Estudiante: Esteban Calvo Vargas
Carné: 201270170
Taller 1

Preguntas teóricas

1. ¿Qué es OpenMP?
OpenMP es una especificación dada para un conjunto de directivas de compilador, rutinas de librería y variables de ambiente que pueden ser usadas para especificar paralelismo de alto nivel en Fortran y programas en C/C++. OpenMP viene del inglés Open Multi-Processing.
Obtenido de: https://www.openmp.org/about/openmp-faq/#WhatIs

2. ¿Cómo se define una región paralela en OpenMP utilizando pragmas?
El formato para la definición de bloque paralelo de código es el siguiente:

Fortran	**!$OMP PARALLEL *[clause ...]*** **IF *(scalar_logical_expression)*** **PRIVATE *(list)*** **SHARED *(list)*** **DEFAULT (PRIVATE \| FIRSTPRIVATE \| SHARED \| NONE)** **FIRSTPRIVATE *(list)*** **REDUCTION *(operator: list)*** **COPYIN *(list)*** **NUM_THREADS *(scalar-integer-expression)*** ***block*** **!$OMP END PARALLEL**
C/C++	**#pragma omp parallel *[clause ...] newline*** **if *(scalar_expression)*** **private *(list)*** **shared *(list)*** **default (shared | none)** **firstprivate *(list)*** **reduction *(operator: list)*** **copyin *(list)*** **num_threads *(integer-expression)*** ***structured_block***

La regla básica es escribir la directiva de compilador antes de la instrucción o bloque estructurado de código que se quiere paralelizar.
Siguiendo el código de C/C++ en el formato inferior, se puede dividir la directiva en las siguientes partes:

- #pragma omp: indica que se utilizará una directiva de compilador de OpenMP.

- parallel: directiva que indica que el siguiente bloque de código debe ser ejecutado de forma paralela.

El resto de la directiva indica diferentes cláusulas a utilizar durante la ejecución paralela.
Obtenido de:

https://computing.llnl.gov/tutorials/openMP/#Directives

3. ¿Cómo se define la cantidad de hilos a utilizar al paralelizar usando OpenMP?
La cantidad de hilos se determina a partir de los siguientes factores:

- La evaluación de la cláusula if.
- Utilizando la cláusula num_threads.
- Utilizando la función de librería omp_set_num_threads().
- Estableciendo la variable de ambiente OMP_NUM_THREADS.
- La implementación por defecto usualmente elige el número de CPUs disponible, pero esto puede ser dinámico.

Obtenido de::

https://computing.llnl.gov/tutorials/openMP/#Directives

4. ¿Cómo se compila un código fuente c para utilizar OpenMP y qué encabezado debe incluirse?
Para compilar un programa con directivas OpenMP en C utilizando GCC se debe añadir la opción -fopenmp a la hora de compilar el programa. Además, se debe incluir el encabezado <omp.h> si se quieren utilizar las funciones de la librería.
Obtenido de:

https://gcc.gnu.org/onlinedocs/libgomp/Enabling-OpenMP.html
https://www.ibm.com/developerworks/aix/library/au-aix-openmp-framework/index.html

5. ¿Cómo maneja OpenMP la sincronización entre hilos y por qué esto es importante?
La sincronización de hilos en OpenMP se puede realizar por medio de directivas que indican diversas formas en las que se ejecuta un código. Algunos ejemplos son:

- master: indica que el bloque de código solo es ejecutado por el hilo maestro, el resto lo ignora.
- critical: indica que una región de código solo puede ser ejecutada por un hilo a la vez.
- barrier: indica un punto en el que los hilos esperan a que se termine la ejecución de todos los demás hilos para continuar.
- taskwait: indica un tiempo de espera desde que se inició la ejecución de la tarea actual.
- atomic: indica que la ejecución de la tarea se debe realizar en un solo paso, como la directiva *critical* pero para operaciones pequeñas.

La importancia de la sincronización es que evita que se escriban variables a memoria en un orden no controlado. De esta forma, se pueden evitar situaciones en las que variables utilizan valores escritos por otros hilos y llevar a una operación indeseada.
Obtenido de:

https://computing.llnl.gov/tutorials/openMP/#Synchronization_Constructs

Ejercicios

Pi serial

1. Analice la implementación del código y detecte qué sección del código podría paralelizarse por medio de la técnica de multihilo.
Se podría paralelizar el ciclo for, nada más. Esto debido a que la forma discreta de realizar el cálculo es por medio de una sumatoria, la cual es una suma de pequeños cálculos dados por cada "paso". En este caso, cada pequeño cálculo puede realizarse por separado en un hilo.

2. Con respecto a las variables de la aplicación (dentro del código paralelizable) ¿cuáles deberían ser privadas y cuáles deberían ser compartidas? ¿Por qué?
Las variables que deben ser privadas son aquellas que el resto de hilos no requiera tener acceso para ejecutar su tarea correctamente. Así, solamente x como resultado de cada cálculo pequeño debe ser una variable privada.
Las variables compartidas son las que si no fueran compartidas, el código no realizaría la función para la que originalmente se diseñó de forma serial. En este caso, sería las variables num_steps debido a que todos los hilos necesitan saber cuándo terminar el ciclo, la variable step que debe ser utilizada por todos los hilos en cada cálculo y la variable sum a la que todos los hilos deben agregar sus subresultados almacenados en x.
No es necesario que la variable i sea privada debido a que OpenMP asigna un valor distinto de i a cada hilo en cada iteración del ciclo.

3. Realice la compilación del código, utilizando el método requerido para aplicaciones de OpenMP.
El código se compiló utilizando el siguiente comando:

```
$ gcc -fopenmp pi.c -o pi
```

4. Ejecute la aplicación. Realice un gráfico de tiempo para al menos 4 números de pasos distintos (iteraciones para cálculo del valor de pi).

Figura 1. Tiempo de ejecución con respecto a la cantidad de pasos para el cálculo del número pi de forma serial.

Pi paralelo

1. Analice el código dado. ¿Cómo se define la cantidad de hilos a ejecutar? ¿Qué funcionalidad tiene el pragma omp single? ¿Qué función realiza la línea: #pragma omp for reduction(+:sum) private(x).

- El número de hilos se define como el doble de la cantidad de procesadores disponibles. Esto se ve en la línea 50 del programa donde se multiplica por 2 la cantidad de procesadores.
- omp single indica que la sección de código debe ser ejecutada solamente por 1 hilo. En este caso, se asigna un solo hilo que imprime la cantidad de hilos que se están ejecutando. Obtenido de: https://msdn.microsoft.com/en-us/library/9kcw2kxz.aspx
- El "for" indica que el siguiente bloque de código se debe ejecutar como un ciclo for paralelo. El "reduction(+:sum)" indica que al final de la ejecución completa del ciclo, la lista de resultados se debe reducir en una sola variable (sum) por medio de una reducción de la operación de suma (se puede ver como que se suman todos los resultados). Por último, "private(x)" indica que cada hilo debe tener su propia variable x y hacerla privada. Obtenido de:
    - https://msdn.microsoft.com/en-us/library/88b1k8y5.aspx
      https://msdn.microsoft.com/en-us/library/6z19s8e0.aspx
      https://msdn.microsoft.com/en-us/library/c3dabskb.aspx

2. Realice la compilación del código.
El código se compiló de la misma forma que el código serial:

```
$ gcc -fopenmp pi_par.c -o pi_par
```

3. Ejecute la aplicación. Realice un gráfico con tiempo de ejecución para las diferentes cantidades de hilos mostradas en la aplicación. Compare el mejor resultado con la cantidad de procesadores de su sistema. Aumente aún más la cantidad de hilos. Explique por qué, a partir de cierta cantidad de hilos, el tiempo aumenta.

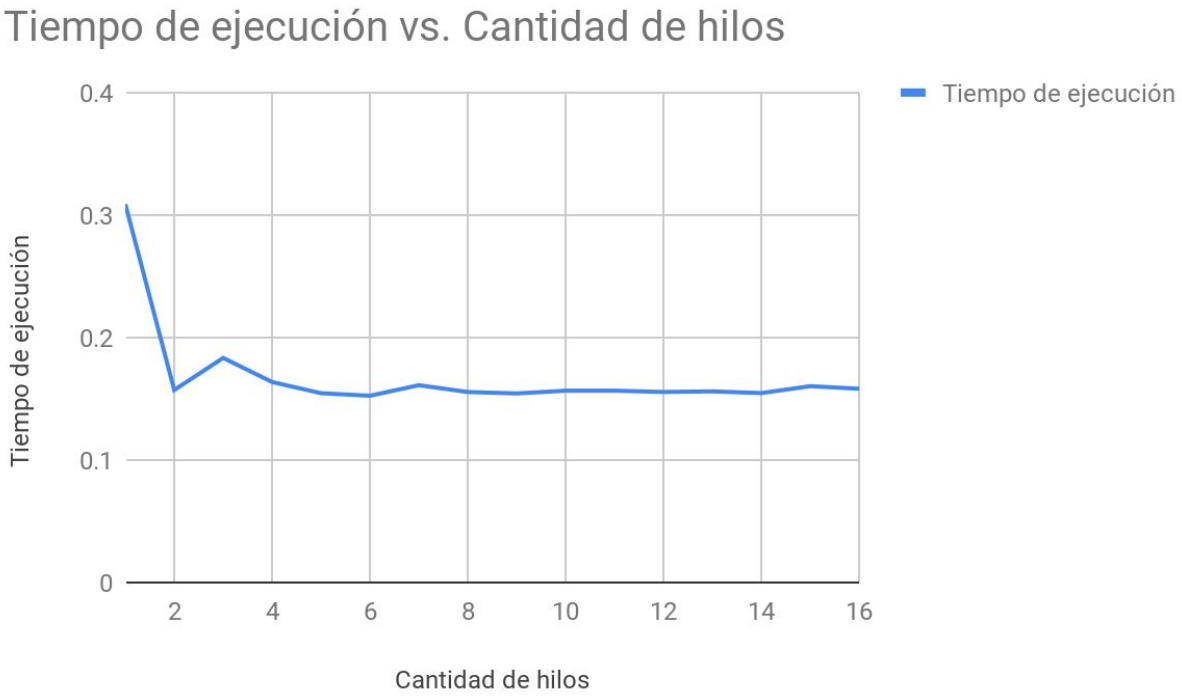

Figura 2. Tiempo de ejecución contra la cantidad de hilos para el cálculo del número pi de forma paralela.

En la figura 2 se puede observar que el tiempo de ejecución no varía significativamente conforme aumenta la cantidad de hilos. Debido a esto, se incluyen los resultados cuantitativos que se pueden observar en la tabla 1. El menor tiempo registrado se da con 6 hilos, sin embargo la variación entre hilos es insignificante y luego de varias ejecuciones del código, el número de hilos para el que se registra el menor tiempo no concuerda siempre con estos resultados.
La computadora en la que se ejecutó el código tiene 4 procesadores y se están utilizando todos, sin embargo, a partir de 2 hilos, la computadora no tiene problemas para ejecutarse en el menor tiempo posible.

Tabla 1. Cantidad de hilos ejecutados contra su tiempo de ejecución en segundos para el cálculo del número pi.

Cantidad de hilos	Tiempo de ejecución
1	0.308946
2	0.15719
3	0.18335
4	0.163697
5	0.154548
6	0.152482
7	0.16108
8	0.155527
9	0.154373
10	0.156588
11	0.156637
12	0.155562
13	0.156047
14	0.154672
15	0.160293
16	0.158177

Además, el código se ejecutó hasta tener 128 hilos, pero el tiempo de ejecución se mantuvo constante, sin variaciones significativas. No se realizaron pruebas con más cantidad de hilos.

Ejercicios prácticos

Todos los ejercicios prácticos se pueden encontrar en el repositorio de Github: https://github.com/Estebanelm/TalleresArqui2
En el caso de este taller, la carpeta Taller1 contiene los ejercicios. Los archivos son los siguientes:

- saxpyser.c: código en c de la operación SAXPY de forma serial.
- saxpypar.c: código en c de la operación SAXPY de forma paralela.
- complexoperation.c: código en c de una aproximación con serie de Fourier de la función f(x)=1.

1. Realice un programa que aplique la operación SAXPY tanto serial (normal) como paralelo (OpenMP), para al menos tres tamaños diferentes de vectores. Mida y compare el tiempo de ejecución entre ambos.

- Para el caso de SAXPY serial, los resultados se pueden observar en la tabla 2.
- Para el caso de SAXPY paralelo, los resultados se pueden observar en la tabla 3.
- Para el caso paralelo se utilizaron 4 hilos. Como se puede observar de las tablas, la ejecución en paralelo es significativamente más rápida. A pesar de ser más rápida y utilizar hasta 4 hilos, el tiempo se reduce a tan solo un poco más de la mitad del tiempo de forma serial. El procesamiento en paralelo nunca logra un rendimiento que permite dividir el tiempo de una tarea serial entre el número de procesadores.

Tabla 2. Tiempo de ejecución de la operación SAXPY con una ejecución serial.

Cantidad de elementos	Tiempo de ejecución
1000000	0.003241
10000000	0.026218
100000000	0.266415

Tabla 3. Tiempo de ejecución de la operación SAXPY con una ejecución paralela.

Cantidad de elementos	Tiempo de ejecución
1000000	0.001832
10000000	0.018021
100000000	0.166377

2. El programa implementado corresponde a la aproximación por series de Fourier de la función f(x) = 1. Idealmente, el programa debe dar la misma respuesta para toda entrada y la aproximación debe ser más cercana a 1 conforme aumenta la cantidad de senos calculados.