**PROYECTO DE PARALELIZACIÓN CON OPEN-MP:   
ECUACIÓN DE POISSON EN 2D**

Asignatura: Sistemas Distribuidos  
Docente: Carlos Andrés Gómez Vasco  
Estudiantes: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Objetivo

Aplicar diferentes directivas de OpenMP para paralelizar un código secuencial que resuelve la ecuación de Poisson 2D mediante diferencias finitas, y analizar el impacto de cada estrategia en el rendimiento y estructura del programa.

# Preparación

1. Compila y ejecuta el código base secuencial.  
2. Registra los resultados a continuación:

Tiempo de ejecución: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ segundos  
Número de iteraciones: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Tolerancia alcanzada: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Número de threads (en este punto): 1

# Actividad 1: #pragma omp parallel for

Paralelizar el bucle principal en solve\_poisson.  
  
Tareas:  
- Aplica #pragma omp parallel for sobre el bucle de i.  
- Usa reduction(max:delta) para evitar condiciones de carrera.  
  
Resultados:  
Número de threads usados: \_\_\_\_\_\_\_\_\_\_\_\_  
Tiempo de ejecución: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
¿Hubo diferencia en el número de iteraciones? \_\_\_\_  
  
- ¿Por qué es necesaria la cláusula reduction?  
- ¿Hubo mejora de rendimiento?

# Actividad 2: collapse(2)

Acelerar anidamiento de bucles.  
  
Tareas:  
- Sustituye el bucle paralelo anterior por:  
#pragma omp parallel for collapse(2) reduction(max:delta)  
  
Resultados:  
Tiempo de ejecución: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
¿Mejoró respecto a la versión anterior? Sí / No  
  
Reflexión:  
- ¿Qué hace collapse(2)?  
- ¿Es siempre recomendable su uso?

# Actividad 3: sections

Ejecutar funciones independientes en paralelo.  
  
Tareas:  
- Paraleliza initialize\_grid y poisson\_source con sections.  
  
Resultados:  
Tiempo de ejecución hasta terminar ambas funciones: \_\_\_\_\_\_\_\_\_\_\_\_  
¿Tuviste que reordenar las funciones? ¿Por qué?  
  
Reflexión:  
- ¿Vale la pena paralelizar funciones tan rápidas?

# Actividad 4: parallel y for separados

Controlar la región paralela.  
  
Tareas:  
- Usa:  
#pragma omp parallel  
{  
 #pragma omp for reduction(max:delta)  
}  
- Añade variantes con schedule(static) y schedule(dynamic).  
  
Resultados de tiempo:  
Estrategia:

* schedule(static).
* schedule(dynamic)

# Actividad 5: nowait, barrier, single, critical

Control de sincronización.  
  
Tareas:  
- Usa nowait en bucles independientes.  
- Imprime inicio de cada iteración con #pragma omp single.  
- Agrega sección artificial crítica con #pragma omp critical.  
  
Resultados:  
Tiempo con nowait: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
¿Hubo cambio con respecto al uso por defecto?  
¿Dónde colocaste la directiva single y por qué?  
¿critical ralentizó la ejecución? ¿En qué contexto sería útil?

# Actividad Final: Comparación de Estrategias

Completa la siguiente tabla:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión | Directiva usada | Tiempo (s) | Iteraciones | Observaciones |
| Secuencial |  |  |  |  |
| Paralelo básico |  |  |  |  |
| Colapsado de bucles |  |  |  |  |
| Inicialización y fuente en paralelo |  |  |  |  |
| Control explícito + schedule |  |  |  |  |
| Control de sincronización |  |  |  |  |

# Conclusiones

1. ¿Cuál fue la versión más rápida?  
Respuesta: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
2. ¿Qué directiva es más fácil de usar?  
Respuesta: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
3. ¿Hubo directivas que empeoraron el rendimiento? ¿Por qué?  
Respuesta: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Actividad 6: Contar el número de iteraciones con acceso compartido

En esta actividad, vas a modificar la función solve\_poisson para contar el número total de iteraciones del bucle externo (las actualizaciones completas de la grilla) utilizando una variable global o compartida.

Instrucciones:

* Declara una variable int iterations = 0; como global o como variable compartida si usas una región paralela.
* Aumenta su valor cada vez que finaliza una iteración del bucle while (delta > TOL).
* Utiliza una de las siguientes estrategias para evitar condiciones de carrera:
* Una sección crítica con #pragma omp critical
* Una operación atómica con #pragma omp atomic

# Conclusiones

* ¿Qué diferencias observas en el rendimiento entre usar critical y atomic?
* ¿En qué casos sería preferible una sobre la otra?

Actividad 7: Exploración con tareas (task)

Hasta ahora se ha usado paralelización por bucles. En esta actividad te proponemos experimentar con la directiva task de OpenMP para dividir el dominio en bloques y asignar tareas a distintos hilos.

Instrucciones:

* Dentro de solve\_poisson, divide la grilla en bloques (por ejemplo, en cuadrantes o bloques de tamaño fijo).
* Dentro del bucle while, crea una región paralela y dentro de ella una sección single con múltiples task para cada bloque.
* Cada tarea deberá actualizar su bloque local de la grilla.
* Asegúrate de sincronizar con #pragma omp taskwait.

Ejemplo simplificado:

#pragma omp parallel

{

#pragma omp single

{

for (int bi = 1; bi < M; bi += block\_size) {

for (int bj = 1; bj < N; bj += block\_size) {

#pragma omp task

{

for (int i = bi; i < std::min(bi + block\_size, M); ++i) {

for (int j = bj; j < std::min(bj + block\_size, N); ++j) {

// actualización de T[i][j]

}

}

}

}

}

#pragma omp taskwait

}

}

**Conclusión:**

¿Cómo se compara el rendimiento usando task con respecto a parallel for?  
¿Qué ventajas y desventajas tiene este enfoque?

**COMO AJUSTAR Y ENTREGAR EL PROYECTO.**

Este es un ejemplo de la estructura final para entregar.

Taller\_OpenMP\_Poisson/

├── Makefile

├── README.md

├── data/

│ └── solucion\_serial.dat #Archivos de salida … todos los de salida

├── src/

│ ├── poisson\_serial.cpp # Código base sin paralelismo

│ ├── poisson\_parallel\_for.cpp # Con parallel for

│ ├── poisson\_critical.cpp # Con sección crítica

│ ├── poisson\_atomic.cpp # Con variable atómica

│ ├── poisson\_task.cpp # Con tareas

│ └── utils.h # Encabezado común (opcional)

├── actividades/

│ └── Informe de Resultados OpenMP\_Ecuacion\_Poisson.docx

├── imag/

│ └── Todas las imágenes conseguidas.

└── bin/

├── poisson\_serial

├── poisson\_parallel\_for

├── poisson\_critical

├── poisson\_atomic

└── poisson\_task