Arquitectura de Computadores II

Ing. M.Sc. Jeferson González Gómez

Taller 1: OpenMP

Santiago Vargas de Kruijf

2013050807

1. ¿Que es OpenMP?

OpenMP es un API que nos provee con una serie de funcionalidades para el desarrollo de aplicaciones con ejecución paralela, se tiene soporte para computadores de todos los tamaños, sus herramientas facilitan la programación multiproceso de memoria compartida.

1. ¿Como se define una región paralela en OpenMP utilizando pragmas?

double result,fresult,gresult,hresult;  
#pragma omp parallel  
{ int num = omp\_get\_thread\_num();  
 if (num==0) fresult = f(x);  
 else if (num==1) gresult = g(x);  
 else if (num==2) hresult = h(x);  
}  
result = fresult + gresult + hresult;

Con la definición del pragma le indicamos al programa que utilice openMP para paralelizar esta sección del código.

1. ¿Como se define la cantidad de hilos a utilizar al paralelizar usando OpenMP?

Se utiliza el siguiente comando:

OMP\_SET\_NUM\_THREADS(n)

1. ¿Como se compila un código fuente c para utilizar OpenMP y qu ́e encabezado debe Incluirse?

Se utiliza el siguiente comando:

gcc -fopenmp <filename>.c

1. ¿Como maneja OpenMP la sincronización entre hilos y por que esto es importante?

OpenMP provee una serie de llamados con los cuales sincroniza los hilos, entre ellos están:

1. Reduction: Permite sumar los resultados de los hilos y sumarlos y almacenar su resultado en una variable final.
2. Master: Permite delimitar un bloque como bloque principal.
3. Critical: Permite definir una Región Crítica en el código.
4. Schedule: Nos permite asignar manualmente las iteraciones de los hilos.
5. Barrier: Permite frenar el avance de algunos hilos para sincronizar su ejecución.

Pi Serial

1. Analice la implementación del código y detecte que sección del código podrıa paralelizar

por medio de la técnica de multihilo.

Es posible paralelizar la sección iterativa compuesta por el *for,* esto haría que se paraleliza en cada ejecución.

1. Con respecto a las variables de la aplicación (dentro del código paralelizable) ¿cuales deberıan ser privadas y cuales deberıan ser compartidas? ¿Por que?

Los valores independientes a la iteración pueden ser privados como lo es “X”, los valores compartidos como es el caso de “sum” el cual se comparte para el almacenamiento del valor final.

1. Realice la compilación del código, utilizando el método requerido para aplicaciones de OpenMP.

Podemos compilar utilizando el llamado normal de GCC pero importando openMP

gcc -fopenmp pi.c

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

Iteraciones utilizadas.

* 3.141593 in 1.051340 seconds
* 3.141593 in 1.055206 seconds
* 3.141593 in 1.051959 seconds
* 3.141593 in 1.052044 seconds
* 3.141593 in 1.051459 seconds
* 3.141593 in 1.049232 seconds

Pi Paralelo

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

* El número de hilos se utiliza usando “omp\_set\_num\_threads(N), N: Número de threads.
* #pragma omp single define qué sección de código solo puede ser utilizada por un hilo a la vez.
* #pragma omp for reduction(+:sum) private(x) define “X” como una variable privada y única para cada hilo, “sum” es definida como una variable compartida.

1. 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 que, a partir de cierta cantidad de hilos, el tiempo aumenta.

num\_threads = 1

pi is 3.141593 in 0.927060 seconds and 1 threads

num\_threads = 2

pi is 3.141593 in 0.466604 seconds and 2 threads

num\_threads = 3

pi is 3.141593 in 0.361636 seconds and 3 threads

num\_threads = 4

pi is 3.141593 in 0.279834 seconds and 4 threads

num\_threads = 5

pi is 3.141593 in 0.314453 seconds and 5 threads

num\_threads = 6

pi is 3.141593 in 0.284909 seconds and 6 threads

num\_threads = 7

pi is 3.141593 in 0.283350 seconds and 7 threads

num\_threads = 8

pi is 3.141593 in 0.280333 seconds and 8 threads

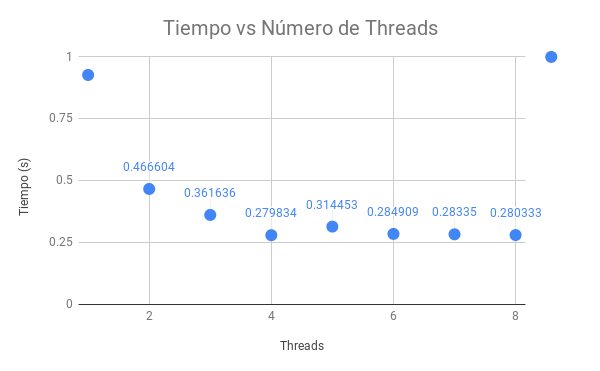


Gráfico 1. Tiempo vs Número de Threads, 8 hilos

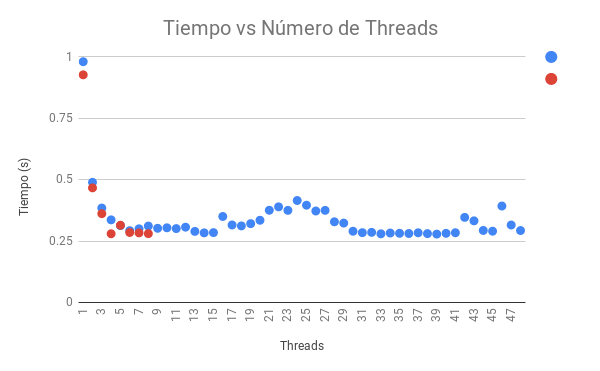


Gráfico 2. Tiempo vs Número de Threads, 48 hilos + 8 hilos

Ejercicios prácticos

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.

100 finished in 0.000001 seconds

10000 finished in 0.000109 seconds

100000 finished in 0.000953 seconds

num\_threads = 32

100 finished in 0.002033 seconds

10000 finished in 0.021493 seconds

100000 finished in 0.156634 seconds

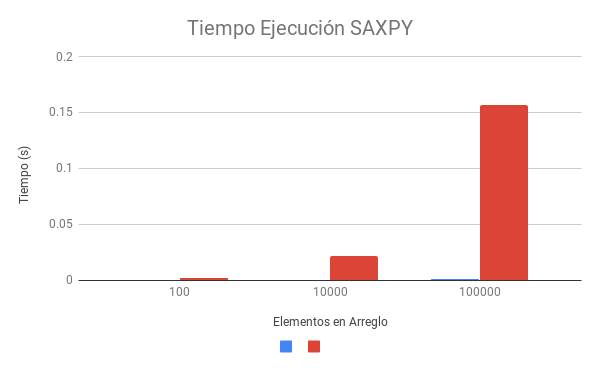


Gráfico 3. Tiempo de Ejecución con y sin OpenMP

Rojo: Con OpenMP, Azul: Sin OpenMP

1. Realice una aplicacin paralelizable que requiera gran cantidad de procesamiento. Su aplicación podrıa ser una aproximación a una integral, una serie, un conjunto de operaciones, etc.

Compilamos utilizando el comando:

Gcc ex\_par.c -lm -fopenmp

La función calcula e^x utilizando aproximaciones de

e^2 es 29 en 0.000538 segundos y 8 threads

e^10 es 101778 en 0.000552 segundos y 8 threads

Referencias bibliográficas

Chapter 10. OpenMP C/C API Multiprocessing Directives. (n.d.). Retrieved from <https://www3.risc.jku.at/education/courses/ws2003/intropar/origin-new/Pragmas/sgi_html/ch10.html>

Factorial program in C. (n.d.). Retrieved from https://www.programmingsimplified.com/c-program-find-factorial