UA

Universidad
de Alicante

ESCUELA POLITÉCNICA SUPERIOR

INTRODUCCIÓN A OpenMP

Curso académico 2024 / 2025
Semestre 2
Grado en Ingeniería en Inteligencia Artificial
Grupo 1

Profesora: Ricardo Moreno Rodríguez
Alumno: Santiago Álvarez Geanta

Universitat d'Alacant
Universidad de Alicante

Febrero 2025

ÍNDICE

Instalación y configuración

- **Instalación en Ubuntu**

Para llevar a cabo la instalación correcta del software en Ubuntu, hemos seguido un proceso estructurado que garantiza su correcto funcionamiento. A continuación, se detallan los pasos realizados:

1. **Actualización del sistema**
   - sudo apt update && sudo apt upgrade -y

Escribiremos lo siguiente en la terminal. En mi caso ya tenía actualizado apt. Al utilizar sudo estaríamos ejecutando una instrucción desde el modo administrador, por tanto, tendremos que introducir nuestra contraseña luego de ejecutar la instrucción.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ sudo apt update && sudo apt upgrade -y
[sudo] contraseña para sant_vz6:
Obj:1 http://es.archive.ubuntu.com/ubuntu noble InRelease
Obj:2 http://security.ubuntu.com/ubuntu noble-security InRelease
Des:3 http://es.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Obj:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu noble InRelease
Obj:5 http://es.archive.ubuntu.com/ubuntu noble-backports InRelease
```

2. **Instalar GCC con soporte para OpenMP**
   - sudo apt install gcc g++ -y

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ sudo apt install gcc g++ -y
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
gcc ya está en su versión más reciente (4:13.2.0-7ubuntu1).
g++ ya está en su versión más reciente (4:13.2.0-7ubuntu1).
```

3. **Verificación de la instalación:Extracción y configuración:**
   - gcc --version

Una vez realizada la instalación comprobaremos que se haya instalado todo correctamente utilizando la instrucción **--version**.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ gcc --version
gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ g++ --version
g++ (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Podemos ver que la instalación está correcta y disponemos de los compiladores gcc y g++. A continuación probaremos compilar un archivo de prueba que simplemente consiste en un Hello World utilizando el paralelismo.

prueba.cc > main()

```
#include <stdio.h>
#include <omp.h>
int main() {
    #pragma omp parallel
    {
    printf("Hola desde el hilo %d\n", omp_get_thread_num());
    }
return 0;
}
```

Para compilar usaremos la siguiente sintaxis. Es importante recalcar que debemos compilar utilizando **-fopenmp**. Esto se debe a que en una compilación normal, openMP no está inicializado por defecto. Por tanto, debemos ser nosotros los que inicialicen openMP manualmente en cada compilación.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ g++ -fopenmp prueba.cc -o prueba
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ ./prueba
Hola desde el hilo 11
Hola desde el hilo 6
Hola desde el hilo 13
Hola desde el hilo 4
Hola desde el hilo 14
Hola desde el hilo 5
Hola desde el hilo 12
Hola desde el hilo 2
Hola desde el hilo 3
Hola desde el hilo 9
Hola desde el hilo 15
Hola desde el hilo 1
Hola desde el hilo 8
Hola desde el hilo 7
Hola desde el hilo 10
Hola desde el hilo 0
```

- **Errores y dificultades que nos pueden suceder en la instalación**

  Algunas consideraciones durante la instalación es el uso de **sudo**, si bien es necesario ejecutar las instrucciones como administradores deberemos tener en cuenta que tendremos que utilizar nuestra contraseña. También deberemos tener espacio disponible en memoria para poder instalar **gcc** y **g++**. Otro detalle importante es que si estamos trabajando con archivos de extensión **.C** deberemos utilizar **gcc**, mientras que para archivos **.CC** utilizaremos **g++**. Esto es bastante importante porque en la práctica se nos indica que debemos compilar con gcc, pero en realidad debemos usar g++.

Compilar y ejecutar:

```
gcc -fopenmp test_openmp.c -o test_openmp && ./test_openmp
```

Ejercicios

Ejercicio 1

En este ejercicio implementaremos un programa en C ++ que imprime un mensaje desde múltiples hilos. Tendremos que importar la biblioteca de OpenMP <omp.h> para poder usar las funciones para la programación paralela en nuestro código. También utilizaremos **#pragma omp parallel** para indicar que el bloque de código que sigue debe ejecutarse en paralelo utilizando múltiples hilos. Utilizaremos las funciones **omp_get_thread_num()**, que nos devuelve el número del hilo actual; y **omp_get_num_threads(),** que devuelve el número total de hilos utilizados en la ejecución paralela.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ ./E
Hola desde el hilo 11 de 16
Hola desde el hilo 12 de 16
Hola desde el hilo 4 de 16
Hola desde el hilo 2 de 16
Hola desde el hilo 13 de 16
Hola desde el hilo 3 de 16
Hola desde el hilo 15 de 16
Hola desde el hilo 10 de 16
Hola desde el hilo 7 de 16
Hola desde el hilo 14 de 16
Hola desde el hilo 6 de 16
Hola desde el hilo 1 de 16
Hola desde el hilo 5 de 16
Hola desde el hilo 9 de 16
Hola desde el hilo 8 de 16
Hola desde el hilo 0 de 16
```

El código para el ejercicio 1 es bastante simple, como ya mencionamos, tan solo necesitaremos utilizar **#pragma omp parallel** para ejecutar la instrucción printf desde mútiples hilos.

```c
#include <stdio.h>
#include <omp.h>
int main() {
    #pragma omp parallel
    {
    printf("Hola desde el hilo %d de %d\n", omp_get_thread_num(), omp_get_num_threads());
    }
return 0;
}
```

Ejercicio 2

En este ejercicio, implementaremos un programa en C++ que calcula la suma de los elementos de un arreglo de manera paralela utilizando OpenMP. Para ello, usaremos **#pragma omp parallel for reduction(+:variable_de_suma)**, que nos permitirá distribuir la carga de trabajo entre múltiples hilos y asegurar que la variable suma acumule los resultados correctamente sin condiciones de carrera. La paralelización nos ayudará a reducir el tiempo de ejecución al distribuir la tarea de suma en varios hilos, lo que es especialmente útil cuando trabajamos con grandes volúmenes de datos.

```
sant_vz6@santvz6-IdeaPad-Gaming-3-15ACH6:~/Escritorio/UA-2/2 CUATRI/CAR/P4$ ./E2
Suma total: 499500.000000
```

Para desarrollar el código primero definiremos el tamaño de la matriz usando #define N 1000. Luego declararemos las variables necesarias:

- suma: variable para almacenar la suma total.
- A[N]: el array donde almacenaremos los valores.

Inicializamos la matriz con valores de 0 a 999, multiplicando cada índice por 1. Paralelizamos la suma usando **#pragma omp parallel for reduction(+:suma)** para así:

- Dividir el bucle entre múltiples hilos para que cada uno sume una parte del arreglo.
- Utilizar la cláusula reduction(+:suma), que evita condiciones de carrera al hacer que cada hilo mantenga su propia suma parcial antes de combinar los resultados.

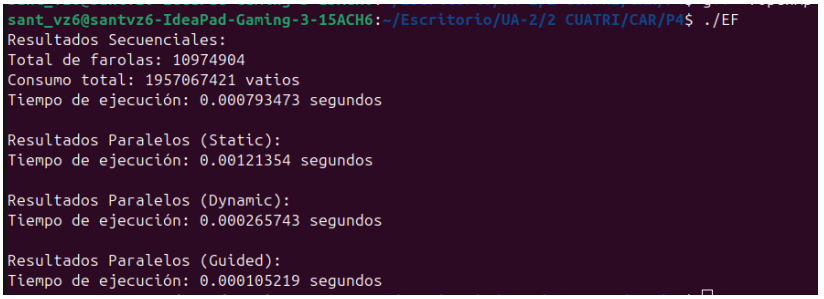
Finalmente imprimimos el resultado final con printf para comprobar la eficiencia del cálculo al aprovechar la computación en paralelo.

```c
#include <stdio.h>
#include <omp.h>

#define N 1000

int main() {
    int i;
    double suma = 0.0;
    double A[N];


    for (i = 0; i < N; i++)
        A[i] = i * 1.0;


    #pragma omp parallel for reduction(+:suma)
    for (i = 0; i < N; i++) {
        suma += A[i];
    }

    printf("Suma total: %f\n", suma);

    return 0;
}
```

Ejercicio Final

Este ejercicio busca simular el consumo energético del alumbrado público en Alicante y comparar el rendimiento entre una implementación secuencial y una versión paralelizada usando OpenMP. Se modelará una cuadrícula de **200x200** celdas, cada una representando una zona de la ciudad. Cada celda contendrá un número aleatorio de farolas (entre **50 y 500**). Además, cada farola podrá tener tres tipos de consumo: **bajo consumo:** 70 - 100 vatios, **consumo medio:** 150 - 200 vatios o **alto consumo:** 250 - 300 vatios. También se calculará el consumo total de todas las farolas en dos versiones: **secuencial** (sin paralelismo) o **paralela** (con OpenMP). Finalmente se compararán los tiempos de ejecución.

Resultados obtenidos:



Resultado Secuencial

En la versión secuencial, el tiempo de ejecución es bastante bajo, lo que puede indicar que la cantidad de operaciones realizadas no es tan grande como para que la ejecución en un solo hilo sea significativamente costosa.

Resultado Paralelismo

Static: Esta estrategia divide el trabajo en bloques de tamaño fijo y los asigna a los hilos de manera equitativa. Curiosamente, aquí el tiempo de ejecución es mayor que en la versión secuencial, lo que sugiere una posible sobrecarga en la creación y sincronización de los hilos.

Dynamic: En esta estrategia, los bloques de trabajo se asignan dinámicamente conforme los hilos van terminando sus tareas. La ejecución es mucho más rápida que en la versión estática y la secuencial, lo que sugiere que la carga de trabajo no está perfectamente equilibrada y se benefició de una distribución más flexible.

Guided: En este caso, los bloques de trabajo comienzan siendo grandes y van reduciendo su tamaño a medida que se asignan. Es **la versión más rápida de todas**, lo que indica que **la reducción progresiva del tamaño** de las tareas ayudó a minimizar el tiempo de espera entre hilos y mejoró el aprovechamiento del procesador.

En conclusión podemos decir que la paralelización con OpenMP **puede acelerar el proceso**, pero no siempre garantiza la mejora, por ejemplo, si la carga de trabajo es pequeña o no está bien distribuida. Por un lado, **"Guided" fue la opción más eficiente**, lo que sugiere que la carga de trabajo no es uniforme y que la división adaptativa del trabajo fue clave para la optimización. Por otro lado, la estrategia "Static" **no funcionó bien**, posiblemente por lo que ya hemos mencionado, **la desigualdad de la carga**.