Práctica Semana 7: Análisis de Rendimiento en OpenMP

PARTE 1: Introducción al Análisis de Rendimiento en OpenMP

1. Qué vas a medir y calcular en esta práctica

Durante la práctica, vas a medir y calcular los siguientes valores:

- **Tiempo de ejecución** (con distintos números de hilos).
- **Speed-up (S)**.
- **Eficiencia (E)**.
- **Ley de Amdahl** (en un ejercicio específico).

2. ¿Qué significa cada concepto?

Tiempo de Ejecución (T_N)

- Tiempo que tarda el programa en ejecutarse usando **N hilos**.
- Se mide en **segundos** utilizando `omp_get_wtime()`.

Speed-up (S)

- Cuánto más rápido es el programa **paralelo** respecto a la versión **secuencial (1 hilo)**.
- Fórmula:

$$S = \frac{T_1}{T_N}$$

- Ejemplo:
  - Si T_1 = 100 seg y T_4 = 30 seg:

$$S = \frac{100}{30} = 3.33$$

Eficiencia (E)

- Qué tan bien se utilizan los hilos disponibles.
- Fórmula:

$$E = \frac{S}{N}$$

- Ejemplo:
  - Si S = 3.33
  - con N = 4 hilos:

$$E = \frac{3.33}{4} = 0.83$$

E=83.25%

Ley de Amdahl (solo en el ejercicio 4)

- Límite teórico del Speed-up según la parte paralelizable del código.
- Fórmula:

$$S = \frac{1}{(1-p) + \frac{p}{N}}$$

- Ejemplo:
  - Si p = 0.85
  - N = 8 hilos: S

$$S = \frac{1}{0.15 + \frac{0.85}{8}} = 4.71$$

3. ¿Qué tienes que hacer en cada ejercicio?

Ejercicio	Qué mides	Qué calculas	Qué concluyes
1	Tiempos con distintos hilos	Speed-up y Eficiencia	Si el programa escala bien al aumentar hilos
2	Tiempos con distintas sincronizaciones	Comparas los tiempos y Speed-up	Qué técnica de sincronización es mejor
3	Tiempos con distintos `schedule`	Comparas los tiempos	Qué `schedule` funciona mejor según el balanceo de carga
4	Speed-up real vs Speed-up teórico (Amdahl)	Ley de Amdahl	Si el límite teórico se cumple o no
5 (opcional)	Tiempos con carga proporcional	Escalabilidad débil	Si el tiempo se mantiene constante

4. Cómo rellenar tus tablas (ejemplo práctico)

Nº de Hilos (N)	Tiempo T_N (s)	Speed-up (S)	Eficiencia (E)
1	100	1.00	1.00 (100%)
2	55	1.82	0.91 (91%)
4	30	3.33	0.83 (83%)
8	18	5.56	0.69 (69%)

5. Recomendaciones

- Haz **3 mediciones** por cada prueba y calcula la **media**.
- Utiliza las funciones `omp_get_wtime()` **antes y después** del bloque que quieres medir.
- Anota los **valores reales** y no inventes datos, el análisis es lo que importa.

Universitat d'Alacant
Universidad de Alicante

PARTE 2: Ejercicios Prácticos

Objetivo General

- Medir el rendimiento de programas paralelos en OpenMP.
- Analizar el impacto del número de hilos, técnicas de sincronización y balanceo de carga sobre el rendimiento.
- Interpretar Speed-up, Eficiencia y aplicar la Ley de Amdahl a partir de medidas reales.

Indicaciones Generales

- Todos los ejercicios deben realizarse y los resultados obtenidos deben incluirse en un **único informe final**, organizado por ejercicios.
- El informe recogerá las **mediciones, cálculos y conclusiones** de cada ejercicio de la práctica.

Ejercicio 1: Medición de Speed-up y Eficiencia (Reducción)

Código base:

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

int main() {
    long long N = 10000000000; // 10.000 millones
    double suma = 0.0;
    double start, end;

    start = omp_get_wtime();

    #pragma omp parallel for reduction(+:suma)
    for (long long i = 1; i <= N; i++) {
        suma += (i * 1.5) / (i + 1.0);
    }

    end = omp_get_wtime();

    printf("Resultado final: %.2f\n", suma);
    printf("Tiempo: %f segundos\n", end - start);
    return 0;
}
```

Tareas:

1. Ejecuta con 1, 2, 4, 8 y 16 hilos.
2. Registra los tiempos de ejecución.
3. Calcula Speed-up (S) y Eficiencia (E).
4. Representa los resultados en una tabla.
5. Conclusiones sobre el rendimiento y eficiencia al aumentar los hilos.

Ejercicio 2: Comparación de Sincronización (Critical vs Atomic vs Reduction)

Descripción:

Compara el impacto en el rendimiento de tres técnicas de sincronización para la misma operación de suma.

Código base Critical (ejemplo):

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

int main() {
    long long N = 1000000000; // 1.000 millones
    double suma = 0.0;
    double start, end;

    start = omp_get_wtime();

    #pragma omp parallel for
    for (long long i = 1; i <= N; i++) {
        double valor = (i * 1.5) / (i + 1.0);
        #pragma omp critical
        {
            suma += valor;
        }
    }

    end = omp_get_wtime();

    printf("Resultado final: %.2f\n", suma);
    printf("Tiempo: %f segundos\n", end - start);
    return 0;
}
```

Tareas:

1. Implementa la versión con `critical`, `atomic` y `reduction`.
2. Ejecuta con 1, 2, 4, 8 hilos.
3. Mide y compara tiempos.
4. Reflexiona sobre la eficiencia de cada técnica de sincronización.

Universitat d'Alacant
Universidad de Alicante

Ejercicio 3: Balanceo de Carga y Schedule

Descripción:

Analiza el efecto de diferentes políticas de `schedule` sobre el rendimiento en cargas desiguales.

Código base:

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

int main() {
    double resultado_final = 0.0;
    double start, end;

    start = omp_get_wtime();

    #pragma omp parallel for reduction(+:resultado_final) schedule(dynamic, 1)
    for (int i = 0; i < 16; i++) {
        double local_result = 0.0;
        long long carga = 100000000 * (i % 4 + 1); // Cargas desiguales
        for (long long j = 0; j < carga; j++) {
            local_result += (j * 0.5) / (j + 1.0); // Cálculo sencillo que evita optimización excesiva
        }
        resultado_final += local_result;
    }

    end = omp_get_wtime();

    printf("Resultado final: %.2f\n", resultado_final);
    printf("Tiempo: %f segundos\n", end - start);

    return 0;
}
```

Tareas:

1. Prueba `schedule(static)`, `schedule(dynamic, 1)` y `schedule(guided)`.
2. Mide los tiempos de ejecución con 4 y 8 hilos.
3. Analiza el balanceo de carga observando los tiempos.
4. Conclusión sobre el `schedule` óptimo en este caso.

Detalles importantes del código:

- El `schedule(dynamic, 1)` se puede cambiar en las pruebas a `static` o `guided` para comparar el rendimiento.
- La carga de trabajo varía según `i`, simulando un desequilibrio en la carga entre iteraciones.
- Si la operación en el bucle interior no tiene suficiente complejidad para evitar que el compilador la elimine por ser trivial añádele carga de trabajo

Ejercicio 4: Análisis de Rendimiento y Ley de Amdahl

Tareas:

1. Supón que el 85% del código es paralelizable (`p = 0.85`).
2. Calcula el Speed-up máximo teórico (hilos infinitos):0

$$S_{max} = 1/(1-p)$$

3. Calcula el Speed-up teórico para 4 y 8 hilos:

$$S = 1/((1-p) + p/N)$$

4. Compara con los Speed-up reales del Ejercicio 1.
5. Reflexiona sobre la relación entre teoría y práctica.

Ejercicio 5 (Opcional): Escalabilidad Débil

Descripción:

Analiza si el sistema mantiene tiempos constantes al aumentar hilos y carga proporcionalmente.

Tareas:

1. Procesa 10.000 millones de elementos por hilo (1 hilo → 10.000 millones, 2 hilos → 20.000 millones, etc.).
2. Mide el tiempo para 1, 2, 4 y 8 hilos.
3. Analiza si el tiempo permanece constante.
4. Reflexiona sobre la escalabilidad débil del sistema.

Entregables

- Un único informe que incluya:
  - Resultados obtenidos en cada ejercicio.
  - Tiempos de ejecución, cálculos de Speed-up y Eficiencia.
  - Gráficos de Speed-up y Eficiencia según el número de hilos.
  - Conclusiones y reflexiones finales.

Universitat d'Alacant
Universidad de Alicante

Anexo: Rúbrica de Evaluación y Justificación Académica

Rúbrica de Evaluación

Criterio	Puntos
Realización de todos los ejercicios	3
Calidad en la recogida de datos y tablas	2
Cálculo correcto de Speed-up y Eficiencia	2
Interpretación adecuada de resultados y conclusiones	2
Presentación clara del informe	1
Total	**10**

Justificación Académica

Esta práctica permite al alumnado:

- Comprender los conceptos de Speed-up, Eficiencia y Escalabilidad.
- Aplicar técnicas de optimización y análisis de rendimiento en entornos de programación paralela con OpenMP.
- Evaluar el impacto de la paralelización y la sincronización en el rendimiento de aplicaciones.
- Fomentar el análisis crítico mediante la comparación de resultados teóricos y experimentales.