Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Sebastián Galindo
Semestre 2, 2024

Corto # 2

Fecha de Entrega: 2 de octubre, 2023.

Descripción: En este corto explorará el algoritmo recursivo de Fibonacci. Fibonacci es una sucesión matemática importante que es ampliamente conocida. Deberá explorar la sucesión de manera recursiva con y sin el uso de OpenMP y evaluar sus resultados

Entregables: Todos los archivos de código que programe debidamente comentados e identificados. Adicionalmente un documento .pdf con los resultados y screenshots de sus pruebas, así como las respuestas a las preguntas planteadas.

Materiales: necesitará una máquina virtual con Linux.

Contenido:

En matemáticas, la sucesión de Fibonacci es la siguiente sucesión infinita de números naturales:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144…

La sucesión comienza con los números 0 y 1; a partir de estos, cada término es la suma de los dos anteriores.

Los números de Fibonacci quedan definidos por las ecuaciones

$f_0 = 0$
$f_1 = 1$
$f_n = f_{n-1} + f_{n-2}$

Con n>=2.

$f_2 = 1$
$f_3 = 2$
$f_4 = 3$
$f_5 = 5$
$f_6 = 8$
$f_7 = 13$
$f_8 = 21$

Iteración 1:

Cree un programa llamado fib.c en donde codificará el cálculo de f_n para un número n cualquiera que puede ser ingresado desde consola. Cree una función llamada “fib_recursive()” que será llamada desde su main(). Su función debe devolver el resultado de este cálculo y su implementación debe ser recursiva. Imprima el resultado en pantalla así como el tiempo de ejecución.

Universidad del Valle de Guatemala
Computación Paralela y Distribuida
Docente: Sebastián Galindo
Semestre 2, 2024

Iteración 2:

Cree una copia de su función recursiva y nómbrela “fib_recursive_omp()”.Ahora introducirá el paralelismo de tareas con OpenMP a través de la directiva “task” para paralelizar los cálculos de su algoritmo. Imprima nuevamente el resultado y el tiempo de ejecución en pantalla.

PREGUNTA: Al realizar la paralelización, ¿qué está pasando internamente y por qué el resultado no mejora?

Iteración 3:

Finalmente, para “corregir” este comportamiento un acercamiento que usualmente se utiliza es delimitar un threshold, para evitar la creación de tareas innecesarias. Copie su función anterior y nombrela “fib_recursive_omp_fix()”. En dicha función intente implementar un threshold para la creación de tareas y evalúe nuevamente el comportamiento de su algoritmo. Haga también pruebas con distintas cantidades de threads y encuentre la mejor ejecución que pueda. Imprima nuevamente el resultado y el tiempo de ejecución en pantalla.

PREGUNTA: ¿Por qué al aplicar el threshold obtenemos resultados más parecidos al de la iteración 1?

Cree una pequeña tabla en donde registre los resultados de sus pruebas en las iteraciones 2 y 3 para un n=40 y el speedup que encuentra en la iteración 3 respecto a la iteración 2.