Proyecto #2 - Programación Paralela con MPI

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).

3. Descripción

En equipos de 3 integrantes, deben utilizar MPI para diseñar un programa que encuentre la llave privada con la que fué cifrado un texto plano. La búsqueda se hará probando todas las posibles combinaciones de llaves, hasta encontrar una que descifra el texto. 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).

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 speedups aparentes del algoritmo, así como la consistencia del mismo con distintas llaves.

4. Actividades

Trabajando con el código base bruteforce.c , realice las siguientes tareas:

1. Investigue sobre DES y describa los pasos requeridos para cifrar/descifrar un texto.
2. Dibuje un diagrama de flujo describiendo el algoritmo DES
3. Haga funcionar el programa bruteforce.c Es probable que necesite una biblioteca que reemplace a “rpc/des_crypt.h”
4. Una vez funcionando su programa base, explique mediante diagramas como funcionan las rutinas:
    a. decrypt (key, *ciph, len) y encrypt (key, *ciph, len)
    b. tryKey (key, *ciph, len)
    c. memcpy
    d. strstr
5. Describa el uso y flujo de comunicación de las primitivas de MPI:
    a. MPI_Irecv
    b. MPI_Send
    c. MPI_Wait
6. Luego, cifre un texto cargado desde un archivo (.txt) usando una llave privada
7. Medir el tiempo de búsqueda de la llave privada mediante bruteforce, recuerde hacer varias mediciones y promediarlas para obtener un tiempo más adecuado.
8. Subir al foro en Canvas un archivo de texto cifrado y la palabra clave a buscar en el texto. La palabra clave debe ser única y suficientemente diferente para evitar que suceda de forma aleatoria.
9. Un aspecto interesante del presente problema/temario es que los speedups obtenidos pueden ser 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, 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=1iter). 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.

Para poder ver y comprender tal fenómeno, realice cambios a la llave privada y analice el desempeño de los siguientes cambios, asumiendo 4 procesos (ojo, adaptar dependiendo de los procesos que usen):

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

10. Modifique en el programa la forma en que se distribuye el rango de búsqueda para cada proceso y lograr un speedup más consistente. Documente y evidencie la mejora obtenida.
11. Buscar la llave de al menos 2 textos cifrados publicados por otros grupos y realizar un mínimo de 10 pruebas en cada caso para medir el speedup logrado. Variar el número de procesos en incrementos de +2 y -2 y realizar la búsqueda sobre los mismos textos, midiendo speedup y eficiencia.

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:

a. Código de autoría propia en C/C++
b. Historial del control de versiones del programa en git como mínimo 2 semanas antes de la entrega del proyecto
c. Uso de OpenMPI
d. Versión secuencial y paralela del algoritmo
e. Las mediciones y elementos requeridos indicados en la sección 4 de Actividades.
f. Tamaño máximo del texto a cifrar y subir a canvas: 350 palabras.

6. Criterios diferenciadores (extra: hasta 20%)

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 15% extra a los grupos que logren interconectar 2+ máquinas y correr el bruteforce en ellas. Pueden usar Raspberry PI’s, las máquinas de sus compañeros de equipo, alguna otra máquina de su propiedad, etc. Pueden basarse o comenzar la configuración siguiendo el siguiente link:
https://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/

7. Evaluación

	Informe − 25 puntos	**valor**
1	Mínimo 1, Maximo 2 páginas de antecedentes numéricos y conceptuales sobre DES, cifrado y bruteforce. El diagrama de flujo puede ser 1 página adicional	4
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	4
3	Conclusión / recomendación − resumir los retos encontrados y las soluciones para la implementación del temario de forma paralela con OpenMPI	4
4	Diagrama de flujo del programa − detalle de todos los pasos incluyendo: • Captura de argumentos • Solicitud de ingreso de datos (ubicación del archivo cifrado, palabra clave) • Programación defensiva • Tareas realizadas por los nodos • Mecanismos de sincronía y mensajes • Despliegue de resultados	4
5	Anexo 1 − Catálogo de funciones • 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	4
6	Anexo 2 − Bitácora de pruebas y speedups, con los mínimos de pruebas indicados anteriormente.	5

	Programa y presentación − 75 puntos	**valor**
1	**Entrega preliminar** con los siguientes criterios: • Documento PDF con la información de antecedentes de DES y diagrama de flujo. Explicación sobre las rutinas decrypt, tryKey, memcpy y strstr. • Texto cifrado usando el programa ejemplo y una llave de 56 bits. • Resultados de la medición de tiempo para encontrar la llave privada de su equipo mediante bruteforce usando la palabra clave como criterio de búsqueda. • Publicación en el foro del texto cifrado y la palabra clave que se encuentra en el texto	25
2	Programa funcionando correctamente para la búsqueda mediante bruteforce de la llave privada usada para cifrar un texto. • 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. • Descifrado de al menos 2 textos cifrados por otros equipos. • 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 el rango numérico para búsqueda de la llave	25
3	Documentación y comentarios explicativos sobre las partes importantes del programa. Diseñe su propio estilo de documentación y úselo consistentemente en todo el programa	5
4	Programa ordenado, con nombres de variables de entrada y salida y nombres de rutinas adecuadamente identificadas (nombres nemotécnicos)	5
5	Inclusión de al menos un mecanismo de intercambio de mensajes. Manejo adecuado de inicialización y destrucción de objetos.	5
6	Coevaluación del trabajo del equipo, evaluación propia y de otros integrantes.	10

8. Entregar en Canvas

<u>Entrega Parcial</u>: (Viernes 29 de Abril)

1. **Planificación de actividades: metas que se esperan alcanzar para la siguiente semana.**
2. **Código parcial en repositorio git con todos los avances al momento.**
3. **Documentación preliminar.**
4. **Mediciones de tiempo sobre código base.**

<u>Entrega Final y presentación: (Viernes 13 de Mayo)</u>.

1. **Material a entregar en *Canvas*:**
    a. Informe del proyecto de investigación en formato PDF, siguiendo las normas para informes UVG en formato pdf, no impreso, 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. Cualquier material relacionado relevante o necesario para el correcto funcionamiento del código.