	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: JORGE SOLANO GALVEZ

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 02

No. de práctica(s): PROYECTO 02 - VERSIONES PARALELAS DE UN ALGORITMO SERIAL-

Integrante(s): Tapia Garcia Andrés

No. de lista o brigada: 36

Semestre: 2024-1

Fecha de entrega: 04/12/23

Observaciones:

CALIFICACIÓN: _____

PROYECTO 2 EDAII -Implementación de dos versiones paralelas de un algoritmo serial -

El algoritmo que voy a paralelizar en este proyecto es el Algoritmo de ordenamiento que realice en el Examen 1 de la materia, al cual llamé junto a mi equipo: “TilinSort”. El Algoritmo TilinSort lo que realiza es algo muy similar que bubbleSort solo que dos veces más eficiente, ya que compara el primer elemento del arreglo (Arreglo en el índice cero), con el último elemento del arreglo (Longitud del arreglo menos 1) (En pseudocódigo: len(Arr)-1), si el último elemento es menor que el primero, realiza un swap, y bajo esa misma iteración, el primer elemento del arreglo se compara con el segundo elemento del arreglo, y si el segundo es menor que el primer elemento, entonces realizan swap, en la siguiente iteración, el primer elemento se compara con el tercer elemento y el penúltimo, este ciclo se detendrá hasta que el elemento que recorre el arreglo por la izquierda sea igual a el elemento que recorre el arreglo por la derecha, o hasta que los dos estén alejados una unidad.

Conociendo la naturaleza de “TilinSort” mencionaré la lógica detrás de la paralelización que se realizó:

- Separamos el arreglo en dos arreglos ocupando las cláusula de openMP:
  - #Pragma omp parallel for

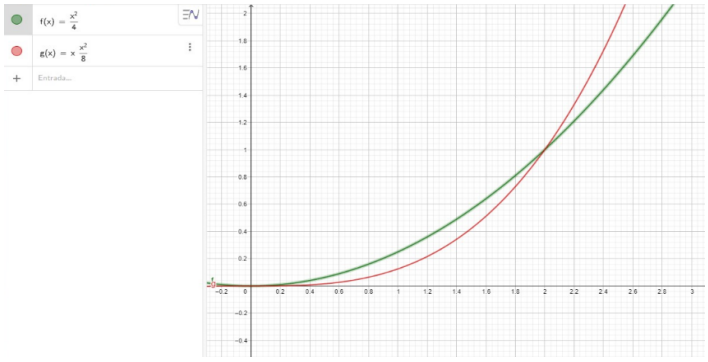
- Los metemos en una región paralela, en la que TilinSort ordenara de forma paralela la primera mitad del arreglo con un hilo, mientras que ordena con otro hilo la otra mitad de arreglo ocupando la cláusula de OpenMP:
  - #pragma omp parallel sections

- Con estos dos arreglos ordenados, salimos de la región paralela y basándonos en la lógica de mergeSort de ordenar dos arreglos que ya están ordenados, creamos 3 variables enteras e inicializadas en cero:

  - ‘i’, ‘j’ y ‘k’

  Estas variables nos servirán para recorrer los arreglos, i, para recorrer la parte izquierda ordenada del arreglo original, j para recorrer la parte derecha ordenada del arreglo original, y k nos servirá para ir metiendo los valores del arreglo L (parte izquierda) y los valores del arreglo R (parte derecha) de forma ordenada en el Arreglo completo

De esta forma puedo asumir que entonces la complejidad de la versión paralela es: n/2*O(TilinSort) siendo la complejidad de TilinSort es O(n^2/4), quedando entonces la complejidad de la versión paralela como O(n^3 /8), Si lo vemos de forma general, quedarían unas gráficas similares a estas:



Podemos ver como para bajas instancias el Algoritmo de ordenamiento conviene que sea paralelizado, pero, llegará un momento en el que no convendrá tener al algoritmo paralelizado, por que tendrá tiempos de ejecución mayores al serial.

La gráfica roja representa el algoritmo paralelo, y la gráfica verde representa el algoritmo serial

- Análisis complejo de TilinSort

ANALISIS COMPLEJO DE TILIN SORT

+ Code + Markdown

```
#########################################################################################################################
def TilinSort(arr):                                 #||MEJOR CASO   ||PEOR CASO     ||CASO PROMEDIO     ||ANALISIS ESPACIAL
    fin=len(arr)-1                                  #||6            ||6             ||6                 ||
    n=0                                             #||3            ||3             ||3                 ||
    flag=True                                       #||3            ||3             ||3                 ||
    while (n<fin):                                  #||(n+1)/2      ||(n+1)/2       ||(n+1)/2           ||
        i=n+1                                       #||5n/2         ||5n/2          ||5n/2              ||
        k=fin                                       #||3n/2         ||3n/2          ||3n/2              ||
        flag=True                                   #||3n/2         ||3n/2          ||3n/2              ||
        while(flag) :                               #||(2n/2)(n/2)  ||(2n/2)(n/2)   ||2n/2(n/2)         ||
            if arr[n]>=arr[k]:                      #||6n/2*n/2     ||(6n/2)*n/2    ||6n/2*n/2          ||
                arr[n], arr[k] = arr[k], arr[n]     #||             ||7n/2*n/2      ||7n/2*n/2          ||
            if arr[n]>=arr[i]:                      #||6n/2*n/2     ||6n/2*n/2      ||6n/2*n/2          ||
                arr[n], arr[i] = arr[i], arr[n]     #||             ||7n/2*n/2      ||7n/2*n/2          ||

            if i==k or k==i+1:                      #||10n/2*n/2    ||10n/2*n/2     ||10n/2*n/2         ||
                flag=False                          #||3n/2*n/2     ||3n/2*n/2      ||3n/2*n/2          ||
            k-=1                                    #||4n/2*n/2     ||4n/2*n/2      ||4n/2*n/2          ||
            i+=1                                    #||4n/2*n/2     ||4n/2*n/2      ||4n/2*n/2          ||
        n+=1                                        #||4n/2         ||4n/2          ||4n/2              ||
    return arr                                      #||2            ||2             ||2                 ||

#POLINOMIO DE COMPLEJIDAD TEMPORAL PARA EL MEJOR CASO =          (35n^2)/4 + 19n/2 + 14  :  O(n^2)
#POLINOMIO DE COMPLEJIDAD TEMPORAL PARA EL PERO, CASO PROMEDIO = (49n^2)/4 + 19n/2 + 14  :  O(n^2)
#########################################################################################################################
```

- CÓDIGO FUENTE DEL TILINSORT PARALELIZADO

```
115  void TilinSortParallel(int Arr[], int size, FILE *file) {
116      int mid = size / 2;
117      int L[mid];
118      int R[size - mid];
119      double start_time, end_time, elapsed_time;
120
121      start_time = omp_get_wtime();  // Medir el tiempo al inicio de la iteración
122
123      #pragma omp parallel for
124      for (int i = 0; i < mid; i++) {
125          L[i] = Arr[i];
126      }
127
128      #pragma omp parallel for
129      for (int j = mid; j < size; j++) {
130          R[j - mid] = Arr[j];
131      }
132
133      // Llamada a TilinSortSerial para ordenar cada subarreglo en paralelo
134      #pragma omp parallel sections
135      {
136          #pragma omp section
137          {
138              TilinSortSerialP(L, mid);
139          }
140
141          #pragma omp section
142          {
143              TilinSortSerialP(R, size - mid);
144          }
145      }
```

```c
    int i = 0;
    int j = 0;
    int k = 0;

    // Combinar los subarreglos ordenados
    while (i < mid && j < size - mid) {
        if (L[i] <= R[j]) {
            Arr[k] = L[i];
            i++;
        } else {
            Arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copiar los elementos restantes de L[], si los hay
    while (i < mid) {
        Arr[k] = L[i];
        i++;
        k++;
    }

    // Copiar los elementos restantes de R[], si los hay
    while (j < size - mid) {
        Arr[k] = R[j];
        j++;
        k++;
    }

```

```c
    end_time = omp_get_wtime();  // Medir el tiempo al final de la iteración
    elapsed_time = end_time - start_time;

    fprintf(file, "%d, %f\n", size, elapsed_time);  // Guardar en el archivo CSV
}
```

- EJECUCIÓN DEL PROGRAMA PARALELO

```
EDAII_PROYECTO_2 > C test.c > main()

    //TilinSortSerialP(AuxArr, elementos);
    TilinSortParallel(AuxArr, elementos);
    //TilinSortParallelTask(AuxArr, elementos);

    printf("]\n  Sorted Arr: [");
    for (int j = 0; j < elementos; j++) {

PROBLEMS   OUTPUT   TERMINAL   ...        powershell

-a----     04/12/2023  06:30 p. m.         145678 test.exe

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> gcc -fopenmp tes
c -o test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2>
```

TilinSort paralelizado con task, CÓDIGO:

PROYECTO 2 > C test.c > main()

```c
void TilinSortParallelTask(int Arr[], int size) {
    int mid = size / 2;
    int L[mid];
    int R[size - mid];

    // Copiar elementos a L y R en paralelo
    #pragma omp parallel for
    for (int i = 0; i < size; i++) {
        if (i < mid) {
            L[i] = Arr[i];
        } else {
            R[i - mid] = Arr[i];
        }
    }

    // Ordenar subarreglos en paralelo usando tareas
    #pragma omp parallel
    {
        #pragma omp single nowait
        {
            #pragma omp task
            {
                TilinSortSerialP(L, mid);
            }
            #pragma omp task
            {
                TilinSortSerialP(R, size - mid);
            }
        }
    }
```

```c
    int i = 0;
    int j = 0;
    int k = 0;

    // Combinar los subarreglos ordenados
    while (i < mid && j < size - mid) {
        if (L[i] <= R[j]) {
            Arr[k] = L[i];
            i++;
        } else {
            Arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copiar los elementos restantes de L[], si los hay
    while (i < mid) {
        Arr[k] = L[i];
        i++;
        k++;
    }

    // Copiar los elementos restantes de R[], si los hay
    while (j < size - mid) {
        Arr[k] = R[j];
        j++;
        k++;
    }
}
```

EJECUCIÓN DE TilinSort con Task:

```
    //TilinSortSerialP(AuxArr, elementos);
    //TilinSortParallel(AuxArr, elementos);
    TilinSortParallelTask(AuxArr, elementos);

    printf("]\n  Sorted Arr: [");
    for (int j = 0; j < elementos; j++) {
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
    return 0;
}
PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]

PS C:\Users\Pablo TG\Desktop\01A_Tapia_A\EDAII_PROYECTO_2> ./test
Arr unsorted: [41,467,334,500,1169,1724,1478,1358,962,464,]
  Sorted Arr: [41,334,464,467,500,962,1169,1358,1478,1724,]
```

GRÁFICA COMPARATIVA CON 20,000 INSTANCIAS: