Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Ing. Juan Luis Garcia
Semestre 2, 2024

Proyecto # 2

Fecha de Entrega: Viernes 11 de octubre, 2023

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.

Objetivos y Competencias:

- Implementa y diseña programas para la paralelización de procesos con memoria distribuida usando Open MPI.
- 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 (bruteforce).
- Comprender y Analizar el comportamiento del speedup de forma estadística.

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 Open MPI, el equipo debe realizar un análisis de speedups y performance 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.

Requisitos: Para la implementación de la solución paralela, cada equipo debe cumplir por lo menos con las siguientes condiciones. Incumplir con ellas conlleva penalizaciones en la nota:

A. Código de autoría propia en C/C++ (Incluyendo README con instrucciones de ejecución de cada programa)
B. Uso de Open MPI
C. Versión secuencial y paralela del algoritmo y sus acercamientos propuestos.
D. Las mediciones y elementos requeridos indicados en las instrucciones del proyecto.

Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Ing. Juan Luis Garcia
Semestre 2, 2024

Contenido:

Puede utilizar como base el código del archivo *bruteforce.c* que se encuentra en Canvas, 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/cree un diagrama de flujo describiendo el algoritmo DES
3. Luego de investigar un poco sobre DES, haremos una prueba de código como introducción y base para el resto del desarrollo. Cree una versión secuencial de su programa en la que de manera secuencial haga la búsqueda de la llave para descifrar el texto a fuerza bruta. Mida el tiempo de búsqueda con distintas longitudes de llaves.
4. 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.
5. Una vez funcionando su programa base, explique, mediante diagramas, texto, dibujos, etc., cómo 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
6. Describa y explique el uso y flujo de comunicación de las primitivas de MPI:
    a. MPI_Irecv
    b. MPI_Send
    c. MPI_Wait

Parte B

1. Ya que estamos familiarizados con el proyecto, 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 que 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: $(2^{56}/4)$. Es decir, 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: $(2^{56}/4) + 1$. Es decir, 18014398509481984L.
d. Reflexione lo observado y el comportamiento del tiempo en función de la llave.

3. Como podemos ver, el acercamiento "naive" no es el mejor posible. Proponga, analice, e implemente 2 alternativas al acercamiento "naive". Tenga como objetivo en mente encontrar un algoritmo que tenga mejor "tiempo paralelo esperado".
4. Describa los acercamientos propuestos, puede apoyar con diagramas de flujo, pseudocódigo, o algoritmo descriptivo.
5. Realice distintas pruebas sobre su código y evalúe como el speedup y tiempos paralelos cambian dependiendo si su llave es "fácil", "medianamente fácil" y "difícil", comparando su acercamiento con la versión "naive" (bruteforce.c) del programa. Para definir en qué es cada uno de estos tipos de llaves lea la discusión que se encuentra a continuación.

Un aspecto interesante del problema de este proyecto 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

Extras:

- 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 Open MP. (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.

Rúbrica:

	Informe – 30%	**Valor**
1	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 Open MPI. 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". No olvide:	10

	● Describir el acercamiento propuesto, se puede apoyar con diagramas de flujo, pseudocódigo, o algoritmo descriptivo. ● Probarlo con 3-4 llaves (fáciles, medianas, difíciles). Compare el tiempo medido con la solución secuencial y la solución "naive".	
5	Anexo 1 – Catálogo de funciones y librerías ● 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%	**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 ● Solución Secuencial funcional ● 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

Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Ing. Juan Luis Garcia
Semestre 2, 2024