1. Antecedentes

MPI es el estándar para el modelo paralelo de memoria distribuida. Debido a su interoperabilidad y portabilidad, MPI nos permite interconectar máquinas para crear conjuntos computacionales homogéneos y heterogéneos. El paradigma de intercambio de mensajes nos permite interconectar máquinas en área local sin depender de la topología de la red de interconexión. Esto hace que MPI sea muy escalable.

2. Objetivos y Competencias

- Implementa y diseña programas para la paralelización de procesos con memoria distribuida usando OpenMPI.
- Optimizar el uso de recursos distribuidos y mejorar el speedup de un programa paralelo.
- Descubrir la llave privada usada para cifrar un texto, usando el método de fuerza bruta (brute force).
- Comprender y Analizar el comportamiento del speedup de forma estadística.

3. Descripción

En equipos de 3 integrantes, deben utilizar MPI para diseñar un programa que encuentre la llave privada con la que fue cifrado un texto plano. La búsqueda se hará probando todas las posibles combinaciones de llaves, hasta encontrar una que descifra el texto (fuerza bruta). Se sabrá si logro descifrar el texto correctamente validando si el mismo contiene como substring una palabra/frase clave de búsqueda, la cual se sabe a priori (por ejemplo, si el texto cifrado fuera el presente párrafo, ‘combinaciones’ podría ser una buena palabra clave).

Es importante elegir una frase clave a buscar de buen tamaño, para asegurar que sea muy improbable que esa clave suceda de forma aleatoria al descifrar (erróneamente) el texto.

Además del cifrado, descifrado y paralelización con OpenMPI, el equipo debe realizar un análisis exhaustivo de speedups, performance y profiling de procesos, para entender y optimizar la distribución del espacio de búsqueda de las llaves. Deberá ponerse especial atención al impacto que tiene la llave solución en el performance y tiempos/speedups aparentes del algoritmo, así como la consistencia de este con distintas llaves.

4. Actividades

Trabajando con el código base *bruteforceNaive.c*, realice las siguientes tareas, evidenciando su procedimiento y resultados en su reporte escrito:

Parte A

1. Investigue sobre DES y describa los pasos requeridos para cifrar/descifrar un texto. Incluya esto en su reporte.
2. Dibuje un diagrama de flujo describiendo el algoritmo DES
3. Luego de trabajar un poco con DES, haremos una prueba de código como introducción y base para el resto del desarrollo. Haga funcionar el programa bruteforce.c. Es probable que necesite una biblioteca que reemplace a "rpc/des_crypt.h" en caso su computadora/instalación/sistema operativo lo requiera, para ello puede utilizar cualquier librería que desee/encuentre.
4. Una vez funcionando su programa base, explique, mediante diagramas, texto, dibujos, etc., como funcionan las rutinas (o la equivalente si uso otra librería en caso de decrypt/encrypt):
    a. decrypt (key, *ciph, len) y encrypt (key, *ciph, len)
    b. tryKey (key, *ciph, len)
    c. memcpy
    d. strstr
5. Describa y explique el uso y flujo de comunicación de las primitivas de MPI:
    a. MPI_Recv
    b. MPI_Send
    c. MPI_Wait

Parte B

1. Ya que estamos familiarizados con el temario, vamos a analizar el problema más a fondo. Modifique su programa para que cifre un texto cargado desde un archivo (.txt) usando una llave privada arbitraria (como parámetro). Muestra una captura de pantalla evidenciando que puede cifrar y descifrar un texto sencillo (una oración) con una clave sencilla (por ejemplo 42).

2. Una vez listo el paso anterior, proceder a hacer las siguientes pruebas, evidenciando todo en su reporte. **Para todas ellas utilice 4 procesos (-np 4).** El texto a cifrar/descifrar: *"Esta es una prueba de proyecto 2"*. La palabra clave a buscar es: *"es una prueba de":*
    a. Mida el tiempo de ejecución en romper el código usando la llave 123456L

b. Mida el tiempo de ejecución en romper el código usando la llave: $(2^{56}/4)$. O sea, 18014398509481983L. [spoiler: se tardará mucho, si es que termina, no se ofusquen si no termina].
c. Mida el tiempo de ejecución en romper el código usando la llave: $(2^{56}/4) + 1$. O sea, 18014398509481984L.
d. Reflexione lo observado y el comportamiento del tiempo en función de la llave.

Un aspecto interesante del presente problema/temario es que los speedups y tiempos paralelos obtenidos son sumamente inconsistentes y dependientes de la llave elegida. Ello debido a que estamos recorriendo el espacio de datos de forma "naive" (incremental y en orden y dividiendo equitativamente los segmentos).

Por ejemplo (tomando como base el inciso anterior), asumiendo 4 procesos, y eligiendo como llave (2^56)/4 +1, podemos ver que el proceso #2 (al que se le asigna el segundo segmento de datos) encontrará la llave en el primer intento (Tpar=1 iter). Un algoritmo secuencial le hubiera tomado (2^56)/4 +1 iteraciones, por lo que el speedup es de (2^56)/4 +1 (algo sumamente alto y que nos puede dar falsa confianza en nuestro algoritmo).

Caso contrario, si la llave fuera (2^56)/4 el proceso #1 la encontraría en su última iteración. Podemos notar que un programa secuencial le tomaría la misma cantidad de iteraciones encontrar esa llave, por lo que no obtendremos speedup alguno en este caso (speedup=1).

Para poder ver y comprender tal fenómeno podemos realizar cambios a la llave privada y analizar el desempeño en función de ellas; asumiendo 4 procesos (ojo, adaptar dependiendo de los procesos que usen):

d. Una llave fácil de encontrar, por ejemplo, con valor de (2^56) / 2 + 1
e. Una llave medianamente difícil de encontrar, por ejemplo, con valor de (2^56) / 2 + (2^56) / 8
f. Una llave difícil de encontrar, por ejemplo, con valor de (2^56) / 7 + (2^56) / 13 aproximados al entero superior

En este punto nos podemos percatar de algo: **el tiempo paralelo es una función del número de procesos y la llave utilizada - *tPar(n,k)*.** De ello sigue una observación importante: nos interesa saber el valor esperado de tPar para poder tener una mejor medida del desempeño del algoritmo. Nótese que cada posible llave [0-2^56] tiene la misma probabilidad de ser elegida (1/ 2^56). Se puede demostrar que el valor esperado para el approach "naive" mencionado es (tip: graficar ayuda, formula de Gauss suma números consecutivos también):

$$\boldsymbol{E[tPar(n,k)] = \sum_i^{\Pi} x_i p_i = \frac{2^{55}}{n} + 1/2} \qquad \textbf{(1)}$$

6. (opcional, fuertemente sugerido ya que les tocará hacer el proceso 2+ veces en siguientes pasos) Demuestre que el valor esperado para el tiempo paralelo es lo indicado en la Ecuación 1 del párrafo anterior.

7. Como podemos ver, el approach “naive” no es el mejor posible. Proponga, analice, e implemente 2 opciones alternativas al acercamiento “naive”. Tenga como objetivo en mente encontrar un algoritmo que tenga mejor “tiempo paralelo esperado” que la versión “naive” demostrada en ecuación (1). Para cada una no olvide:
    a. Describir el acercamiento propuesto, se puede apoyar con diagramas de flujo, pseudocódigo, o algoritmo descriptivo.
    b. Derivar el valor esperado de ***tPar(n,k)*** de ese acercamiento y compararlo con el del acercamiento “naive”. Además del valor esperado, discuta su procedimiento y razonamiento. Mencione cómo se comporta el speedup en este acercamiento.
    c. Implementelo en código y pruébelo con 3-4 llaves (fáciles, medianas, difíciles). Compare el tiempo medido con el tiempo pronosticado por su función ***tPar(n,k)***

5. Requisitos

Para la implementación de la solución paralela la búsqueda de una llave privada usada para cifrar un texto plano mediante DES, cada equipo debe cumplir con las siguientes condiciones (además de lo solicitado en el documento):

a. Código de autoría propia en C/C++
b. Historial del control de versiones del programa en git evidenciando un flujo de trabajo adecuado.
c. Uso de OpenMPI
d. Versión secuencial y paralela del algoritmo y sus acercamientos propuestos.
e. Las mediciones y elementos requeridos indicados en la sección 4 de Actividades.

6. Criterios diferenciadores (extra: hasta 35%)

- Hemos mencionado con anterioridad que MPI es el estándar de memoria distribuida, y que su facilidad de interconectar conjuntos computacionales en Red Local lo hace bastante escalable. Naturalmente, “red local” nos dice que puede ir más allá de limitarse a interconectar procesos en una misma máquina, permitiéndonos distribuir trabajo en otras máquinas, o incluso en clusters de máquinas.

  Se otorgará hasta un **20%** extra a los grupos que logren interconectar 2+ máquinas y correr el bruteforce en ellas a modo cluster MPI. Pueden usar Raspberry PI’s, las máquinas de sus compañeros de equipo, alguna otra máquina de su propiedad, etc (excepto virtualizar, claro). Pueden basarse o comenzar la configuración siguiendo el siguiente link: https://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/

- Es posible apalancarse de la memoria compartida utilizando OpenMP en conjunto con MPI. Se otorgará hasta un **10%** extra al grupo que incorpore OpenMP en su proyecto y que logre demostrar un speedup consistente con la versión sin OpenMP. (tip: Barlas da ejemplo de ello)

- Se otorgará hasta un **5%** extra al grupo que realice 3+ approaches adicionales al naive de ejemplo (o sea, dividir espacio equitativamente en n bloques, distribuirlos, y recorrer los sub-espacios incrementalmente), con sus análisis respectivos.

7. Evaluación

	Informe – 30 puntos	valor
1	Mínimo 1 página de antecedentes numéricos y conceptuales sobre DES, cifrado y bruteforce. El diagrama de flujo puede ser 1 página adicional	3
2	Formato según guía de informes UVG: carátula, índice, introducción, cuerpo, citas textuales / píe de página, conclusiones / recomendaciones, apéndice con material suplementario y al menos 3 citas bibliográficas relevantes y confiables	3
3	Discusión / Conclusión / recomendación – resumir los retos encontrados, lo aprendido y descubierto, y las soluciones para la implementación del temario de forma paralela con OpenMPI. El programa base funciona (con su propia librería DES o la del ejemplo) y recibe el texto como se pide.	5
4	Mínimo 2 acercamientos distintos al "naive" de ejemplo, como se solicita en Parte B Inciso 6 y 7. No olvide: • Describir el acercamiento propuesto, se puede apoyar con diagramas de flujo, pseudocódigo, o algoritmo descriptivo. • Derivar el valor esperado de ***tPar(n,k)*** de ese acercamiento y compararlo con el del acercamiento "naive". Además del valor esperado, discuta su procedimiento y razonamiento. Mencione cómo se comporta el speedup en este acercamiento. • Probarlo con 3-4 llaves (fáciles, medianas, difíciles). Compare el tiempo medido con el tiempo pronosticado por su función ***tPar(n,k)***	10
5	Anexo 1 – Catálogo de funciones y librerias • Entradas – nombres de variables, tipos y descripción de su uso • Salidas – nombres de variables, tipos y descripción de su uso • Descripción – Propósito de la función / clase / subrutina y descripción del funcionamiento	5
6	Anexo 2 – Bitácora de pruebas y speedups. Capturas de evidencia. (* Sin evidencia y los pasos solicitados no se califica el reporte.	4*

	Programa y presentación – 70 puntos	valor
1	Programa funcionando correctamente para la búsqueda mediante bruteforce de la llave privada usada para cifrar un texto. Incluyendo pero no limitado a: • Programación defensiva para evitar problemas en el intercambio de mensajes, en el ingreso de la ubicación del archivo o de la palabra clave. • Impresión de la llave posible, del nombre del archivo cifrado y de la palabra clave. • Pruebas y textos cifrados adecuados • Bitácora de pruebas para la medición de tiempos de búsqueda y de variación de tiempos con diferente número de procesos. • Mejora a la forma de dividir y recorrer el rango numérico para búsqueda de la llave • Acercamiento Naive funcional • Acercamientos (2+) alternativos funcionales • Manejo adecuado de memoria (malloc/free, new/delete, etc.) • Utilización de OpenMPI e intercambio de mensajes	60
2	Documentación y comentarios explicativos sobre las partes importantes del programa. Diseñe su propio estilo de documentación y úselo consistentemente en todo el programa	5
3	Programa ordenado, con nombres de variables de entrada y salida y nombres de rutinas adecuadamente identificadas (nombres nemotécnicos)	5

8. Entregar en Canvas

Entrega Final y presentación: (Miercoles 25 de octubre).

- **Material a entregar en *Canvas*:**
  a. Informe del proyecto de investigación en formato PDF, siguiendo las normas para informes UVG (caratula, indice, la estructura, etc.) en formato PDF, conteniendo:
     i. Antecedentes numéricos del tema asignado.
     ii. Diagrama de flujo de su programa.
     iii. Descripción de catálogo de las funciones desarrolladas para implementar el algoritmo de solución.
     iv. Retos encontrados para la implementación y conclusiones sobre el proceso de implementación paralela.
     v. Los demás requisitos mencionados en este documento.
  b. Código fuente funcional. **NO SE PERMITE SOLAMENTE EL LINK A SU REPOSITORIO, DEBEN SUBIR EL CÓDIGO. DE NO HACERLO PERDERÁN ESOS PUNTOS.**
  c. Link a repositorio.
  d. Cualquier material relacionado relevante o necesario para el correcto funcionamiento del código (makefiles, appfiles, hostfiles, etc.).