UNIVERSIDAD TECNOLOGICA DE SANTIAGO
(UTESA)

PRESENTADO POR:

ERIK CRUZ
1-18-0759

PRESENTADO A:

IVAN MENDOZA

ASIGNATURA:

ALGORITMOS PARALELOS

ASIGNACION:

TAREA SEMANA 2 - ACTIVIDAD

Santiago de los Caballeros
República Dominicana
Febrero, 2024

GITHUB: https://github.com/kelok3rik/TAREA-SEMANA-2-PARALELOS

ACTIVIDAD:

Crear una aplicación con los diferentes algoritmos de sorteos o búsquedas seleccionados en esta plantilla, la aplicación consiste en ordenar un solo arreglo con todos los algoritmos de búsquedas, ejecutarlos todos al mismo tiempo y calcular el tiempo de que algoritmo termino primero el proceso. Presentarlo en clase.

Indice de la entrega:

1. Introducción

Este proyecto se centra en optimizar la búsqueda y ordenación de datos mediante algoritmos implementados en un entorno paralelo. Aprovechando los sistemas modernos con múltiples núcleos de procesamiento, exploramos la distribución de la carga de trabajo entre procesadores para acelerar estas operaciones.

Exploraremos una variedad de algoritmos de búsqueda y ordenación, desde los clásicos hasta los más avanzados, adaptándolos para ejecutarse en paralelo. A través de este enfoque, buscamos mejorar significativamente el rendimiento y la eficiencia de estas operaciones, lo que resultará en aplicaciones más rápidas y robustas en el mundo real.

2. Descripción del Proyecto

El proyecto se centra en la creación de una aplicación que emplea múltiples algoritmos de búsqueda y ordenación en un entorno paralelo. La aplicación ejecutará estos algoritmos de manera simultánea y medirá el tiempo necesario para que cada uno complete su tarea. Posteriormente, se realizará una comparación de los tiempos obtenidos para determinar qué algoritmo ofrece la mayor eficiencia en términos de velocidad de ejecución.

3. Objetivos

a. Objetivo General

Desarrollar una aplicación que implemente algoritmos de búsqueda y ordenación en un entorno paralelo y comparar su rendimiento.

b. Objetivos Específicos

- Implementar algoritmos de búsqueda y ordenación seleccionados.
- Calcular y comparar el tiempo de ejecución de cada algoritmo.
- Analizar el consumo de memoria del proceso paralelo.

4. Definición de Algoritmos Paralelos

Los algoritmos paralelos están diseñados para ejecutarse de manera simultánea en varios procesadores o núcleos. Estos algoritmos dividen una tarea en subproblemas que pueden resolverse de manera independiente, y luego unifican los resultados para obtener la solución completa. El objetivo es mejorar la velocidad de procesamiento y la eficiencia en entornos de procesamiento paralelo.

5. Etapas de los Algoritmos paralelos

a. Partición

En esta etapa, el problema inicial se divide en partes más pequeñas y manejables que pueden ser procesadas de forma independiente por los diferentes procesadores o núcleos de procesamiento. La idea es distribuir la carga de trabajo de manera equitativa.

b. Comunicación

Durante la ejecución paralela, es probable que los diferentes procesadores necesiten intercambiar datos entre sí para coordinar la ejecución y compartir resultados parciales. Esta etapa implica el desarrollo de mecanismos de comunicación eficientes que minimicen el tiempo de transferencia de datos y optimicen el rendimiento general del sistema.

c. Agrupamiento

El agrupamiento implica la organización de tareas relacionadas en unidades lógicas que pueden ser ejecutadas de manera conjunta por un conjunto específico de procesadores.

d. Asignación

En esta etapa, se asignan las tareas a los diferentes procesadores o núcleos de procesamiento de manera que se distribuya la carga de trabajo de manera equitativa y eficiente. La asignación debe tener en cuenta la capacidad de procesamiento de cada procesador, así como las dependencias entre las diferentes tareas, para garantizar un rendimiento óptimo del sistema.

6. Técnicas Algorítmicas Paralelas

las técnicas algorítmicas paralelas son enfoques específicos utilizados para diseñar algoritmos que puedan aprovechar la capacidad de procesamiento paralelo de los sistemas informáticos modernos. Estas técnicas se centran en dividir el problema en partes más pequeñas y distribuir esas partes entre múltiples procesadores o núcleos de procesamiento para que trabajen simultáneamente en la solución. Las mas comunes son:

- **División y conquista:** Esta técnica implica dividir el problema en subproblemas más pequeños y resolver cada subproblema de manera independiente. Luego, los resultados se combinan para obtener la solución final. Es importante que los subproblemas sean lo suficientemente independientes entre sí para que puedan ser procesados en paralelo.
- **Programación dinámica:**La programación dinámica es una técnica que se utiliza para resolver problemas recursivos dividiendo el problema en subproblemas más pequeños y resolviéndolos de manera recursiva. Sin embargo, a diferencia de la división y conquista, la programación dinámica almacena los resultados de los subproblemas resueltos para evitar recalcularlos varias veces. Esto permite reducir la complejidad temporal de algunos algoritmos.
- **Enfoque basado en flujos de datos:** En este enfoque, los datos se dividen en flujos más pequeños que pueden ser procesados de forma independiente. Cada flujo de datos se procesa en un hilo de ejecución diferente, lo que permite una paralelización eficiente de la tarea. Esto es especialmente útil en aplicaciones que implican el procesamiento de grandes volúmenes de datos, como el procesamiento de imágenes o el procesamiento de señales.

7. Modelos de Algoritmos Paralelos

Son estructuras que describen cómo los procesos paralelos se coordinan para ejecutar tareas eficientemente en sistemas con múltiples procesadores o núcleos. Algunos modelos comunes son:

- **Modelo Maestro/Esclavo:** Un proceso maestro divide el trabajo y lo asigna a procesos esclavos para su procesamiento.
- **Modelo de Paso de Mensajes:** Los procesos se comunican intercambiando mensajes entre sí.
- **Modelo de Memoria Compartida:** Todos los procesos acceden a una región de memoria compartida para coordinar sus acciones.
- **Modelo de Flujo de Datos:** Los datos fluyen a través de una red de procesadores, donde cada uno realiza operaciones en los datos que pasan.

8. Algoritmos de Búsquedas y Ordenamiento (Adjuntar PSeudocódigo, código de cada uno y concepto)

a. Búsqueda Secuencial

La búsqueda secuencial recorre secuencialmente cada elemento del arreglo hasta encontrar el valor buscado o hasta recorrer todo el arreglo si el valor no está presente.

Pseudocodigo:

```
Para cada elemento en el arreglo:
    Si el elemento es igual al valor buscado:
        Devolver la posición del elemento
Devolver -1 si no se encuentra el elemento
```

Codigo:

```
static int BusquedaSecuencial(int[] arreglo, int valor)
{
    for (int i = 0; i < arreglo.Length; i++)
    {
        if (arreglo[i] == valor)
        {
            return i;
        }
    }
    return -1;
      }
```

b. Búsqueda Binaria

Funciona dividiendo repetidamente el espacio de búsqueda a la mitad y descartando una mitad en cada paso, similar a cómo se busca un término en un diccionario. Si el valor buscado es igual al valor en el medio del rango actual, se devuelve su posición. Si el valor buscado es menor, se busca en la mitad izquierda; si es mayor, se busca en la mitad derecha. Este proceso se repite hasta que se encuentra el elemento o el rango de búsqueda se reduce a cero.

Pseudocodigo:

```
Inicio = 0
Fin = longitud del arreglo - 1
Mientras el inicio sea menor o igual que el fin:
    Medio = (Inicio + Fin) / 2
    Si el valor en la posición Media es igual al valor buscado:
```

```
        Devolver Media

    Si el valor en la posición Media es mayor que el valor buscado:

        Fin = Media - 1

    Si el valor en la posición Media es menor que el valor buscado:

        Inicio = Media + 1

Devolver -1 si el valor no se encuentra
```

Codigo:

```
static int BusquedaBinaria(int[] arreglo, int valor)
 {
     int inicio = 0;
     int fin = arreglo.Length - 1;

     while (inicio <= fin)
     {
         int medio = (inicio + fin) / 2;
         if (arreglo[medio] == valor)
         {
             return medio;
         }
         else if (arreglo[medio] < valor)
             inicio = medio + 1;
         else
             fin = medio - 1;
     }

     return -1;
        }
```

c. Algoritmo de Ordenamiento de la Burbuja

compara elementos adyacentes y los intercambia si están en el orden incorrecto. Cada pasada a través de la lista coloca el elemento más grande en su posición final. El proceso se repite para cada elemento restante hasta que toda la lista esté ordenada.

Pseudocodigo:

```
Inicio de OrdenamientoBurbuja(arr: arreglo)

    Para i de 0 a n-1:

        Para j de 0 a n-i-1:

            Si arreglo[j] > arreglo[j+1]:

                Intercambiar(arreglo[j], arreglo[j+1])

Fin de OrdenamientoBurbuja
```

Codigo:

```
static void Burbuja(int[] arreglo)
{
    for (int i = 0; i < arreglo.Length - 1; i++)
    {
        for (int j = 0; j < arreglo.Length - 1 - i; j++)
        {
            if (arreglo[j] > arreglo[j + 1])
            {
                int temp = arreglo[j];
                arreglo[j] = arreglo[j + 1];
                arreglo[j + 1] = temp;
            }
        }
    }

      }
```

d. Quick Sort

Selecciona un elemento como pivote y divide el arreglo en dos subarreglos: uno con elementos menores que el pivote y otro con elementos mayores que el pivote. Luego, ordena recursivamente los subarreglos y combina los resultados. Quick Sort es rápido y eficiente para arreglos grandes y desordenados.

Pseudocodigo:

```
Inicio de QuickSort(arr: arreglo, inicio: entero, fin: entero)

    Si inicio < fin:

        Pivote = Particionar(arr, inicio, fin)

        QuickSort(arr, inicio, Pivote - 1)

        QuickSort(arr, Pivote + 1, fin)

Fin de QuickSort


Inicio de Particionar(arr: arreglo, inicio: entero, fin: entero)

    Pivote = arr[fin]

    i = inicio - 1

    Para j de inicio a fin-1:

        Si arr[j] < Pivote:

            Incrementar i

            Intercambiar(arr[i], arr[j])

    Intercambiar(arr[i+1], arr[fin])
```

```
    Devolver i + 1

Fin de Particionar
```

Codigo:

```
static void QuickSort(int[] arreglo, int izquierda, int derecha)
{
    if (izquierda < derecha)
    {
        int indiceParticion = Particionar(arreglo, izquierda, derecha);

        QuickSort(arreglo, izquierda, indiceParticion - 1);
        QuickSort(arreglo, indiceParticion + 1, derecha);
    }
      }

static int Particionar(int[] arreglo, int izquierda, int derecha)
{
    int pivote = arreglo[derecha];
    int indiceMenor = izquierda - 1;

    for (int j = izquierda; j < derecha; j++)
    {
        if (arreglo[j] <= pivote)
        {
            indiceMenor++;
            int temp = arreglo[indiceMenor];
            arreglo[indiceMenor] = arreglo[j];
            arreglo[j] = temp;
        }
    }

    int temp1 = arreglo[indiceMenor + 1];
    arreglo[indiceMenor + 1] = arreglo[derecha];
    arreglo[derecha] = temp1;

    return indiceMenor + 1;
}
```

e. Método de Inserción

El método de inserción es un algoritmo de ordenamiento que recorre el arreglo y compara cada elemento con los elementos precedentes, insertándolo en su posición correcta en el subarreglo ya ordenado. Comienza con el segundo elemento del arreglo y lo inserta en la posición adecuada entre los elementos anteriores. Este proceso se repite para cada elemento restante hasta que todo el arreglo esté ordenado.

Pseudocodigo:

```
Inicio de Insercion(arr: arreglo)

    Para i de 1 a n-1:
```

```
        ValorActual = arreglo[i]

        j = i - 1

        Mientras j sea mayor o igual que 0 y arreglo[j] sea mayor que
ValorActual:

            arreglo[j+1] = arreglo[j]

            Decrementar j

        arreglo[j+1] = ValorActual

Fin de Inserción
```

Codigo:

```
static void Insercion(int[] arreglo)
{
    for (int i = 1; i < arreglo.Length; i++)
    {
        int clave = arreglo[i];
        int j = i - 1;

        while (j >= 0 && arreglo[j] > clave)
        {
            arreglo[j + 1] = arreglo[j];
            j = j - 1;
        }
        arreglo[j + 1] = clave;
    }
      }
```

9. Programa desarrollado

a. Explicación de su funcionamiento

El programa comienza generando un arreglo aleatorio de 1000 elementos al iniciar la aplicación. Este arreglo servirá como conjunto de datos sobre el cual se realizarán las operaciones de búsqueda y ordenamiento.

Cuando el usuario hace clic en el botón "Buscar", el programa verifica si se ha ingresado un valor válido en el campo de texto. Una vez validado el valor, el programa procede a ejecutar en paralelo los algoritmos de búsqueda y ordenamiento sobre el arreglo generado.

Para ejecutar los algoritmos en paralelo, se utilizan tareas (Task), una para cada algoritmo de búsqueda y ordenamiento. Cada tarea registra el tiempo de ejecución del algoritmo correspondiente utilizando un objeto Stopwatch. Una vez que todas las tareas se completan mediante el uso de Task.WhenAll(), el programa compara los tiempos de ejecución de cada algoritmo para determinar cuál fue el más rápido en cada categoría (búsqueda y ordenamiento). Para esto, se utiliza una simple lógica de comparación de tiempos.

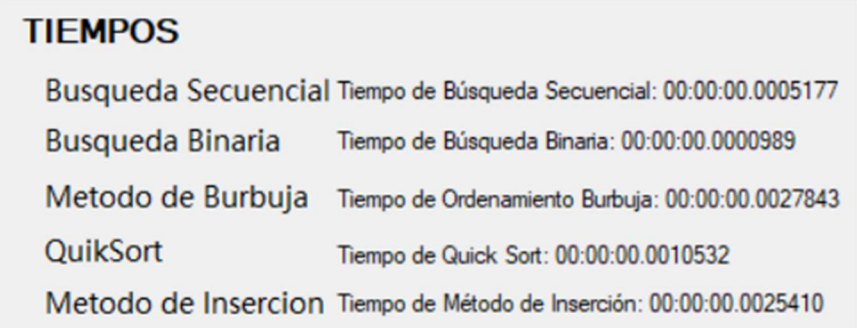
Una vez identificados los algoritmos más rápidos en cada categoría, el programa muestra los resultados. Estos mensajes incluyen el nombre del algoritmo más rápido de búsqueda y su tiempo de ejecución, así como el nombre del algoritmo más rápido de ordenamiento y su tiempo de ejecución.

b. Fotos de la aplicación

c. Link de Github y Ejecutable de la aplicación

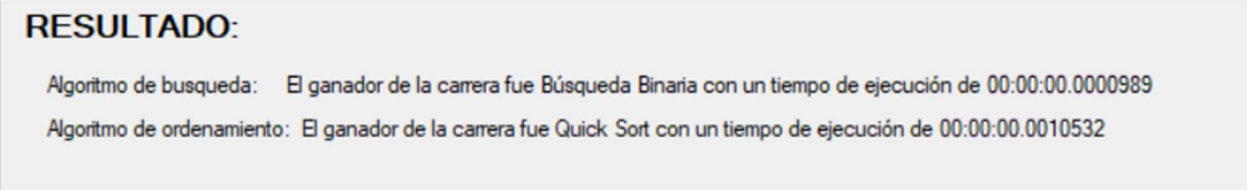
GITHUB: https://github.com/kelok3rik/TAREA-SEMANA-2-PARALELOS

d. Resultados (Tiempo en terminar los ordenamientos y búsqueda de cada algoritmo)



e. ¿Qué tanta memoria se consumió este proceso?

10. ¿Cuál fue el algoritmo que realizo la búsqueda y el ordenamiento más rápido?



11. Conclusión

La actividad se centra en la implementación y comparación de algoritmos de búsqueda y ordenamiento en un entorno paralelo. Se exploran diversas técnicas algorítmicas paralelas, como la división y conquista, la programación dinámica y el enfoque basado en flujos de datos, adaptándolas para aprovechar al máximo la capacidad de procesamiento paralelo. Se definen modelos de algoritmos paralelos, como el modelo Maestro/Esclavo, el modelo de Paso de Mensajes, el modelo de Memoria Compartida y el modelo de Flujo de Datos. Se implementan algoritmos de búsqueda como la Búsqueda Secuencial y la Búsqueda Binaria, así como algoritmos de ordenamiento como el Ordenamiento de la Burbuja, Quick Sort y el Método de Inserción. El programa desarrollado ejecuta los algoritmos en paralelo sobre un conjunto de datos generado aleatoriamente y compara los tiempos de ejecución para determinar los más rápidos en cada categoría. Se concluye que el proyecto proporciona una sólida comprensión de los algoritmos paralelos y su aplicación en la optimización de la búsqueda y ordenación de datos, destacando la importancia de considerar el rendimiento y la eficiencia en el diseño de aplicaciones que aprovechan la capacidad de procesamiento paralelo.