	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 5

Número de Práctica(s): Guía práctica de estudio 11: Introducción a OpenMP

Integrante(s): MURRIETA VILLEGAS ALFONSO |VALDESPINO MENDIETA JOAQUÍN

Núm. De Equipo de cómputo empleado: 38

Semestre : 2019 - 1

Fecha de entrega: 30 DE OCTUBRE DE 2018

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN A OPENMP

OBJETIVOS DE LA PRÁCTICA

- El estudiante conocerá y aprenderá a utilizar algunas de las directivas de OpenMP utilizadas para realizar programas paralelos.

CONCEPTOS PREVIOS

La entidad software principal en un sistema de cómputo es el proceso, y su contraparte en hardware es el procesador o bien la unidad de procesamiento. En la actualidad tanto los sistemas de cómputo como los dispositivos de comunicación cuentan con varias unidades de procesamiento, lo que permite que los procesos se distribuyan en éstas para agilizar la funcionalidad de dichos sistemas/dispositivos. Esta distribución la puede hacer el sistema operativo o bien los desarrolladores de sistemas de software. Por lo tanto, es necesario entender lo que es un proceso, sus características y sus variantes.

PROCESO

Un proceso tiene varias definiciones, por mencionar algunas:

- programa en ejecución,
- procedimiento al que se le asigna el procesador y
- centro de control de un procedimiento.

Por lo que no solamente un proceso se limita a ser un programa en ejecución. Por ejemplo, al ejecutar un programa éste puede generar varios procesos que no necesariamente están incluidos en un programa o bien partes del código de un mismo programa son ejecutados por diversos procesos. Los procesos se conforman básicamente de las siguientes partes.

- El código del programa o segmento de código (sección de texto).
- La actividad actual representada por el contador de programa (PC) y por los contenidos de los registros del procesador.
- Una pila o stack que contiene datos temporales, como los parámetros de las funciones, las direcciones de retorno y variables locales. También el puntero de pila o stack pointer.
- Una sección de datos que contiene variables globales.
- Un heap o cúmulo de memoria, que es asignada dinámicamente al proceso en tiempo de ejecución.

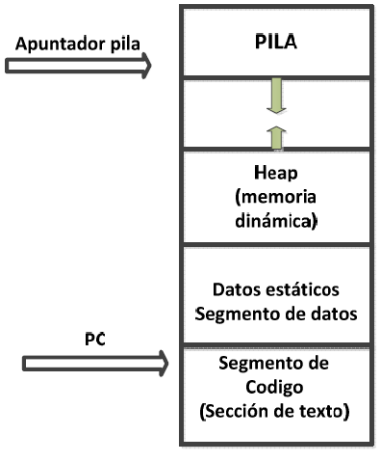


Imagen 1. Estructura en la que se conforman los procesos.

Respecto a los estados de un proceso, se puede encontrar en cualquiera de los siguientes: Listo, en ejecución y bloqueado. Los procesos en el estado listo son los que pueden pasar a estado de ejecución si el planificador los selecciona. Los procesos en el estado ejecución son los que se están ejecutando en el procesador en ese momento dado. Los procesos que se encuentran en estado bloqueado están esperando la respuesta de algún otro proceso para poder continuar con su ejecución (por ejemplo, realizar una operación de E/S).

INTRODUCCIÓN

Un proceso puede ser un programa que tiene solo un hilo de ejecución. Por ejemplo, cuando un proceso está ejecutando un procesador de texto, se ejecuta solo un hilo de instrucciones. Este único hilo de control permite al proceso realizar una tarea a la vez, por ejemplo, el usuario no puede escribir simultáneamente caracteres y pasar al corrector ortográfico dentro del mismo proceso, para hacerlo se necesita trabajar con varios hilos en ejecución, para llevar a cabo más de una tarea de forma simultánea.
Entonces un proceso tradicional (o proceso pesado) tiene un solo hilo de control que realiza una sola tarea y un proceso con varios hilos de control, puede realizar más de una tarea a la vez. Así también se pueden tener varios procesos y cada uno con varios hilos.

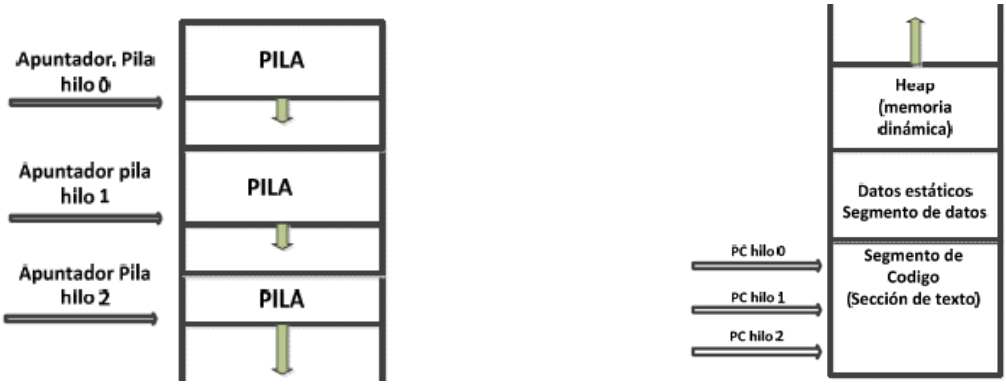


Imagen 2. Estructura en la que se conforman los procesos cuando se llevan a través de hilos

Un proceso es un hilo principal que puede crear hilos y cada hilo puede ejecutar concurrentemente varias secuencias de instrucciones asociadas a funciones dentro del mismo proceso principal. Si el hilo principal

termina entonces los hilos creados por este salen también de ejecución.

Todos los hilos dentro de un proceso comparten la misma imagen de memoria, como el segmento de código, el segmento de datos y recursos del sistema operativo. Pero no comparte el contador de programa (por lo que cada hilo podrá ejecutar una sección distinta de código), la pila en la que se crean las variables locales de las funciones llamadas por el hilo, así como su estado.

CONCURRENCIA VS PARALELISMO

Concurrencia: Es la existencia de varias actividades realizándose simultáneamente y requieren de una sincronización para trabajar en conjunto. Se dice que dos procesos o hilos son concurrentes cuando están en progreso simultáneamente, pero no al mismo tiempo.

Imagen 3. Esquema de cómo se vería de forma abstracta la memoria al llevarse a cabo actividades de forma concurrentes.

Paralelismo: Actividades que se pueden realizar al mismo tiempo. Dos procesos o hilos son paralelos cuando se están ejecutándose al mismo tiempo, para lo cual se requiere que existan dos unidades de procesamiento.

Imagen 4. Esquema de cómo se vería de forma abstracta la memoria al llevarse a cabo actividades de forma paralela.

PROGRAMACIÓN SECUENCIAL, CONCURRENTE Y PARALELA

Cuando se realiza un programa secuencial, las acciones o instrucciones se realizan una tras otra, en cambio en un programa concurrente se tendrán actividades que se pueden realizar de forma simultánea y se ejecutarán en un solo elemento de procesamiento. En un programa paralelo se tienen acciones que se pueden realizar también de forma simultánea, pero se pueden ejecutar en forma independiente por diferentes unidades de procesamiento.

Por lo anterior, para tener un programa paralelo se requiere de una computadora paralela, es decir una computadora que tenga dos o más unidades de procesamiento

Memoria Compartida en Computadora

En el hardware de una computadora, la memoria compartida se refiere a un bloque de memoria de acceso aleatorio a la que se puede acceder por varias unidades de procesamiento diferentes. Tal es el caso de las computadoras *Multicore* donde se tienen varios núcleos que comparten la misma memoria principal.

En software, la memoria compartida es un método de comunicación entre procesos/hilos, es decir, una manera de intercambiar datos entre programas o instrucciones que se ejecutan al mismo tiempo. Un proceso/hilo creará un espacio en la memoria RAM a la que otros procesos pueden tener acceso.

OPENMP (OPEN MULTI-PROCESSING)

Es un interfaz de programación de aplicaciones (API) multiproceso portable, para computadoras paralelas que tiene una arquitectura de memoria compartida. OpenMp está formado:

- Un conjunto de directivas del compilador, las cuales son ordenes abreviadas que instruyen al compilador para insertar ordenes en el código fuente y realizar una acción en particular.
- Una biblioteca de funciones
- Variables de entorno.

que se utilizan para paralelizar programas escritos es lenguaje C, C++ y Fortran.

ARQUITECTURA OPENMP

Para paralelizar un programa se tiene que hacer de forma explícita, es decir, el programador debe analizar e identificar qué partes del problema o programa se pueden realizar de forma concurrente y por tanto se pueda utilizar un conjunto de hilos que ayuden a resolver el problema.

OpenMp trabaja con la llamada arquitectura **fork-join**, donde a partir del proceso o hilo principal se genera un número de hilos que se utilizarán para la solución en paralelo llamada región paralela y después se unirán para volver a tener solo el hilo o proceso principal. El programador especifica en qué partes del programa se requiere ese número de hilos. De aquí que se diga que OpenMp combina código serial y paralelo.

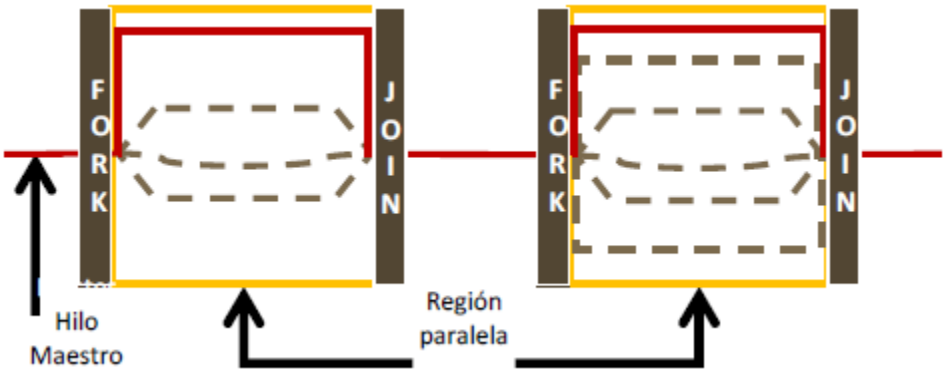


Imagen 5. Esquema de la composición de la arquitectura Fork-Join.

DIRECTIVAS Y PRAGMAS

Agregar una directiva o pragma en el código es colocar una línea como la que sigue

#pragma omp nombreDelConstructor <clausula o clausulas>

Donde se puede observar se tiene los llamados constructores y las cláusulas. Los constructores es el nombre de la directiva que se agrega y las cláusulas son atributos dados a algunos constructores para un fin específico; una cláusula nunca se coloca si no hay antes un constructor.
En la parte del desarrollo de esta guía se irán explicando constructores, cláusulas y funciones de la biblioteca ***omp.h.*** mediante actividades que facilitan la comprensión (Apartado de actividades de la práctica)

ACTIVIDADES DE LA PRÁCTICA (DESARROLLO)

Para la mejor comprensión de la práctica fue necesario plantear el respectivo análisis de cada ejercicio con sus respectivos resultados, a continuación, se muestran uno por uno cada actividad:

NOTAS PREVIAS

Para aprender más sobre openMP en lo siguiente se desarrollan actividades que facilitaran la comprensión. Para el desarrollo de la parte práctica, en esta guía se utilizará el compilador gcc de Linux.

El primer constructor por revisar es el más importante, parallel, el cual permite crear regiones paralelas, es decir generar un número de hilos que ejecutarán ciertas instrucciones y cuando terminen su actividad se tendrá solo al maestro.

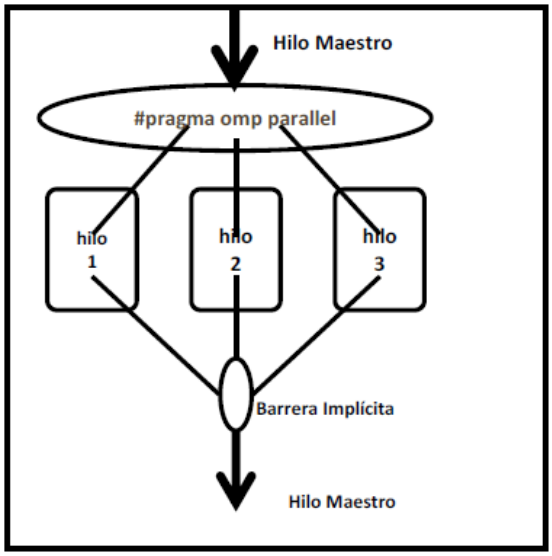


Imagen 5. Esquema del uso de PARALLEL para la creación de hilos.

La sintaxis es como sigue es la siguiente:

```
#pragma omp parallel
{
        //Bloque de código
}
```

NOTA: La llave de inicio de la región debe ir en el renglón siguiente de la directiva, si no se coloca así ocurrirá error.

ACTIVIDAD 0.

Debido a que la biblioteca de OpenMP (Checar el apartado previo de la práctica) corre en Linux, se tuvo que plantear afrontar a un pequeño problema debido a que ninguno de nosotros tenía alguna distribución de Linux, como solución rápido lo que se planteó fue instalar MinGW.

MinGW es una contracción de "Minimalist GNU for Windows", es un entorno de desarrollo minimalista para aplicaciones nativas de Microsoft Windows.

NOTA: Para mayor comodidad, también se empleó como IDE CodeBlocks donde a través del manejo de las variables de entorno y modificando algunos aspectos del compilador (GCC) es como se pudo adaptar totalmente todo para poder trabajar en Windows.

Dicho lo anterior, lo primero que se realizó fue una pequeña prueba probando este entorno a través de la creación y uso de un hilo, a continuación, se muestra la salida probando la biblioteca de omp

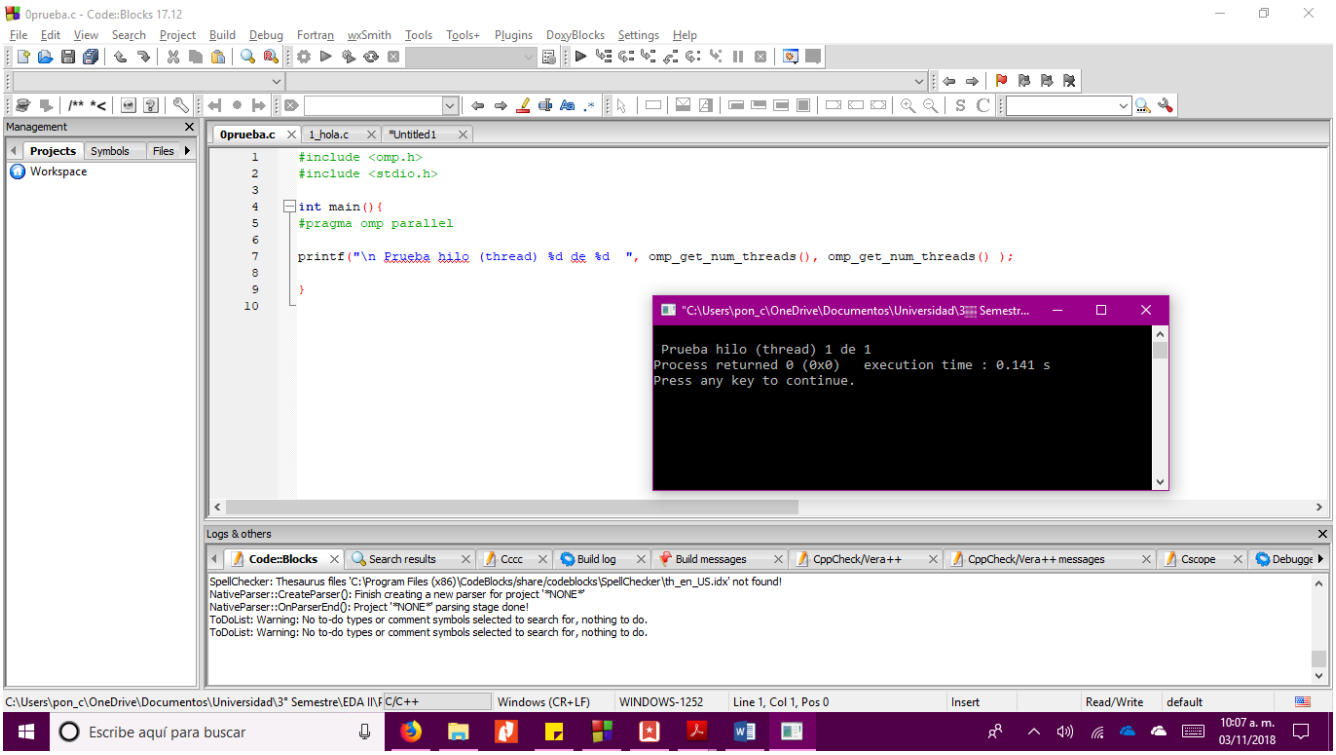


Imagen 1: Salida del programa para comprobar el uso de la biblioteca OpenMP en Windows

ACTIVIDAD 1.

Para esta actividad se realizaron los siguientes pasos:

1) Una vez que se realizó el código en su versión serial, se agregó el constructor parallel desde la declaración de la variable hasta el final de la iteración, a continuación, se muestra la salida del código empleando tanto la versión serial como la versión en paralelo:

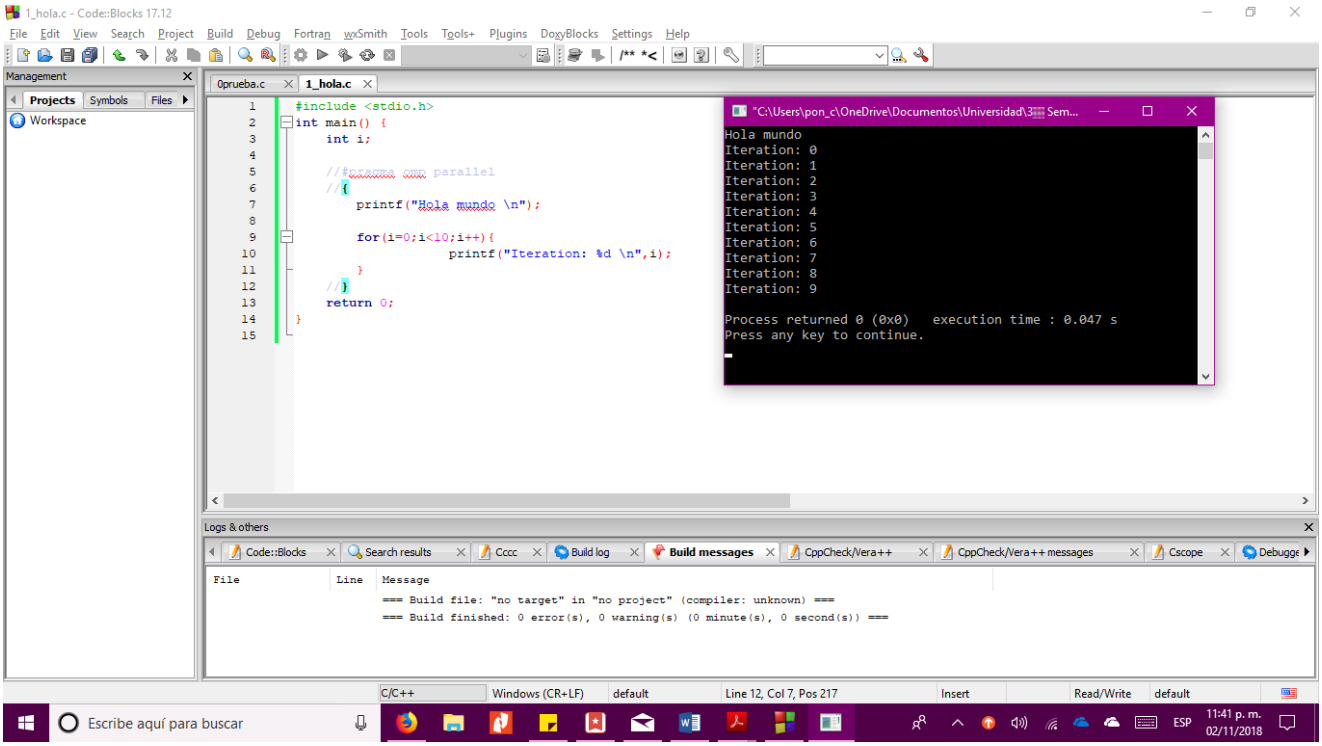


Imagen 2: Salida del programa en su versión serial.

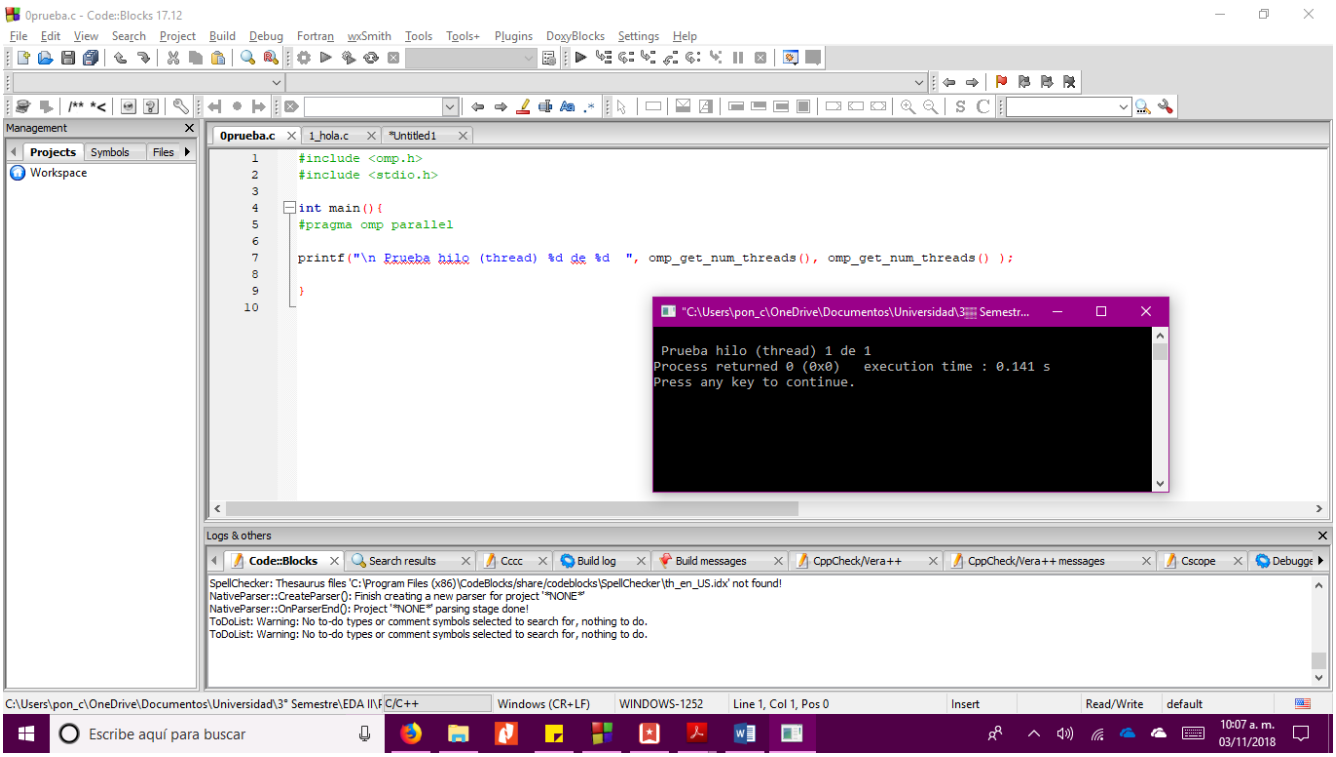


Imagen 3: Salida del programa en su versión paralela.

ANÁLISIS DE LA SALIDA

Realmente la única diferencia que se puede notar es el tiempo en que se lleva a cabo la realización del código en su versión serial y paralela, y esto e s debido a que la forma en que se aborda cada uno de los códigos es muy distinto, sobre todo porque en una se dividen las tareas cada uno de los núcleos del procesador mientras que en la versión serial solamente se lleva a cabo en un procesador.

ACTIVIDAD 2.

En cada región paralela hay un número de hilos generados por defecto y ese número es igual al de unidades de procesamiento que se tengan en la computadora paralela que se esté utilizando, en este caso el número de núcleos que tenga el procesador.

Para este apartado lo que se realizó fue:

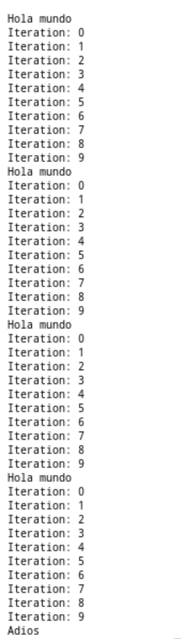
1-Modificar la variable de ambiente OMP_NUM_THREADS desde la consola, de la siguiente forma:
export OMP_NUM_THREADS=4

2- Cambiar el número de hilos a n (un entero llamando a la función omp_set_num_threads(n) que se encuentra en la biblioteca omp.h (hay que incluirla).

3-Agregar la cláusula num_threads(n) seguida después del constructor parallel, esto es:
#pragma omp parallel num_threads (n)

ANÁLISIS DE LA SALIDA

Una vez que se realizó lo anterior, se corrió nuevamente el código de la práctica del apartado anterior, a continuación, se muestra la salida correspondiente a este apartado.



```
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Adios
```

Imagen 4: Salida del programa mostrando cada uno de los resultados realizados por cada hilo

Como se puede ver por cada uno de los Hilos se realizó la ejecución general del programa, al declarar n igual a 4 es por ello que se ve todo lo ingresado dentro del apartado de parallel 4 veces impreso.
(Cada hilo realizó las mismas actividades).

ACTIVIDAD 3.

En la programación paralela en computadoras que tienen memoria compartida puede presentarse la llamada condición de carrera (**race condition**) que ocurre cuando varios hilos tienen acceso a recursos compartidos sin control. El caso más común se da cuando en un programa varios hilos tienen acceso concurrente a una misma dirección de memoria (variable) y todos o algunos en algún momento intentan escribir en la misma localidad al mismo tiempo. Esto es un conflicto que genera salidas incorrectas o impredecibles del programa.

En OpenMP al trabajar con hilos se sabe que hay partes de la memoria que comparten entre ellos y otras no. Por lo que habrá variables que serán compartidas entre los hilos, (a las cuales todos los hilos tienen acceso y las pueden modificar) y habrá otras que serán propias o privadas de cada uno.

Dentro del código se dirá que cualquier variable que esté declarada fuera de la región paralela será compartida y cualquier variable declarada dentro de la región paralela será privada.
Del código que se está trabajando, sacar de la región paralela la declaración de la variable entera, compilar y ejecutar el programa varias veces.

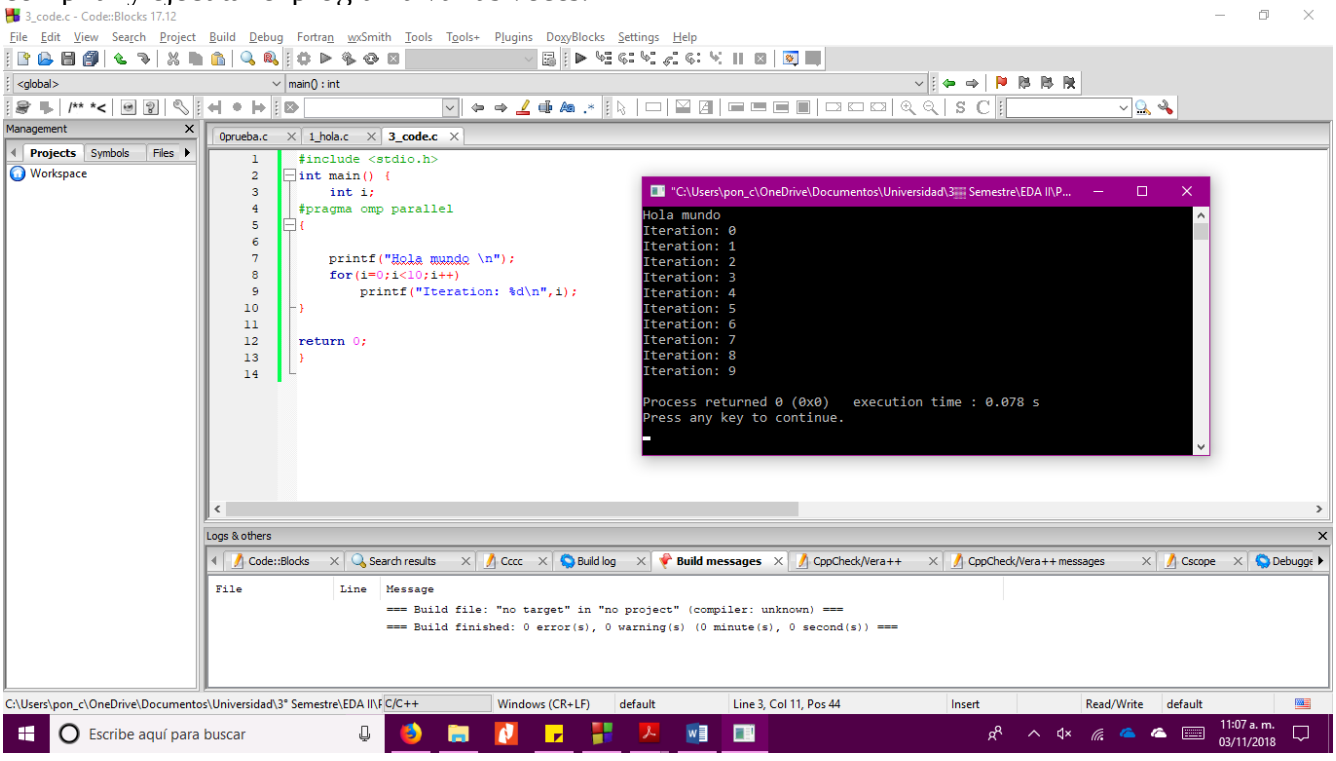


Imagen 5: Salida del programa en su versión paralela donde se puede ver la salida de forma ordenada.

ANÁLISIS DE LA SALIDA

NOTA: Realmente para este apartado no se vio algún cambió o diferencia respecto a la actividad 1, obvio respecto a la actividad 2 sí debido a que no se declararon más hilos.

Tal vez la principal diferencia que se quería hacer notar respecto a la actividad 1 era que en este caso la salida no iba a estar desordena y como previamente se mencionó, esto se debía a que al declarar la variable i fuera del apartado de parallel ninguno de los hilos podía acceder accidentalmente a la dirección de memoria de la variable.

Sin embargo, el llegar a una misma localidad de memoria en muchos casos y como menciona la práctica es poco probable.

NOTA IMPORTANTE:

A partir de las siguientes actividades debido a que no se veían cambios algunos en el entorno *MinGW*, es por ello que se decidió trabajar directamente en una distribución de Linux que fue Ubuntu 18.10

ACTIVIDAD 4.

Existen dos cláusulas que pueden forzar a que una variable privada sea compartida y una compartida sea privada y son las siguientes:

shared(): Las variables colocadas separadas por coma dentro del paréntesis serán compartidas entre todos
los hilos de la región paralela. Sólo existe una copia, y todos los hilos acceden y modifican dicha copia.

private(): Las variables colocadas separadas por coma dentro del paréntesis serán privadas. Se crean p copias, una por hilo, las cuales no se inicializan y no tienen un valor definido al final de la región paralela ya que se destruyen al finalizar la ejecución de los hilos.

En este apartado se pidió agregar al código resultante de la actividad 3, la cláusula private( ) después del constructor parallel y colocar la variable de la siguiente forma

```
#pragma omp parallel private(i)
```

ANÁLISIS DE LA SALIDA

```
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Hola mundo
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Adios
```

Imagen 6: Salida del programa

Nuevamente en este apartado no se volvieron a ver realmente cambios respectos a cambios anteriores, sin embargo, lo que trató de puntualizar este apartado es que a través del uso de "private" es como se puede abordar la solución al hecho de que al emplear hilos todo lo que se maneja se pierde una vez que se termina el uso del hilo.

ACTIVIDAD 5.

Dentro de una región paralela hay un número de hilos generados y cada uno tiene asignado un identificador. Estos datos se pueden conocer durante la ejecución con la llamada a las funciones de la biblioteca omp_get_num_threads() y omp_get_thread_num() respectivamente.

Para este apartado se pidió probar y notar un código de la práctica donde para su buen funcionamiento se tuvo que indicar que la variable fuera privada dentro de la región paralela, ya que de no ser así todos los hilos escribirían en la dirección de memoria asignada a dicha variable sin un control (**racecondition**), es decir "competirán" para ver quién llega antes y el resultado visualizado puede ser inconsistente e impredecible.

ANÁLISIS DE LA SALIDA

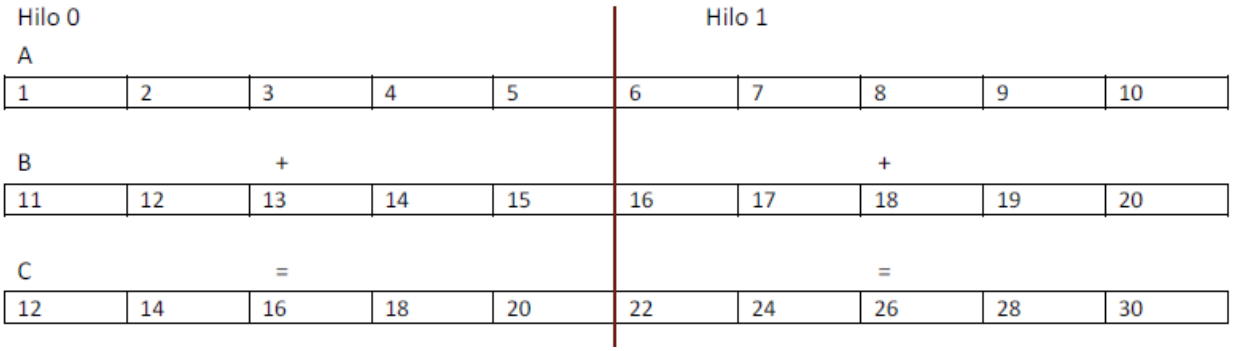
```
Hola Mundo desde el hilo 0 de un total de 2
Hola Mundo desde el hilo 1 de un total de 2
Adios
```

Imagen 7: Salida del programa donde se emplearon dos métodos de OpenMP

Como se puede apreciar en la salida anterior, es la salida de un Hola mundo desde distintos Hilos lo cual demuestra como a través del uso de cada uno de ellos se puede realizar una tarea en específico, pero además y como apartado más importante es la importante de que cada Hilo tiene un identificador asignado, esto con el propósito de poder saber a qué estamos asignan cada tarea.

ACTIVIDAD 6.

Para esta actividad se requirió realizar la suma de dos arreglos unidimensionales de 10 elementos de forma paralela utilizando solo dos hilos. Para ello se utilizará un paralelismo de datos o descomposición de dominio, es decir, cada hilo trabajará con diferentes elementos de los arreglos a sumar y, pero ambos utilizarán el mismo algoritmo para realizar la suma.



Partiendo de la versión serial donde la suma se realiza de la siguiente manera (se asume que y ya tienen valores para ser sumados):

for(i=0; i<10; i++)
C[i]=A[i]+B[i]

6.1 VERSIÓN SERIAL

A continuación, se muestra la salida de la versión serial de esta actividad:

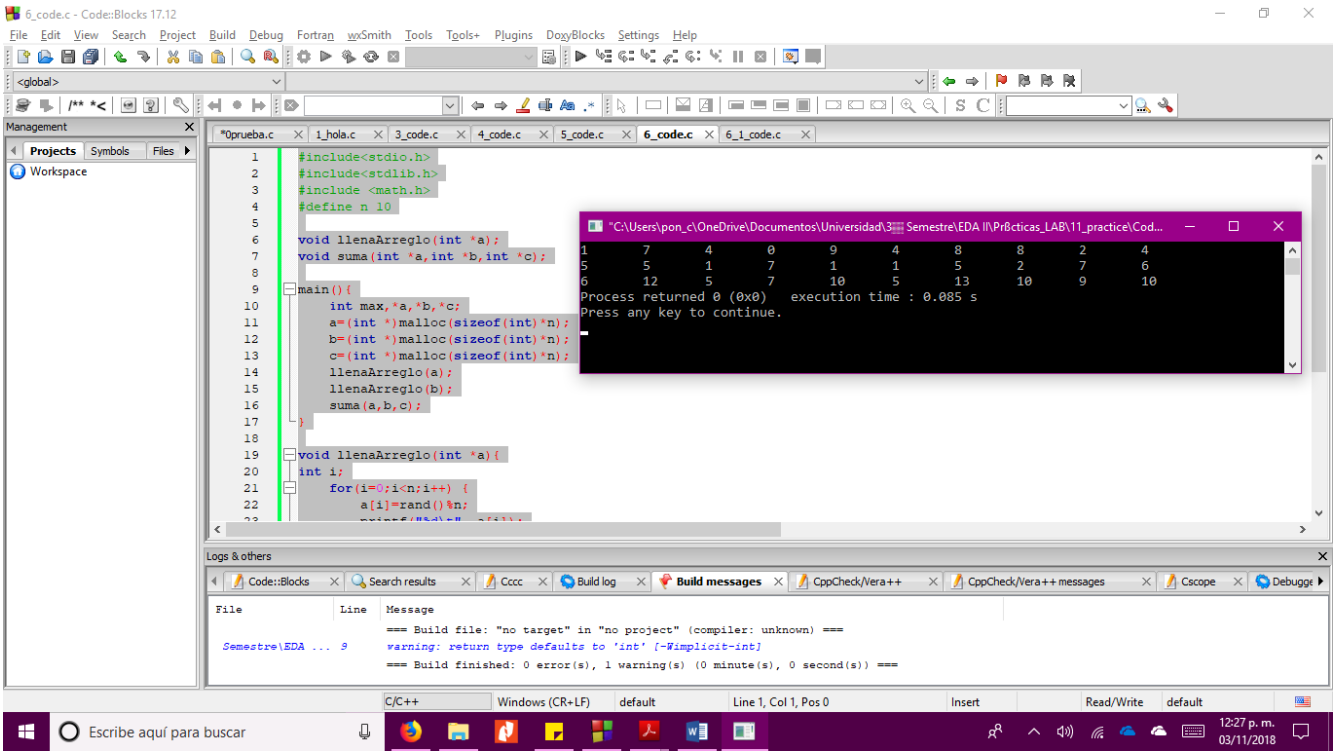


Imagen 8: Salida del programa en su versión serial (Realizada en Windows)

6.2 VERSIÓN EN PARALELO

Para la versión paralela, el hilo 0 sumará la primera mitad de A con la primera de B y el hilo 1 sumará la segunda mitad de A con la segunda de B. Para conseguir esto cada hilo realizará las mismas instrucciones, pero utilizará índices diferentes para referirse a diferentes elementos de los arreglos, entonces cada uno iniciará y terminará el índice en valores diferentes

```
3       6       7       5       3       5       6       2       9       1
2       7       0       9       3       6       0       6       2       6
hilo 0 calculo C[0]= 5
hilo 0 calculo C[1]= 13
hilo 0 calculo C[2]= 7
hilo 0 calculo C[3]= 14
hilo 0 calculo C[4]= 6
hilo 1 calculo C[5]= 11
hilo 1 calculo C[6]= 6
hilo 1 calculo C[7]= 8
hilo 1 calculo C[8]= 11
hilo 1 calculo C[9]= 7
```

Imagen 9: Salida del programa en su versión en paralelo

ANÁLISIS DE LA SALIDA

Realmente creo esta actividad es la más importante sobre todo por que destaca el hecho de que se pueden realizar cálculos o procesos tanto en su versión serial como paralela.
Como se puede ver en la imagen 6 toda la salida sale directa debido a que solamente se lleva por parte de un procesador, sin embargo, por el otro lado en la versión paralela se puede notar que la salida es la misma en cuanto a resultado sin embargo esta fue llevada a cada mediante dos hilos distintos.

ACTIVIDAD 7.

Otro constructor es el for, el cual divide las iteraciones de una estructura de repetición for. Para utilizarlo se debe estar dentro de una región paralela. Su sintaxis es:

```
#pragma omp parallel
{
…..
#pragma omp for
      for(i=0;i<12;i++) {
           Realizar Trabajo();
      }
…..
}
```

La variable índice de control i se hará privada de forma automática. Esto para que cada hilo trabaje con su propia variable i. Este constructor se utiliza comúnmente en el llamado paralelismo de datos o descomposición de dominio, lo que significa que, cuando en el análisis del algoritmo se detecta que varios hilos pueden trabajar con el mismo algoritmo o instrucciones que se repetirán, pero sobre diferentes datos y no hay dependencias con iteraciones anteriores. Por lo anterior, no siempre es conveniente dividir las iteraciones de un ciclo for.

Para este apartado se pidió modificar el código de la actividad 1, de manera que se dividieran las iteraciones de la estructura de repetición for, a continuación, se muestra la salida del programa:

```
Hola mundo
Iteration:0
Iteration:1
Iteration:2
Iteration:3
Iteration:4
Hola mundo
Iteration:5
Iteration:6
Iteration:7
Iteration:8
Iteration:9
Adios
```

Imagen 10: Salida del programa en su versión en paralelo

ANÁLISIS DE LA SALIDA

Como se puede ver en la imagen 10, el uso del constructor for en paralelo nos brinda la oportunidad de poder realizar cierta cantidad de iteración en un hilo distinto es por ello que por ejemplo para poder destacar que fue llevado a través de su forma en paralelo se dejó el apartado de la impresión "Hola mundo" dentro del parallel para así destacar que iteraciones fueron realizadas en un hilo y cuales en otro.

ACTIVIDAD 8.

Para esta actividad se pidió la suma de dos arreglos unidimensionales de n elementos de forma paralela, utilizando los hilos por defecto que se generen en la región paralela y el constructor for.

Como se explicó en la actividad 6 los hilos realizarán las mismas operaciones, pero sobre diferentes elementos del arreglo y eso se consigue cuando cada hilo inicia y termina sus iteraciones en valores diferentes, para referirse a diferentes elementos de los arreglos A y B. Esto lo hace el constructor for, ya que al dividir las iteraciones cada hilo trabaja con diferentes valores del índice de control.

A continuación, se muestra la salida del programa:

```
3       6       7       5       3       5       6       2       9       1
2       7       0       9       3       6       0       6       2       6
hilo 0 calculo C[0]= 5
hilo 1 calculo C[5]= 11
hilo 0 calculo C[1]= 13
hilo 0 calculo C[2]= 7
hilo 0 calculo C[3]= 14
hilo 0 calculo C[4]= 6
hilo 1 calculo C[6]= 6
hilo 1 calculo C[7]= 8
hilo 1 calculo C[8]= 11
hilo 1 calculo C[9]= 7
```

Imagen 11: Salida del programa empleando el uso del construtor for en su forma paralela

ANÁLISIS DE LA SALIDA

Para esta actividad lo primero que se puede ver es que la salida respecto a la versión serial obtenida en la actividad 6 y en su versión en paralelo, pero sin el uso del constructor for es la misma, lo cual nos da a entender que realmente el uso del constructor for en su versión paralela realmente nos puede dar ventajas enormes al segmentar cada operación – iteración en un hilo distinto en vez de realizarlo seriamente como comúnmente se realizaría en cualquier programa introductorio de C

CONCLUSIONES

MURRIETA VILLEGAS ALFONSO

A lo largo de las distintas actividades propuestas por el manual de prácticas se conoció una de las interfaces de programación de aplicaciones (API) para la programación multiproceso de memoria compartida a través del lenguaje de programación C, C++ y Fortran, conocida como **OpenMP**.

OpenMP se basa en el modelo **fork-join**, el cual es un paradigma que proviene de los sistemas basados en Unix, donde una tarea muy pesada o larga de realizar mediante su versión serial se divide en n hilos (fork) de manera que cada uno tenga un menor peso, para posteriormente juntar los resultados obtenidos por cada hilo y posteriormente unirlos en un solo resultado (join).

Open MP es sin duda una buena herramienta para el desarrollo de programas que necesariamente necesitan abordarse mediante el paradigma de programación en paralelo, además de que como se vieron en las distintas actividades de esta práctica, el buen manejo de los hilos nos proporciona los mismos

resultados que en la versión serial, pero con la ventaja de aprovechar toda la capacidad que nos ofrecen nuestras computadoras que comúnmente ya tienen más de un solo procesador.

Por último, realmente el paradigma de programación en paralelo es sin duda un aspecto que suena realmente interesante y sobre todo tiene su fama por todo el contexto que tiene tanto histórico como de desarrollo.

VALDESPINO MENDIETA JOAQUÍN

En conclusión, en esta práctica se han cumplido los objetivos, dado el seguimiento del manual, realmente como se ha observado cambiar de paradigma es lo más difícil de la programación, sin embargo la herramienta OpenMP permite el uso de la memoria y los procesos múltiples de manera más directa, aunque en un inicio ver un programa paralelizado es complicado al analizarlo puede ser algo más práctico, dando así que para obtener la lógica de paralelización o la implementación es lo más laborioso y complejo, ya que hay restricciones para no caer en error y uno de ellos es que las tareas divididas sean completamente independientes, ya que un proceso al terminar la tarea solo se encargara de enviar su resultado, si depende de otra tarea esta quedara en espera lo cual podría generar un error, aparte de que se necesitan algunas características en el sistema ,para poder ejecutar las sentencias en el codigo que permite OpenMP, en fin la programación paralela puede llegar ser de utilidad para obtener resultados de manera más rápida, sin embargo llegar al algoritmo es una desventaja.

REFERENCIAS

(2007). Introduction to parallel programming, Intel Software collage.

Galvin Baer Peter, Gagne Greg . *Fundamentos de Sistemas operativos*. MacGraw Hill.

Elba Karen Saenz García. (2017). *Manual de Prácticas del laboratorio de Estructuras de Datos y algoritmos 1I.* UNAM, Facultad de Ingeniería.