Practica N 12 ” ALGORITMOS PARALELOS 1” EDA 2

Oscar Daniel Rosario Morales,Israel Hipolito Mejia Alba

17 de noviembre de 2018

Grupo: 8
Profesor : Tista Garcia Edgar

0.1. Objetivo:

El estudiante conocerá y aprenderá a utilizar algunas de las directivas de OpenMP para implementar algún algoritmo paralelo.

0.2. Actividades:

-Realizar ejemplos de funcionamiento de directivas de OpenMP en el lenguaje C, tales como los constructores critical, section,single,master,barrier y las cláusulas reduction y nowait. -Realizar programas que resuelvan un problema de forma paralela utilizando distintas directivas de OpenMP.

0.3. Desarrollo

Actividad 1 Completar la versión serie y paralela del ejemplo explicado de búsqueda del valor mayor de los elementos de un arreglo unidimensional de enteros

Algoritmo secuencial.

Podemos ver como el algoritmo busca elemento a elemento el de mayor valor y lo va guardando en la variable max si este cambia, para al final solo imprimirlo.

```c
/*Actividad 1: completar la version serie y paralela del ejemplo
mayor de los elementos de un arreglo unidimensional de enteros*/
#include <stdio.h>
#include <omp.h>
 int buscarMaximo(int *a, int n);
 int buscarMaximoparalelo(int *a, int n);
 int main(){
    printf("Actividad 1");
    int a[10]={0,1,2,3,4,10,6,7,8,9};
    int b=10;
    printf("\n\nllamando a buscar maximo de forma en serie = %d", buscarMaximo(a, b));
    printf("\n--------------------------------------------------");
    printf("\n\nllamando a buscar maximo de forma paralela = %d", buscarMaximoparalelo(a, b));
 }

 int buscarMaximo(int *a, int n){
    int max, i;
    max=a[0];
        for(i=1; i<n; i++){
        if(a[i]>max)
            max=a[i];
        }
    return max;
 }
```

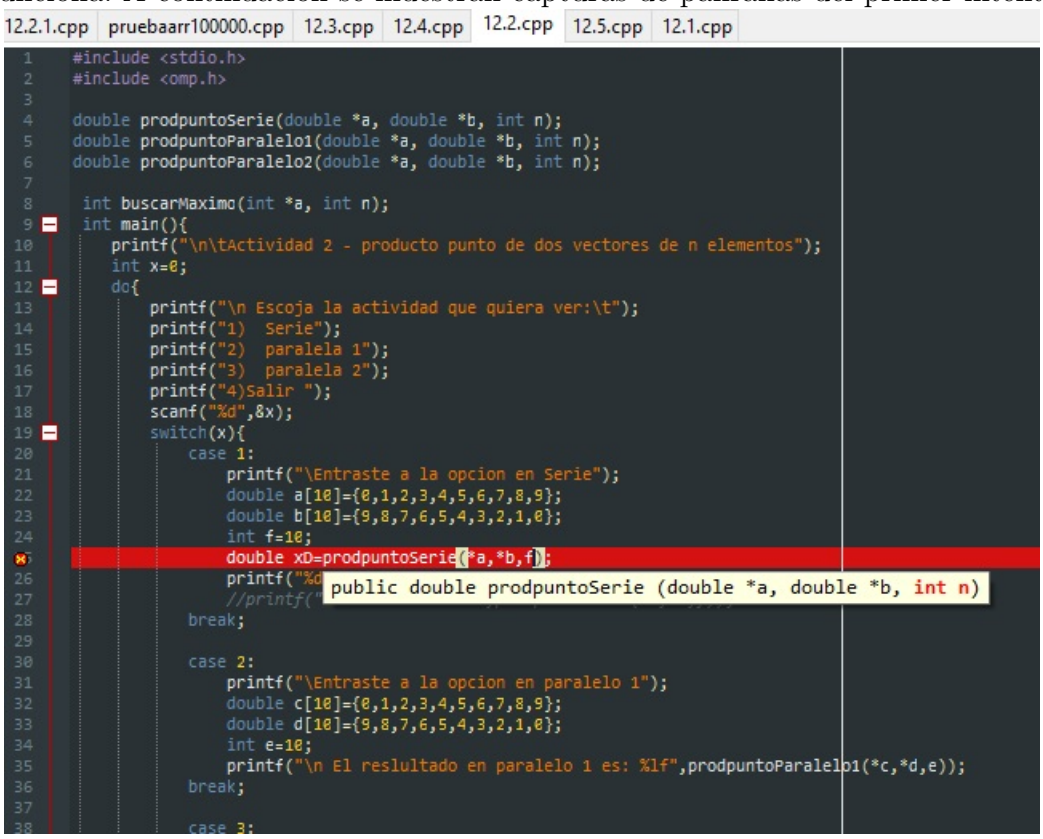
Algoritmo paralelo.

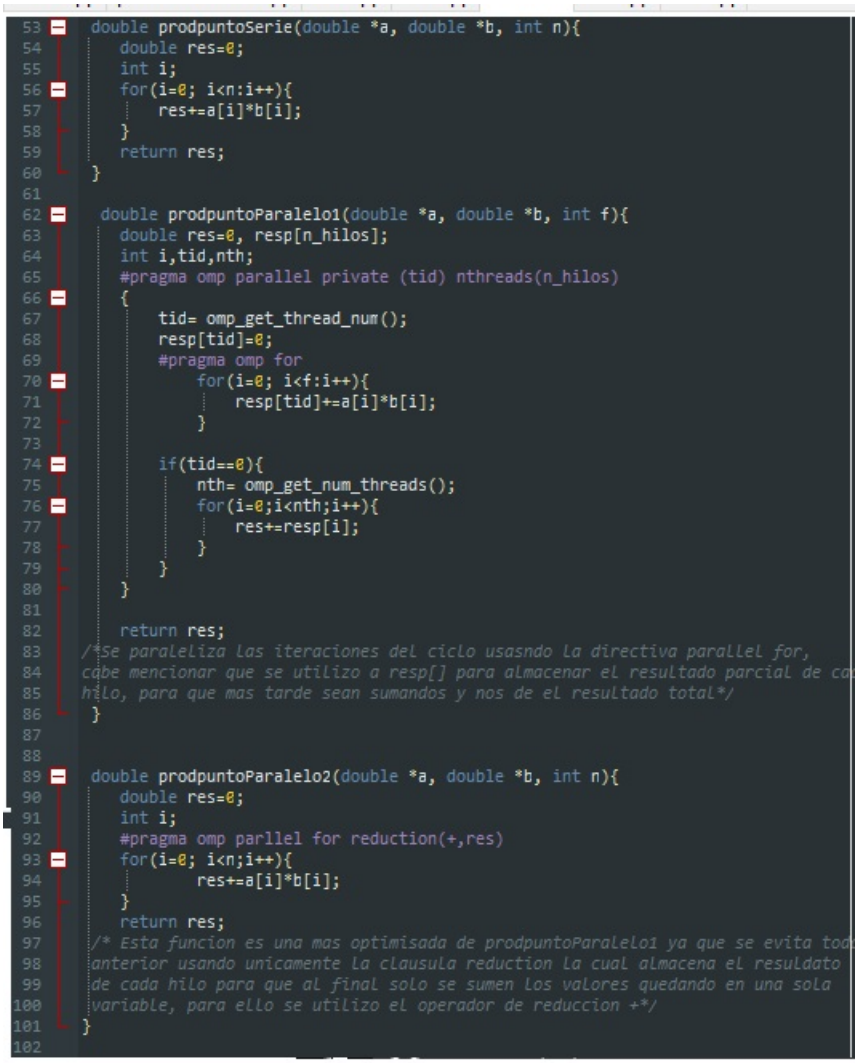
Este algoritmo las iteraciones del ciclo for en distintos hilos con parallel for, después con la directiva critical hace que se use la variable max un hilo a la vez para no tener problemas al actualizar su valor en ejecución.

Actividad 2 . Completar la versión serie y sus dos versiones paralelas del ejemoki explicado del producto punto de dos vectores de n elementos enteros

Tuvimos problemas al mandar a llamar las funciones y a pesar de estar revisándolas constantemente, no logramos hacer que funcionaran, sin embargo realizamos esta actividad nuevamente

pero sin usar métodos y logro funcionar. Primero se mostrara el intentento fallido y después el que si funciona. A continuación se muestran capturas de panrallas del primer intento:





```
double prodpuntoSerie(double *a, double *b, int n){
    double res=0;
    int i;
    for(i=0; i<n:i++){
        res+=a[i]*b[i];
    }
    return res;
}

 double prodpuntoParalelo1(double *a, double *b, int f){
    double res=0, resp[n_hilos];
    int i,tid,nth;
    #pragma omp parallel private (tid) nthreads(n_hilos)
    {
        tid= omp_get_thread_num();
        resp[tid]=0;
        #pragma omp for
            for(i=0; i<f:i++){
                resp[tid]+=a[i]*b[i];
            }

        if(tid==0){
            nth= omp_get_num_threads();
            for(i=0;i<nth;i++){
                res+=resp[i];
            }
        }
    }

    return res;
/*se paraleliza las iteraciones del ciclo usasndo la directiva parallel for,
cabe mencionar que se utilizo a resp[] para almacenar el resultado parcial de ca
hilo, para que mas tarde sean sumandos y nos de el resultado total*/
 }


 double prodpuntoParalelo2(double *a, double *b, int n){
    double res=0;
    int i;
    #pragma omp parllel for reduction(+,res)
    for(i=0; i<n;i++){
            res+=a[i]*b[i];
    }
    return res;
/* Esta funcion es una mas optimisada de prodpuntoParalelo1 ya que se evita tod
anterior usando unicamente la clausula reduction la cual almacena el resuldato
de cada hilo para que al final solo se sumen los valores quedando en una sola
variable, para ello se utilizo el operador de reduccion +*/
}
```

Si el programa hubiera funcionado, en la función prodpuntoSerie se ubiera hecho la suma de todos los resultados de la multiplicación de cada elemento de ambos arreglos en un solo hilo, por lo que se haría el producto punto como normalmente se conoce. En el primer producto punto paralelo, se usaría parallel for para realizar la paralelización de las iteraciones de cada ciclo y con resp[] se almacena cada resultado parcial de cada hilo, de esta manera al final solo se suman y tenemos nuestro producto punto realizado en conjunto de todos los hilos. Finalmente con el prodpuntoParalelo2 se utiliza únicamente la cláusula reduction(+,res) lo cual nos suma todos los resultados de res en cada hilo, de tal manera que al final tenemos un único valor para res, que es el que retorna. Ahora analizaremos rápidamente el segundo intento en el que si funciono correctamente:

12.2.1.cpp | pruebaarr100000.cpp | 12.3.cpp | 12.4.cpp | 12.2.cpp | 12.5.cpp | 12.1.cpp

```
#include <stdio.h>
#include <omp.h>
#include <time.h>
 int buscarMaximo(int *a, int n);
 int main(){
    double a[10]={0,1,2,3,4,5,6,7,8,9};
    double b[10]={9,8,7,6,5,4,3,2,1,0};
    int n=10;
    clock_t t_inicio, t_fin;
    double seg;
    t_inicio=clock();
    printf("\nA continuacion se mostrara el producto punto realizado en serie:");

    double res=0;
    int i;
    for(i=0; i<n;i++)
        res+=a[i]*b[i];
        t_fin=clock();
        seg=(double)(t_fin - t_inicio)/CLOCKS_PER_SEC;
     printf("\n%lf ",res);
     printf("El tiempo de calculo del producto punto en serie es: %lf segundos ",seg);

     printf("\n\nA continuacion se mostrara el producto punto realizado en Paralelo:");
    res=0; double resp[4];
    int tid,nth;
    t_inicio=clock();
    #pragma omp parallel private (tid)
    #pragma omp nthreads(n_hilos)
    {
        tid= omp_get_thread_num();
        resp[tid]=0;
        #pragma omp for
            for(i=0;i<n;i++){
                resp[tid]+=a[i]*b[i];
            }
        if(tid==0){
            nth= omp_get_num_threads();
            for(i=0;i<n;i++){
                res+=resp[i];
            }
        }
    }
    printf("\n%lf ",res);
    t_fin=clock();
    seg=(double)(t_fin - t_inicio)/CLOCKS_PER_SEC;
    printf("\nEl tiempo de calculo del producto punto en serie es: %lf segundos ",seg);

/*Se paraleliza las iteraciones del ciclo usasndo la directiva parallel for,
cabe mencionar que se utilizo a resp[] para almacenar el resultado parcial de cada
hilo, para que mas tarde sean sumandos y nos de el resultado total*/


    printf("\n\n\nEntraste a la opcion en paralelo 2");
    t_inicio=clock();
    res=0;
    #pragma omp parllel for reduction(+,res)
    for(i=0; i<n;i++){
          res+=a[i]*b[i];
    }
    t_fin=clock();
        seg=(double)(t_fin - t_inicio)/CLOCKS_PER_SEC;
     printf("\nEl tiempo de calculo del producto punto en serie es: %lf segundos ",seg);
    printf("\nEl resultado de paralelizacion 2 es: %lf",res);
```

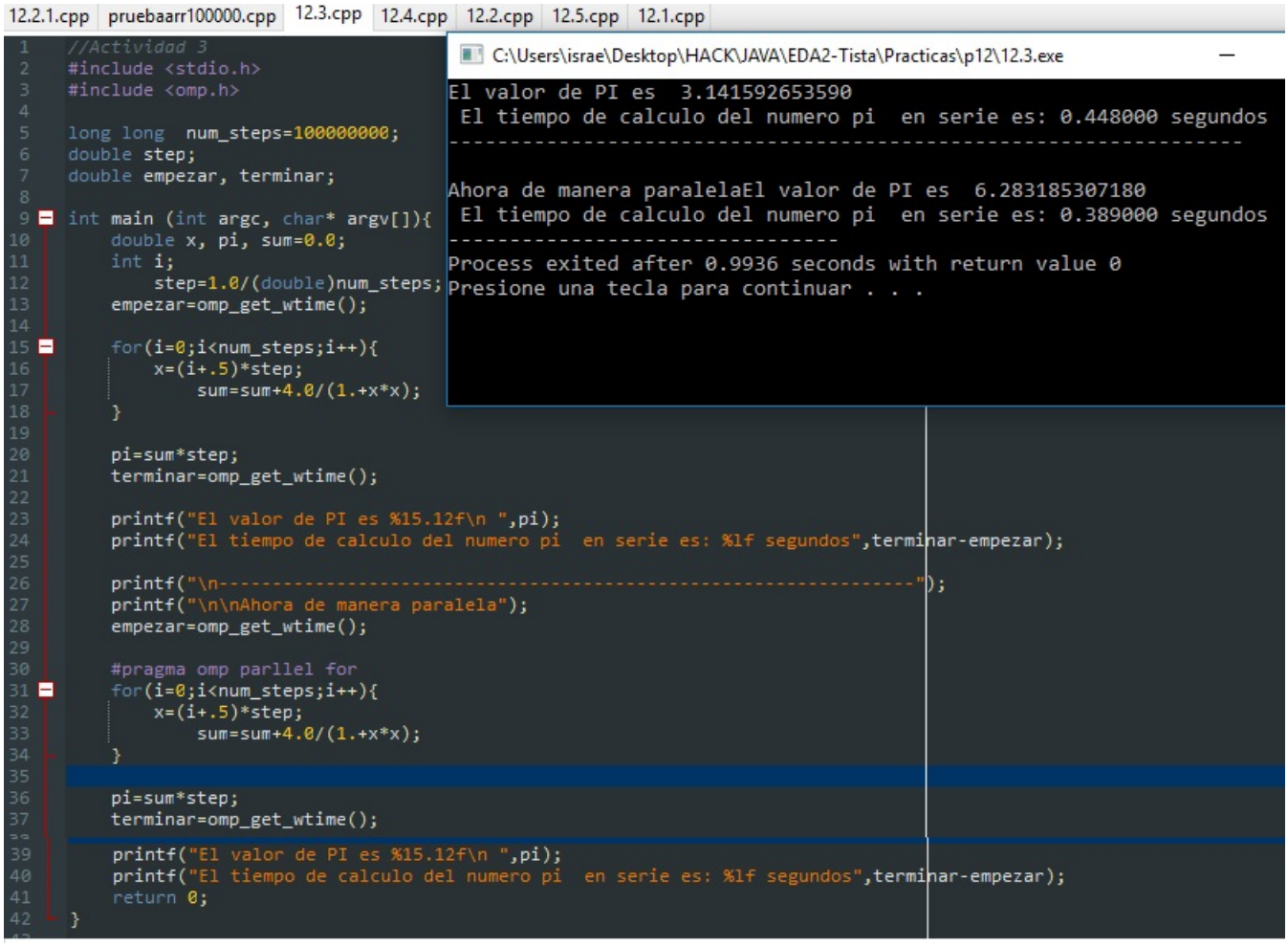
```
A continuacion se mostrara el producto punto realizado en serie:
120.000000 El tiempo de calculo del producto punto en serie es: 0.000000 segundos

A continuacion se mostrara el producto punto realizado en Paralelo:
135.000000
El tiempo de calculo del producto punto en serie es: 0.001000 segundos


Entraste a la opcion en paralelo 2
El tiempo de calculo del producto punto en serie es: 0.000000 segundos
El resultado de paralelizacion 2 es: 120.000000
--------------------------------
Process exited after 0.1147 seconds with return value 0
Presione una tecla para continuar . . .
```

Se puede ver que para este caso tuvimos que usar parallel private y ntdreads de manera separada para que funcionara el programa en el caso de paralelo 1, además se hizo uso de la librería time.h

para medir el tiempo, fuera de eso el procedimiento es mismo, solo que ya no usamos funciones para que el programa haga cada acción.

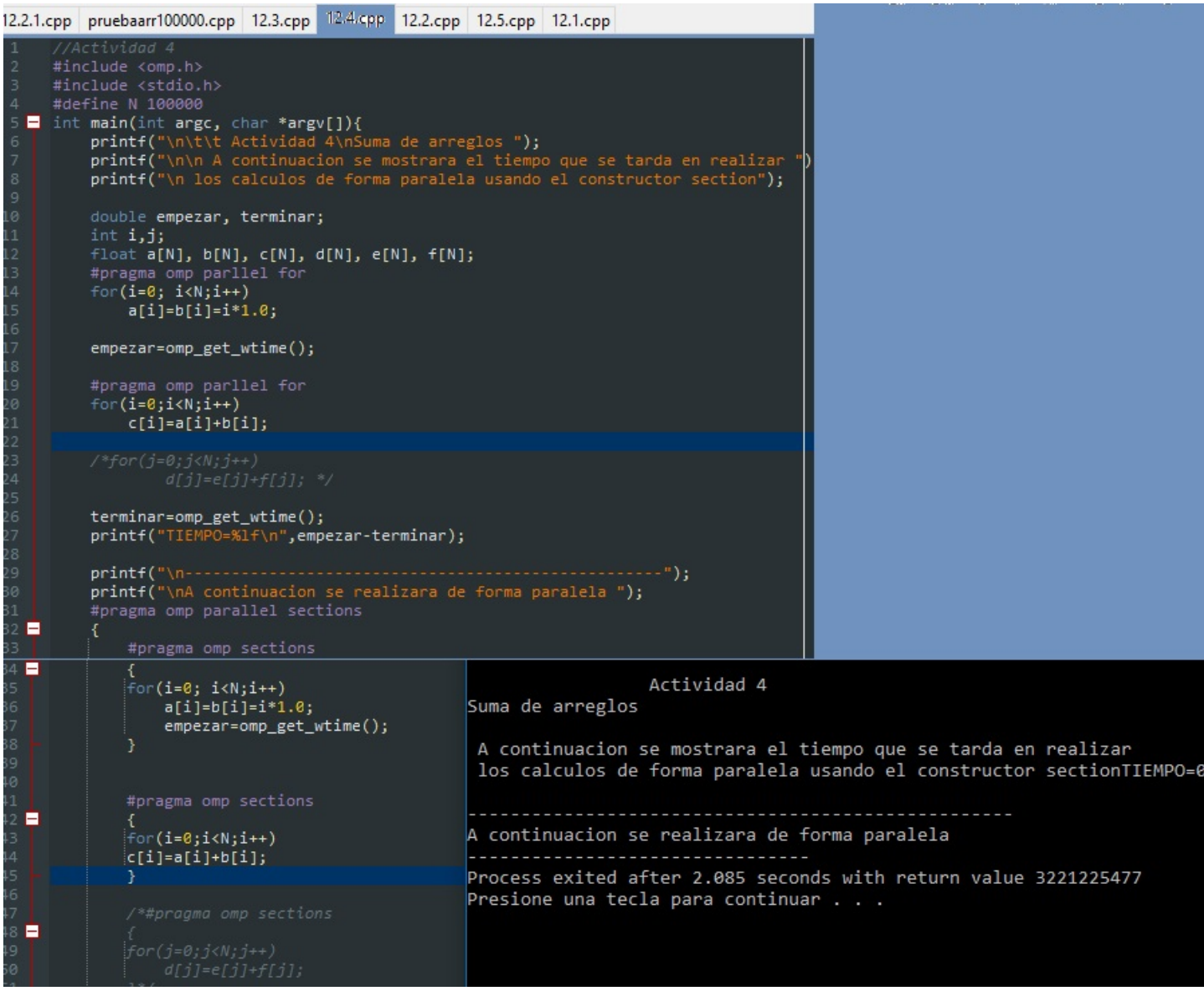
Actividad 3 Obtención del numero Pi utilizando la regla del trapecio

Para esta actividad se realizó la parte secuencial con el código dado en la práctica y se hizo paralelo poniéndole un parallel for antes del ciclo que le da valor a x y en consola podemos ver la siguiente información:



Actividad 4 Obtencion del resultado usando el constructor for y section

Esta actividad no se pude realizar del todo ya que al usar el contructor for esperábamos que se realizara las iteraciones de los ciclos for de manera paralela, pero los resultados se desborraban. Por otro lado al usar el constructor sections se esperaría que cada ciclo se realizara de manera independiente por hilo, sin embargo, no estamos del todo seguros que fue así y después de esto la salida a pantalla nos da a entender que retorna el resultado aunque no se pidió que lo hiciera de esta manera.



Actividad 5

¿Qué sucede si se quita la barrera? Los hilos realizan sus operaciones sin esperar que el hilo maestro acabe de imprimir los elementos del arreglo

Si en lugar de utilizar el constructor master se utilizara single, ¿Qué otros cambios se tienen que hacer en el cogido? Estuvimos checando si era necesario realizar algún cambio, pero no lo encontramos, al parecer el programa funciona correctamente como se ve en la imagen. Cabe destacar que al hacer uso del constructor single todos los hilos tienen que esperar a que termine de imprimir este hilo.

12.2.1.cpp | pruebaarr100000.cpp | 12.3.cpp | [*] 12.4.cpp | 12.2.cpp | 12.5.cpp | 12.1.cpp

```c
//Actividad 5
#include <omp.h>
#include <stdio.h>

int main(){
    int a[5],i;
    #pragma omp parallel
    {
        #pragma omp for
        for (i=0;i<5;i++)
            a[i]=i*i;

        #pragma omp master
            for(i=0; i<5; i++)
                printf("a[%d]= %d\n",i,a[i]);

        #pragma opm barrier

        #pragma omp for
        for(i=0;i<5;i++)
            a[i]+=i;
    }

    printf("-------------------------------
    printf("\nQuitando la barrera\n");

    #pragma omp parallel
    {
        #pragma omp for
        for (i=0;i<5;i++)
            a[i]=i*i;

        #pragma omp master
            for(i=0; i<5; i++)
                printf("a[%d]= %d\n",i,a[i]);

    //  #pragma opm barrier

        #pragma omp for
        for(i=0;i<5;i++)
            a[i]+=i;
    }
    /*Al quitar la barrera los hilos realizan sus operaciones sin fijarse
    si el hilo maestro ya acabo de imprimir */

    printf("------------------------------------------------------------");
    printf("\nUsando el constructor single \n");
        #pragma omp parallel
    {
        #pragma omp for
        for (i=0;i<5;i++)
            a[i]=i*i;

        #pragma omp single
            for(i=0; i<5; i++)
                printf("a[%d]= %d\n",i,a[i]);

    //  #pragma opm barrier

        #pragma omp for
        for(i=0;i<5;i++)
            a[i]+=i;
    }
    /*Al usar el constructor single todos los demas hilos esperan que esté termine
    de imrpimir para continuar, es como si ya tuviera su propia barrera*/

}
```

C:\Users\israe\Desktop\HACK\JAVA\EDA

```
a[0]= 0
a[1]= 1
a[2]= 6
a[3]= 12
a[4]= 20
--------------------------------
Quitando la barrera
a[0]= 0
a[1]= 1
a[2]= 6
a[3]= 12
a[4]= 20
--------------------------------
Usando el constructor single
a[0]= 0
a[1]= 1
a[2]= 4
a[3]= 9
a[4]= 16

--------------------------------
Process exited after 0.115 secc
Presione una tecla para continu
```

1. Conclusiones

Rosario Morales Oscar Daniel

En la realización de está practica se pudo comprender de manera teórica distintas directivas de OpenMp, como lo fue section, master y sus relaciones con las barreras. A pesar de que ya habíamos realizado la practica 13 porque pensamos que esa será más complicada, no logramos realizar esta como se esperaba, pero considero que los conocimientos los pudimos obtener fueron buenos y sirvió para ver de otra manera distintos aspectos que logramos hacer en la practica 13.

Israel Hipolito Mejia Alba

La realización de esta práctica fue un tanto extraña, ya que primero hicimos la 13 pensando que nos tardaríamos más en ella, pero después cuando intentábamos hacer esta nuestros programas marcaban múltiples errores y a pesar de que comprendíamos la parte teórica de la guía de laboratorio, cuando lo queríamos pasar a código se nos complicó mucho y en algunos casos no funcionaba correctamente. De cualquier manera, sirvió para reforzar, aunque sea un poco más nuestros conocimientos, como algo completamente nuevo para mi fue la cláusula reduction.