unidad de trabajo.

Referencias:

https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard
https://www.tutorialspoint.com/cryptography/data_encryption_standard.html

Link a GitHub:

https://github.com/GitElean/paralela_proyecto2