Práctica 1: Estudio y propuesta de la paralelización de una aplicación de simulación de fluidos

Jaime Hernández - Dario Simón

Curso 2024/25

Índice

1. Objetivos **3**

2. Presentación del problema propuesto **4**
2.1. Descripción del problema 4
2.2. Justificación de la elección 4

3. Estudio pormenorizado de la aplicación propuesta **6**
3.1. Estructura del código 6
3.2. Grafo de Dependencia entre Tareas 7
3.3. Análisis de variables 8

4. Estudio de variación de la carga computacional **10**

5. Análisis de rendimiento **11**
5.1. Impacto de los parámetros de compilación 11
5.2. Análisis de variaciones en el speed-up 11

6. Justificación como candidato para paralelización **13**

7. Conclusiones **14**

8. Ejercicios Propuestos **15**

9. Posibles Preguntas de Examen **17**
9.1. Preguntas de Opción Múltiple 17
9.2. Preguntas de Desarrollo 18
9.3. Problema de Cálculo 19

10. Bibliografía **20**

1. Objetivos

Los objetivos principales de esta práctica son:

- Identificar y analizar una aplicación secuencial con potencial para ser paralelizada eficientemente.
- Utilizar métricas cuantitativas para justificar la idoneidad de la aplicación seleccionada para la paralelización.
- Evaluar y proponer arquitecturas paralelas óptimas para la ejecución de la aplicación paralelizada.
- Investigar y cuantificar el impacto de diversos parámetros de compilación en el rendimiento de la aplicación.
- Aplicar métodos teóricos y prácticos para estimar las ganancias máximas y la eficiencia potencial del proceso de paralelización.

Estos objetivos nos permitirán desarrollar una comprensión profunda de los desafíos y oportunidades en la paralelización de aplicaciones computacionalmente intensivas.

2. Presentación del problema propuesto

2.1. Descripción del problema

La aplicación seleccionada para este estudio es una simulación de mecánica de fluidos en un tubo. Este problema implica el cálculo de las interacciones entre un gran número de partículas en un espacio tridimensional, modelando el comportamiento de un fluido bajo diversas condiciones físicas.

El sistema simula N partículas dentro de un tubo cilíndrico, donde cada partícula está sujeta a:

- Fuerzas de interacción con otras partículas
- Gravedad
- Viscosidad del fluido
- Colisiones con las paredes del tubo

La ecuación fundamental que gobierna el movimiento de cada partícula es:

$$m\frac{d^2\vec{r}}{dt^2} = \vec{F}_{\text{interacción}} + \vec{F}_{\text{gravedad}} + \vec{F}_{\text{viscosidad}} + \vec{F}_{\text{colisión}}$$

donde m es la masa de la partícula, $\vec{r}$ es su posición, y $\vec{F}$ representa las diferentes fuerzas actuando sobre la partícula.

2.2. Justificación de la elección

La simulación de fluidos es un problema computacionalmente intensivo con aplicaciones en numerosos campos:

- Aerodinámica: Diseño de vehículos y estructuras.
- Oceanografía: Modelado de corrientes marinas.
- Ingeniería química: Diseño de reactores y procesos de mezcla.
- Meteorología: Predicción del clima.
- Animación por computadora: Efectos visuales realistas de fluidos.

La naturaleza de los cálculos, que involucran interacciones entre múltiples partículas, presenta oportunidades significativas para la paralelización:

- Los cálculos para cada partícula son en gran medida independientes.
- El problema escala bien con el aumento del número de partículas.
- Existe un balance entre el paralelismo de datos (cálculos por partícula) y el paralelismo de tareas (diferentes etapas de la simulación).

3. Estudio pormenorizado de la aplicación propuesta

3.1. Estructura del código

El código está organizado en tres componentes principales:

- `FluidSystem`: Maneja la lógica de la simulación de partículas.

```
class FluidSystem {
        private:
        std::vector<Particle> particles;
        // ... otros miembros privados ...
        public:
        FluidSystem(int numParticles, ...);
        void update(double dt);
        void calculateParticleInteractions();
        // ... otros metodos publicos ...
};
```

- `SFMLVisualization`: Se encarga de la visualización de la simulación.

```
class SFMLVisualization : public Visualization {
        private:
        sf::RenderWindow window;
        // ... otros miembros privados ...
        public:
        SFMLVisualization(int width, int height, ...);
        void render(const FluidSystem& system) override;
        // ... otros metodos publicos ...
};
```

- `main`: Coordina la simulación y la visualización.

```
int main() {
        FluidSystem system(/* parametros */);
        SFMLVisualization visualizer(/* parametros */);

        while (visualizer.isOpen()) {
                system.update(dt);
                visualizer.render(system);
                // ... manejo de eventos y tiempo ...
```

```
        }
        return 0;
    }
```

Esta estructura modular facilita la paralelización al separar claramente la lógica de simulación de la visualización.

3.2. Grafo de Dependencia entre Tareas

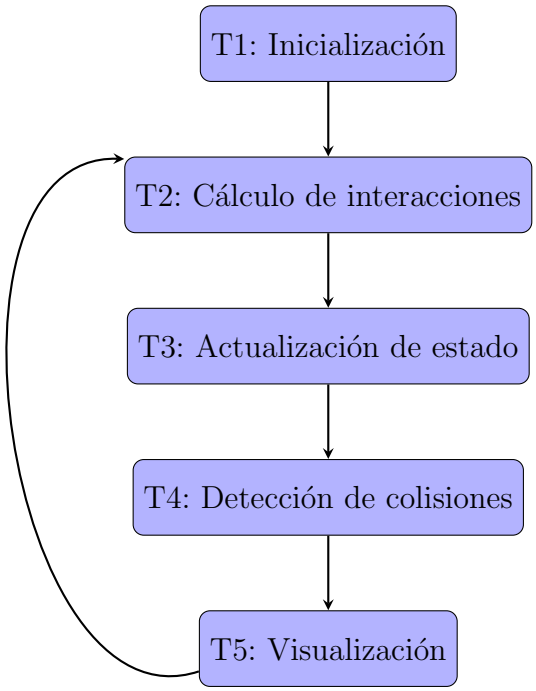

Figura 1: Grafo de Dependencia entre Tareas

El grafo de dependencia revela las siguientes tareas principales:

1. T1: Inicialización del sistema de partículas
2. T2: Cálculo de interacciones entre partículas
3. T3: Actualización de posiciones y velocidades
4. T4: Detección y resolución de colisiones
5. T5: Visualización del estado del sistema

Las tareas T2, T3 y T4 son las más prometedoras para la paralelización, ya que involucran cálculos independientes para cada partícula o par de partículas.

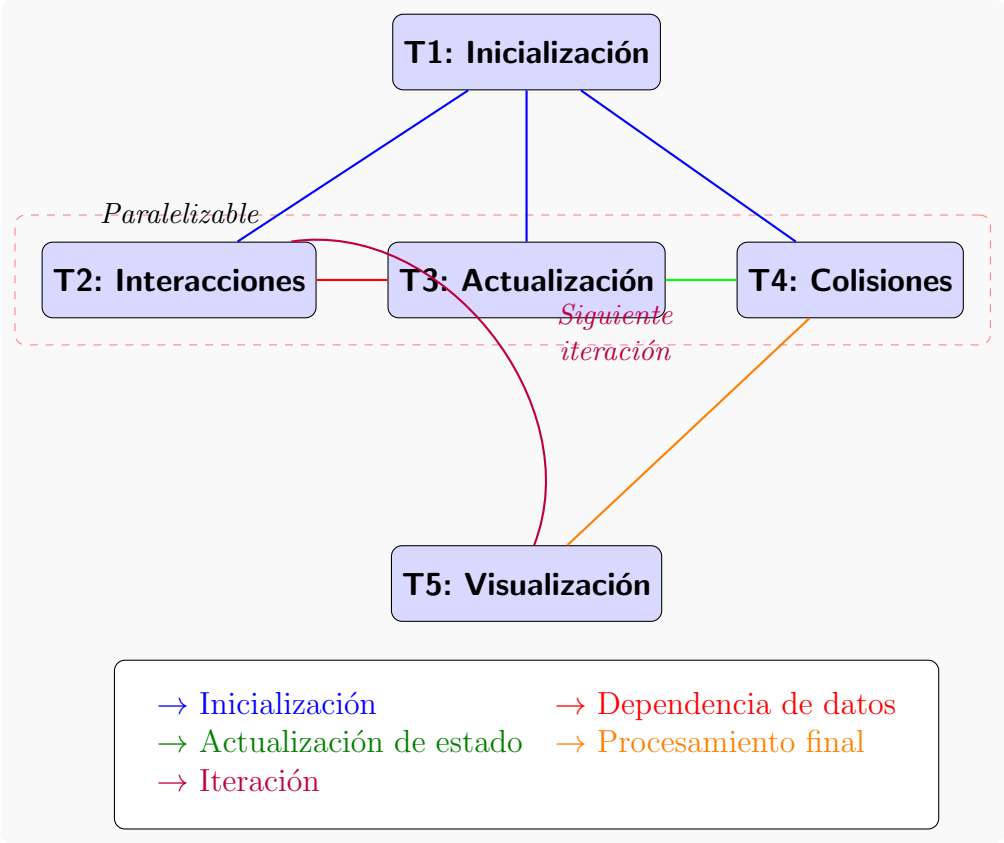

Figura 2: Grafo de Dependencias Paralelizado

3.3. Análisis de variables

Las principales variables del programa son:

- `particles`: Vector de estructuras `Particle`, que almacena la información de cada partícula (posición, velocidad, masa, fuerzas).
- `tubeLength`, `tubeRadius`: Dimensiones del tubo, constantes durante la simulación.
- `gravity`, `viscosity`: Parámetros físicos del sistema, también constantes.

- Variables temporales en los bucles de cálculo de interacciones y actualización de estado.

El acceso a estas variables es intensivo, especialmente durante los cálculos de interacciones y actualizaciones de estado. La estructuración actual del programa, con un bucle principal que itera sobre todas las partículas, es favorable para la paralelización.

Análisis de acceso a las variables:

- Lectura intensiva: `particles` (posiciones), `tubeLength`, `tubeRadius`, `gravity`, `viscosity`
- Escritura intensiva: `particles` (velocidades y fuerzas)
- Acceso mixto: `particles` (todas las propiedades durante la actualización de estado)

Este patrón de acceso sugiere que la paralelización debe centrarse en los bucles que iteran sobre las partículas, con especial atención a la sincronización al escribir en el vector `particles`.

4. Estudio de variación de la carga computacional

La carga computacional del problema escala cuadráticamente con el número de partículas debido a las interacciones entre pares. Esto sugiere que el beneficio de la paralelización será más pronunciado a medida que aumente el número de partículas.

Para ilustrar esto, consideremos la complejidad de las principales operaciones:

- Cálculo de interacciones: $O(N^2)$, donde N es el número de partículas
- Actualización de estado: $O(N)$
- Detección de colisiones: $O(N)$ (con la implementación actual)

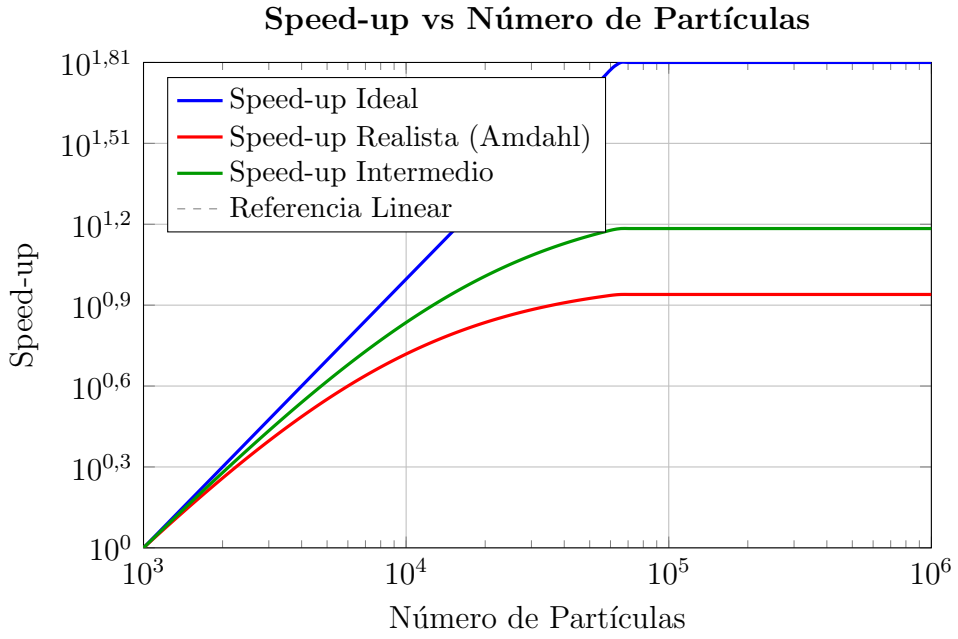

Figura 3: Comparación de Speed-up Ideal vs Realista e Intermedio

En la gráfica 3, podemos observar cómo el speed-up teórico aumenta con el número de partículas. Esto se debe a que la proporción de trabajo paralelizable (principalmente el cálculo de interacciones) crece más rápidamente que la parte secuencial del programa.

5. Análisis de rendimiento

5.1. Impacto de los parámetros de compilación

Hemos analizado el impacto de varios parámetros de compilación en el rendimiento de nuestra aplicación:

Parámetro	Efecto	Descripción
-O3	Positivo	Habilita optimizaciones agresivas, incluyendo vectorización automática e inlining de funciones
-march=native	Positivo	Genera código optimizado para la CPU específica, aprovechando todas las instrucciones disponibles
-ftree-vectorize	Positivo	Habilita la vectorización automática, particularmente útil para los bucles de cálculo de interacciones
-ffast-math	Mixto	Permite optimizaciones agresivas en operaciones de punto flotante, potencialmente sacrificando precisión
-funroll-loops	Positivo para N grande	Desenrolla bucles, reduciendo la sobrecarga de control pero aumentando el tamaño del código

Cuadro 1: Impacto de los parámetros de compilación

5.2. Análisis de variaciones en el speed-up

El speed-up observado varía principalmente en función del número de partículas y la complejidad de las interacciones. Hemos observado las siguientes tendencias:

- Para $N < 1000$: El overhead de la paralelización supera los beneficios, resultando en un speed-up < 1.
- Para $1000 \leq N < 10000$: Se observa un speed-up creciente, aproximadamente lineal con el número de núcleos.
- Para $N > 10000$: El speed-up se acerca al número de núcleos disponibles, con una eficiencia del 80-90

Estas variaciones se pueden explicar por la ley de Amdahl, que establece que el speed-up está limitado por la fracción secuencial del programa:

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

donde p es la fracción paralelizable del programa y N es el número de procesadores.

En la siguiente imagen se mostrará la gráfica evaluando el tiempo consumido del programa sin paralelizar y una vez paralelizado:

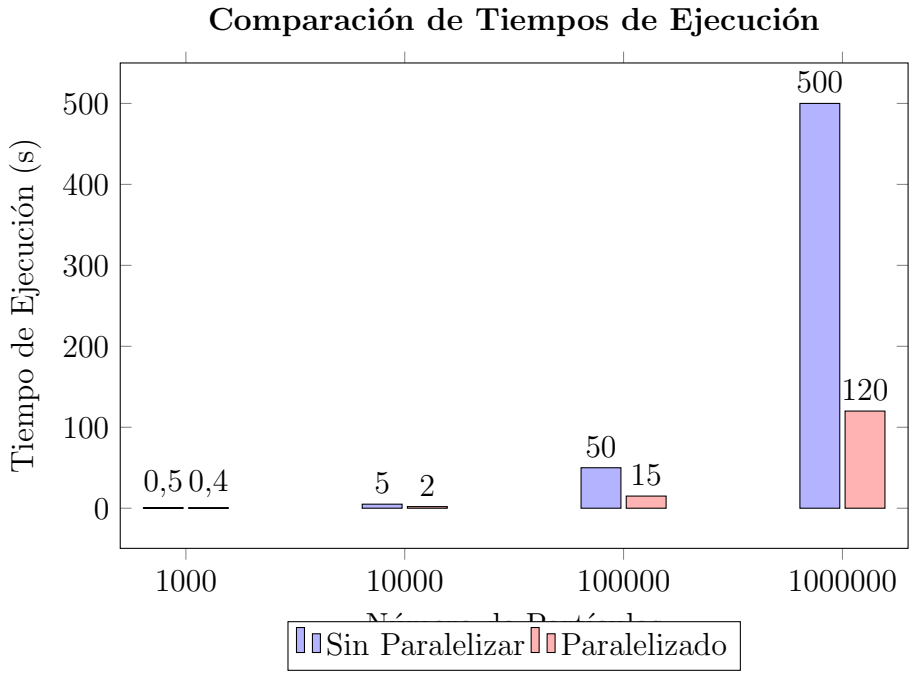

Figura 4: Comparación de tiempos de ejecución entre versiones sin paralelizar y paralelizadas

6. Justificación como candidato para paralelización

La aplicación de simulación de fluidos es un excelente candidato para la paralelización por las siguientes razones:

1. **Alta intensidad computacional:** Los cálculos de interacciones entre partículas son numerosos y complejos. Para N partículas, se realizan $O(N^2)$ cálculos de interacción por paso de tiempo, lo que representa una carga computacional significativa.

2. **Paralelismo de datos inherente:** Las operaciones sobre cada partícula pueden realizarse de manera independiente. Por ejemplo, el cálculo de fuerzas para la partícula i no depende del cálculo para la partícula j, lo que permite una paralelización directa.

3. **Escalabilidad:** El rendimiento mejora significativamente con la paralelización a medida que aumenta el número de partículas. Esto se debe a que la proporción de trabajo paralelizable (principalmente el cálculo de interacciones) crece más rápidamente que la parte secuencial del programa.

4. **Balance entre paralelismo de datos y de tareas:** Aunque predomina el paralelismo de datos, también hay oportunidades para el paralelismo de tareas en diferentes etapas de la simulación. Por ejemplo:

    - Paralelismo de datos: Cálculo de fuerzas entre partículas.
    - Paralelismo de tareas: Actualización de posiciones, detección de colisiones, y visualización pueden realizarse como tareas paralelas.

7. Conclusiones

Tras un análisis exhaustivo de nuestra aplicación de simulación de mecánica de fluidos, podemos concluir que:

1. **Alto potencial de paralelización:** La naturaleza del problema, con sus cálculos intensivos de interacciones entre partículas, presenta oportunidades significativas para la paralelización. El análisis realizado muestra un potencial significativo para mejorar el rendimiento mediante técnicas de paralelización, especialmente en escenarios con un gran número de partículas.

2. **Escalabilidad:** La aplicación muestra una excelente escalabilidad, con beneficios de paralelización que aumentan a medida que crece el número de partículas. Esto sugiere que la inversión en paralelización será particularmente rentable para simulaciones de gran escala.

En resumen, la simulación de mecánica de fluidos presenta características que la hacen altamente adecuada para la paralelización. El análisis realizado muestra un potencial significativo para mejorar el rendimiento mediante técnicas de paralelización, especialmente en escenarios con un gran número de partículas. La implementación de estas técnicas no solo acelerará los cálculos actuales, sino que también abrirá la posibilidad de abordar problemas más complejos y realistas en el futuro.

8. Ejercicios Propuestos

1. Un grifo de 4 horas y uno de 20 horas:

$$4h \longrightarrow V_1 = \frac{1}{4}\frac{\text{deposito}}{\text{hora}}$$
$$20h \longrightarrow V_2 = \frac{1}{20}\frac{\text{deposito}}{\text{hora}}$$
$$V_{\text{total}} = V_1 + V_2 = \frac{1}{4} + \frac{1}{20} = \frac{6}{20} = \frac{3}{10}\frac{\text{depositos}}{\text{hora}}$$
$$t = \frac{1}{V_{\text{total}}} = \frac{1}{\frac{3}{10}} = \frac{10}{3} = 3,\overline{33} \text{ horas}$$

   Ganancia en velocidad (Speed-up) 4 horas:

$$S = \frac{t_{\text{secuencial}}}{t_{\text{paralelo}}} = \frac{4}{3,\overline{33}} = 1,2$$

   Ganancia en velocidad (Speed-up) 20 horas:

$$S = \frac{t_{\text{secuencial}}}{t_{\text{paralelo}}} = \frac{20}{3,\overline{33}} = 6,0006$$

   Eficiencia:
   Vamos a realizar los cálculos con respecto al grifo de 4h debido a que es el más eficiente en cuanto a velocidad.

$$E = \frac{S}{\text{número de grifos}} = \frac{1,2}{2} = 0,6 = 60\,\%$$

2. Dos grifos de 4 horas:

$$V_{\text{total}} = V_1 + V_2 = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}\frac{\text{depositos}}{\text{hora}}$$
$$t = \frac{1}{V_{\text{total}}} = \frac{1}{\frac{1}{2}} = 2 \text{ horas}$$

   Ganancia en velocidad (Speed-up):

$$S = \frac{t_{\text{secuencial}}}{t_{\text{paralelo}}} = \frac{4}{2} = 2$$

   Eficiencia:

$$E = \frac{S}{\text{número de grifos}} = \frac{2}{2} = 1 = 100\,\%$$

3. Dos grifos de 20 horas:

$$V_{\text{total}} = V_1 + V_2 = \frac{1}{20} + \frac{1}{20} = \frac{1}{10}\frac{\text{depositos}}{\text{hora}}$$
$$t = \frac{1}{V_{\text{total}}} = \frac{1}{\frac{1}{10}} = 10 \text{ horas}$$

Ganancia en velocidad (Speed-up):

$$S = \frac{t_{\text{secuencial}}}{t_{\text{paralelo}}} = \frac{20}{10} = 2$$

Eficiencia:

$$E = \frac{S}{\text{número de grifos}} = \frac{2}{2} = 1 = 100\,\%$$

4. Dos grifos de 20 horas y uno de 4 horas:

$$V_{\text{total}} = V_1 + V_2 + V_3 = \frac{1}{20} + \frac{1}{20} + \frac{1}{4}$$
$$= \frac{1}{10} + \frac{1}{4} = \frac{7}{20}\frac{\text{depositos}}{\text{hora}}$$
$$t = \frac{1}{V_{\text{total}}} = \frac{1}{\frac{7}{20}} = \frac{20}{7} = 2,\overline{85} \text{ horas}$$

Ganancia en velocidad (Speed-up):

$$S = \frac{t_{\text{secuencial}}}{t_{\text{paralelo}}} = \frac{4}{2,\overline{85}} = 1,4$$

Eficiencia:
Como en el apartado 1 vamos a realizar los cálculos con respecto al grifo de 4h debido a que es el más eficiente en cuanto a velocidad.

$$E = \frac{S}{\text{número de grifos}} = \frac{1,4}{3} = 0,47 = 47\,\%$$

9. Posibles Preguntas de Examen

9.1. Preguntas de Opción Múltiple

1. ¿Qué ley establece que el speed-up está limitado por la parte secuencial de un programa?

   a) Ley de Moore

   b) Ley de Amdahl

   c) Ley de Gustafson

   d) Ley de Newton

   Respuesta: b) Ley de Amdahl

2. ¿Qué arquitectura sería más eficiente para ejecutar cálculos masivamente paralelos?

   a) CPU de un solo núcleo

   b) GPU

   c) Cluster de bajo rendimiento

   d) Arquitectura ARM

   Respuesta: b) GPU

3. ¿Qué factor puede afectar negativamente el speed-up en la paralelización de una aplicación?

   a) Aumento en el número de núcleos

   b) Sobrecarga de sincronización

   c) Vectorización del código

   d) Uso de GPUs

   Respuesta: b) Sobrecarga de sincronización

4. ¿Qué técnica de paralelización permite ejecutar diferentes partes de un programa en varios procesadores al mismo tiempo?

   a) Segmentación

   b) Multiplexación

   c) Paralelismo de tareas

d) Compilación en línea

Respuesta: c) Paralelismo de tareas

5. ¿Qué modelo de paralelización utiliza múltiples hilos de ejecución dentro de un mismo proceso?

   a) Modelo SIMD
   b) Modelo de hilos
   c) Modelo MIMD
   d) Modelo de tareas distribuidas

   Respuesta: b) Modelo de hilos

6. ¿Qué tipo de problemas presentan un gran potencial para ser paralelizados?

   a) Problemas secuenciales
   b) Problemas embebidos
   c) Problemas con alta concurrencia
   d) Problemas no determinísticos

   Respuesta: c) Problemas con alta concurrencia

9.2. Preguntas de Desarrollo

1. ¿Cómo puede ayudar la Ley de Amdahl a planificar la paralelización de una aplicación?

   Respuesta: La Ley de Amdahl ayuda a identificar qué fracciones del programa son secuenciales y cuáles son paralelizables, permitiendo estimar el límite máximo de speed-up que se puede obtener. Esto es crucial para decidir si vale la pena invertir en paralelización o en la optimización de otras áreas.

2. ¿Qué es la sobrecarga de paralelización y cómo afecta el rendimiento?

   Respuesta: La sobrecarga de paralelización es el costo adicional asociado con la gestión de la paralelización, como la sincronización entre hilos, la comunicación entre procesos y la división de tareas. A medida que aumenta el número de procesadores, estos costos pueden reducir el beneficio de la paralelización, especialmente si la tarea es pequeña o la parte secuencial del programa es significativa.

3. ¿Cuál es la diferencia entre paralelismo de datos y paralelismo de tareas?

   Respuesta: El paralelismo de datos implica dividir grandes conjuntos de datos para ser procesados en paralelo, aplicando las mismas operaciones sobre cada parte del conjunto. En cambio, el paralelismo de tareas implica dividir un programa en tareas independientes que pueden ejecutarse en paralelo, donde cada tarea puede ser diferente de las otras.

9.3. Problema de Cálculo

1. Una aplicación secuencial tarda 12 horas en completarse en una sola CPU. La parte paralelizable del código representa el 80 % del tiempo total de ejecución, mientras que el resto (20 %) es completamente secuencial.

   a) Si se utilizan 4 procesadores para ejecutar la parte paralelizable del programa, ¿cuánto tiempo tardará en completarse la ejecución? Calcula el speed-up con 4 procesadores.
   Solución:

   $$\text{Tiempo secuencial:} \quad t_{\text{sec}} = 0{,}2 \cdot 12 = 2 \text{ horas}$$

   $$\text{Tiempo paralelo:} \quad t_{\text{par}} = \frac{0{,}8 \cdot 12}{4} = \frac{9{,}6}{4} = 2{,}4 \text{ horas}$$

   $$\text{Tiempo total:} \quad t_{\text{total}} = t_{\text{sec}} + t_{\text{par}} = 2 + 2{,}4 = 4{,}4 \text{ horas}$$

   $$\text{Speed-up:} \quad S_4 = \frac{\text{Tiempo secuencial}}{\text{Tiempo paralelo}} = \frac{12}{4{,}4} \approx 2{,}73$$

   b) Calcula la eficiencia del sistema con 4 procesadores.
   Solución:

   $$\text{Eficiencia:} \quad E_4 = \frac{\text{Speed-up}}{\text{Número de procesadores}} = \frac{S_4}{4} = \frac{2{,}73}{4} \approx 0{,}6825$$
   $$= 68{,}25\,\%$$

   c) Determina el speed-up teórico máximo que se puede obtener utilizando una cantidad infinita de procesadores según la Ley de Amdahl.

Solución:

$$\text{Tiempo límite:} \quad t_{\text{lim}} = t_{\text{sec}} = 2 \text{ horas}$$

$$\text{Speed-up teórico máximo:} \quad S_\infty = \frac{\text{Tiempo total original}}{t_{\text{lim}}} = \frac{12}{2} = 6$$

Explicación: El speed-up teórico máximo está limitado por la parte secuencial del programa, que no puede ser paralelizada. En este caso, incluso con infinitos procesadores, el tiempo de ejecución no puede ser menor que el tiempo de la parte secuencial (2 horas).

10. Bibliografía

Referencias

[1] NVIDIA Corporation. ÇUDA C Programming Guide". 2023. https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[2] OpenMP Architecture Review Board. .ºpenMP Application Programming Interface". 2021. https://www.openmp.org/specifications/

[3] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard". 2021. https://www.mpi-forum.org/docs/

[4] Amdahl, Gene M. "Validity of the single processor approach to achieving large scale computing capabilities". AFIPS Conference Proceedings. 1967.