Universitat d'Alacant
Universidad de Alicante

PRÁCTICA SEMANA 4: INTRODUCCIÓN A OPENMP Y PREPARACIÓN DEL ENTORNO

NOTA: importante: Adaptación de la Guía y Resolución de Problemas

Esta guía proporciona un enfoque **general** para la instalación y configuración de OpenMP en entornos Windows y Ubuntu. Sin embargo, cada estudiante debe **adaptarla a su caso particular**, ya que pueden encontrarse con situaciones específicas que no estén detalladas aquí.

Para ello, pueden **utilizar inteligencia artificial como asistente** para la configuración personalizada de sus PC o los equipos del aula.

Se valorará **especialmente la capacidad de resolución de problemas**:

- **Identificación de errores durante la instalación o ejecución.**
- **Investigación de soluciones adecuadas.**
- **Documentación clara de cada problema y su solución en el informe.**

Es por ello que cada estudiante debe incluir en su informe **todos los problemas encontrados y cómo los ha solucionado**, ya que esto **será un criterio clave en la evaluación.**

Objetivos de la práctica

1. **Preparar el entorno de desarrollo** en Windows o Ubuntu para programar con OpenMP.
2. **Documentar detalladamente** el proceso de instalación y configuración.
3. **Comprender los fundamentos de OpenMP**, su sintaxis y ejecución en programas simples.
4. **Desarrollar y analizar programas paralelos** con OpenMP.
5. **Aplicar OpenMP a un problema real**: estimación del consumo energético del alumbrado público en Alicante.

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Universitat d'Alacant
Universidad de Alicante

Parte 1: Instalación y Configuración del Entorno (70% de la nota)

Cada estudiante debe elaborar un **informe detallado** con:

1. **Especificaciones del sistema** (SO, versión, procesador, memoria RAM).
2. **Pasos detallados de instalación** de OpenMP con comandos y capturas de pantalla.
3. **Configuración del compilador** para habilitar OpenMP.
4. **Pruebas de verificación** de la instalación con código de prueba.
5. **Posibles errores y soluciones** documentadas.

Los problemas encontrados y cómo se han resuelto deben estar bien explicados, ya que esto **formará parte fundamental de la evaluación.**

Guía general de instalación (cada estudiante debe adaptarla según su caso)

Instalación en Ubuntu

1. **Actualizar el sistema**

```
sudo apt update && sudo apt upgrade -y
```

2. **Instalar GCC con soporte para OpenMP**

```
sudo apt install gcc g++ -y
```

3. **Verificar la instalación**

```
gcc --version
```

4. **Ejecutar un programa de prueba**

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

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

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Compilar y ejecutar:

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

INSTALACIÓN EN WINDOWS

OPCIÓN 1: MINGW CON GCC

1. Descargar e instalar **MinGW-W64** desde MinGW-W64.
2. Agregar MinGW al **PATH del sistema**.
3. Verificar la instalación con:

```
gcc --version
```

4. **Compilar y ejecutar** el mismo código de prueba que en Ubuntu.

OPCIÓN 2: VISUAL STUDIO

1. Descargar e instalar **Visual Studio Community**.
2. Seleccionar el paquete de **Desarrollo en C++ con CMake**.
3. Habilitar OpenMP en la configuración del compilador.

Universitat d'Alacant
Universidad de Alicante

PARTE 2: PROGRAMACIÓN CON OPENMP (30% DE LA NOTA)

Los estudiantes deben presentar un **informe completo** con:

1. **Código fuente** correctamente comentado.
2. **Diagramas de flujo** explicando el funcionamiento de cada programa en el que queda claras las decisiones del flujo en base a qué criterio se toman (por ejemplo, aquí utilizo Schedule(static) porque las tareas en ésta parte del programa son de un tamaño predecible y los hilos que genera también…).
3. **Análisis de rendimiento** comparando ejecución secuencial y paralela.

EJERCICIO 1: "HOLA MUNDO PARALELO"

Implementar un programa en C que imprima un mensaje desde múltiples hilos.

```
#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: PARALELIZACIÓN DE UN BUCLE SIMPLE

Implementar un programa que sume los elementos de un arreglo en paralelo.

```
#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;
}
```

A continuación deberás resolver el ejercicio final propuesto

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

Universitat d'Alacant
Universidad de Alicante

EJERCICIO FINAL: ESTIMACIÓN DEL CONSUMO ENERGÉTICO DEL ALUMBRADO PÚBLICO EN ALICANTE

OBJETIVO

El objetivo de este ejercicio es utilizar OpenMP para **paralelizar la simulación del consumo energético** del alumbrado público en la ciudad de Alicante. Se simulará el trabajo de **20 trabajadores** que registran el número de farolas en distintas zonas y clasifican su tipo de consumo energético.

El objetivo final es:

- **Comparar el rendimiento** de una versión secuencial y una versión paralela del código.
- **Analizar cómo el número de hilos influye en el tiempo de ejecución**.
- **Evitar condiciones de carrera** mediante el uso de reducción (`reduction(+:variable)`) y estrategias de sincronización.

TAREAS:

- Modelar una cuadrícula de 200x200 celdas con farolas de diferentes consumos.
- Implementar una versión secuencial y otra paralela con OpenMP.
- Comparar el tiempo de ejecución y optimizar el código.

El informe debe incluir los códigos fuente, diagrama de flujo detallado y una comparación entre la versión secuencial y la paralela.

DESCRIPCIÓN DEL PROBLEMA

- Se modelará una cuadrícula de **200x200 celdas**, donde cada celda representa una zona de la ciudad.
- Cada celda contiene un número aleatorio de farolas entre **50 y 500**.
- Cada farola tiene un consumo energético diferente según su tipo:
  - **Bajo consumo**: entre **70 y 100 vatios**.
  - **Consumo medio**: entre **150 y 200 vatios**.
  - **Alto consumo**: entre **250 y 300 vatios**.
- Se calculará el **consumo total** de todas las farolas en la ciudad en **dos versiones**:
  1. **Versión secuencial** (sin paralelismo).
  2. **Versión paralela** (con OpenMP).

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

IMPLEMENTACIÓN EN C++ CON OPENMP

PASO 1: INICIALIZACIÓN DEL MAPA

Cada celda de la cuadrícula se inicializa con un número aleatorio de farolas.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <omp.h>

using namespace std;

const int MAP_SIZE = 200;
const int MIN_FAROLAS = 50, MAX_FAROLAS = 500;
const int BAJO_MIN = 70, BAJO_MAX = 100;
const int MEDIO_MIN = 150, MEDIO_MAX = 200;
const int ALTO_MIN = 250, ALTO_MAX = 300;

struct Celda {
    int num_farolas;
    int consumo_total;
};

void inicializarMapa(vector<vector<Celda>> &mapa) {
    srand(time(0));
    for (int i = 0; i < MAP_SIZE; i++) {
        for (int j = 0; j < MAP_SIZE; j++) {
            mapa[i][j].num_farolas = rand() % (MAX_FAROLAS - MIN_FAROLAS + 1) + MIN_FAROLAS;
            for (int k = 0; k < mapa[i][j].num_farolas; k++) {
                int tipo = rand() % 3;
                if (tipo == 0) mapa[i][j].consumo_total += rand() % (BAJO_MAX - BAJO_MIN + 1) + BAJO_MIN;
                else if (tipo == 1) mapa[i][j].consumo_total += rand() % (MEDIO_MAX - MEDIO_MIN + 1) + ME
DIO_MIN;
                else mapa[i][j].consumo_total += rand() % (ALTO_MAX - ALTO_MIN + 1) + ALTO_MIN;
            }
        }
    }
}
```

PASO 2: CÁLCULO SECUENCIAL

```
void calcularConsumoSecuencial(const vector<vector<Celda>> &mapa, long long &total_farolas,
long long &consumo_total) {
    total_farolas = 0;
    consumo_total = 0;
    for (int i = 0; i < MAP_SIZE; i++) {
        for (int j = 0; j < MAP_SIZE; j++) {
            total_farolas += mapa[i][j].num_farolas;
            consumo_total += mapa[i][j].consumo_total;
        }
    }
}
```

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es

PASO 3: CÁLCULO PARALELO CON OPENMP

```
void calcularConsumoParalelo(const vector<vector<Celda>> &mapa, long long &total_farolas,
long long &consumo_total) {
    total_farolas = 0;
    consumo_total = 0;
    #pragma omp parallel for reduction(+:total_farolas, consumo_total) schedule(dynamic)
    for (int i = 0; i < MAP_SIZE; i++) {
        for (int j = 0; j < MAP_SIZE; j++) {
            total_farolas += mapa[i][j].num_farolas;
            consumo_total += mapa[i][j].consumo_total;
        }
    }
}
```

PASO 4: ANÁLISIS DEL RENDIMIENTO

- Comparar los tiempos de ejecución **con diferentes números de hilos** (`OMP_NUM_THREADS`).
- Evaluar el impacto del **tipo de schedule (static, dynamic, guided)**.

Universitat d'Alacant
Universidad de Alicante

RÚBRICA DE EVALUACIÓN

PARTE 1: INSTALACIÓN Y CONFIGURACIÓN (70%)

Criterio	Ponderación	Descripción
Instalación correcta	20%	OpenMP configurado y operativo.
Documentación detallada	30%	Capturas de pantalla y pasos claros.
Pruebas de verificación	20%	Código de prueba funcional.
Resolución de problemas	30%	Se valora la identificación de errores y su solución.

PARTE 2: PROGRAMACIÓN CON OPENMP (30%)

Criterio	Ponderación	Descripción
Implementación correcta	15%	Código funcional y bien estructurado.
Diagramas de flujo	10%	Explicación clara del código.
Análisis de rendimiento	5%	Comparación secuencial vs paralelo.

ANEXO: JUSTIFICACIÓN ACADÉMICA

Esta práctica está alineada con las competencias generales, transversales y específicas de la asignatura **Computación de Alto Rendimiento**. Se vincula directamente con los resultados de aprendizaje establecidos en el programa del curso, como la **implementación de aplicaciones paralelas y la selección de arquitecturas adecuadas para la resolución de problemas complejos.**

Asignatura: computación de alto rendimiento
Proferor: Ricardo Moreno
Email: ricardo.moreno@ua.es