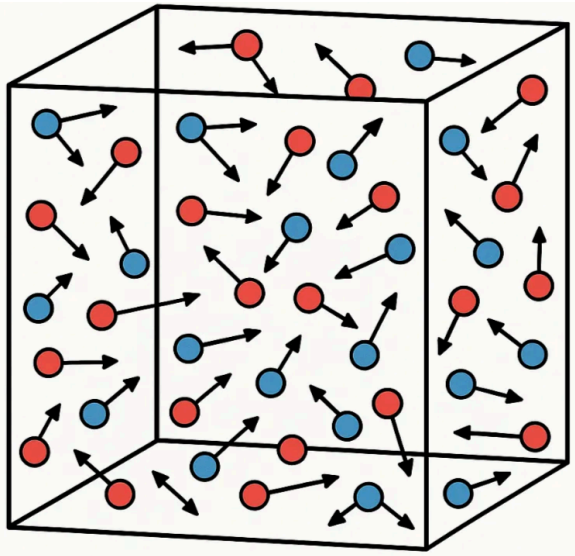
Simulación de un sistema de Dinámica Molecular

En mi tarea estamos simulando un sistema dinámico mólecular utilizando el potencial de Lennard-Jones y acelerar dicha simulación mediante técnicas de paralelización con OpenMP en el lenguaje C.

En esencia, estás creando un "universo en una caja" a nivel microscópico para ver cómo se comportan las partículas (átomos o moléculas) y demostrando que puedes hacer que esa simulación se ejecute mucho más rápido usando múltiples núcleos de un procesador.



Planteamiento del Problema

La simulación de dinámica molecular es una herramienta fundamental en la ciencia computacional para predecir el comportamiento de materiales, fármacos y sistemas biológicos a nivel atómico. Este proyecto aborda la simulación de un sistema de `N` partículas cuya interacción se rige por el potencial de Lennard-Jones.

El principal obstáculo computacional de este problema reside en el cálculo de las fuerzas de interacción. Un enfoque directo requiere que para cada partícula se calcule la fuerza ejercida por todas las demás, lo que resulta en una **complejidad computacional de O(N2)**. Este crecimiento cuadrático convierte rápidamente a las simulaciones de sistemas grandes (con miles o millones de partículas) en un proceso **computacionalmente intratable** en arquitecturas de un solo procesador, limitando severamente el alcance y la viabilidad de la investigación científica.

Para darle una mejor imagen de lo qué es primero hay qué comprender la complejidad cuadrática de **$O(n^2)$**, describe sistemas donde el costo computacional es proporcional al cuadrado del número de elementos (n). No se refiere a la velocidad de las partículas, sino a cómo escala el número total de interacciones del sistema. Esto ocurre cuando cada elemento debe ser comparado o interactuar con todos los demás. Como resultado, al duplicar los elementos, el esfuerzo se cuadruplica, volviéndolo muy ineficiente para grandes conjuntos de datos, como en simulaciones gravitacionales o algoritmos básicos.

Para qué informáticos como nosotros entendamos te lo planteo de la siguiente manera.

Tenemos qué:

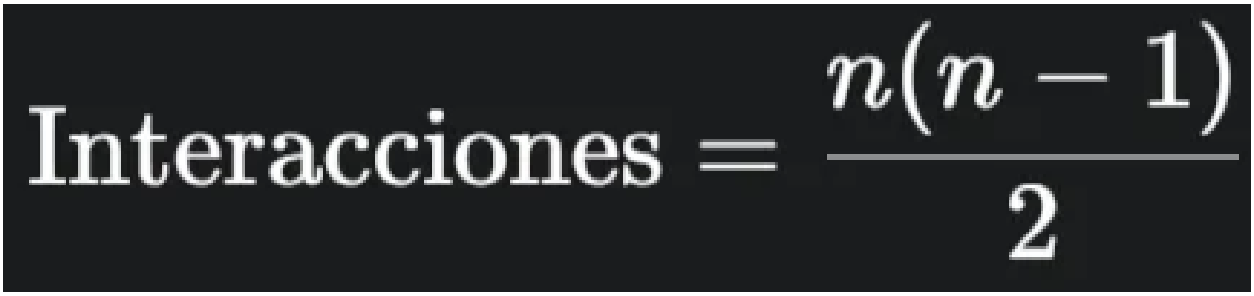
```
def proceso_cuadratico(lista):
    n = len(lista)
    contador_operaciones = 0
    for i in range(n):
        for j in range(n):
            contador_operaciones += 1
    return contador_operaciones
```

▼ ← Por aquí comentado

```
def encontrar_pares(elementos):
    """
    Esta función demuestra un comportamiento O(n^2)
    al comparar cada elemento con todos los demás en la lista.
    """
```

```
    # n es el número de elementos en la lista.
    n = len(elementos)
    contador_operaciones = 0

    print(f"Procesando una lista de {n} elementos.")

    # Primer bucle (externo): se ejecuta n veces.
    for i in range(n):
        # Segundo bucle (anidado): también se ejecuta n veces por cada
        # iteración del bucle externo.
        for j in range(n):
            # Realizamos una operación simple por cada par (i, j).
            # Por ejemplo, una comparación o un cálculo.
            # No imprimimos para no saturar la salida, solo contamos.
            contador_operaciones += 1
            # print(f"Comparando elemento {elementos[i]} con {elementos[j]}")

    print(f"Total de operaciones realizadas: {contador_operaciones} (que es n
    print("-" * 20)

# --- Pruebas ---
encontrar_pares([1, 2, 3])  # n=3. Operaciones = 3*3 = 9
encontrar_pares([1, 2, 3, 4, 5]) # n=5. Operaciones = 5*5 = 25
encontrar_pares([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) # n=10. Operaciones = 10*10 = 1
```

Básicamente sé está resolviendo la ecuación



Dónde las partículas interactúan n(n-1)/2) con las partículas, átomos y moléculas.

Cómo lo solucionaremos?

Vamos a utilizar La Fuerza de Lennard-Jones (Para calcular las interacciones) dónde estaremos simulando **n** moléculas neutras (como átomos de Argón) en una caja. Queremos calcular la energía potencial total del sistema en un instante dado.

Objetivo

El objetivo de este proyecto es simular un sistema de
N moléculas neutras (como átomos de Argón) confinadas en una caja cúbica. La simulación calcula la trayectoria de estas moléculas a lo largo del tiempo, permitiendo analizar propiedades del sistema como su energía. Para lograrlo, se combinan un modelo de fuerzas intermoleculares, un método de integración numérica y condiciones de frontera específicas.

1. Energía Potencial del Sistema

La interacción entre cada par de moléculas se modela con el **potencial de Lennard-Jones**. La energía potencial (Uij) entre dos moléculas (i y j) separadas por una distancia (rij) se define como:

$$U(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \quad \text{si} \quad r_{ij} \leq r_c$$

Y se considera U(rij)=0 si rij>rc.

- **Parámetros:**

  ϵ (Épsilon): La profundidad del pozo de potencial, representa la fuerza de la atracción.

  σ (Sigma): El diámetro efectivo de las moléculas.

  - Se calcula `r2` = rij2.

- Se calcula `r6_inv` = (r2σ2)3, que es equivalente a (rijσ)6.
- Se calcula `r12_inv` = `r6_inv` * `r6_inv`, equivalente a (rijσ)12.
- La energía de este par se suma a la energía total: `potential_energy += 4.0 * EPSILON * (r12_inv - r6_inv)`.

Para entenderlo mejor en python tenemos qué

```
# Definición del potencial de Lennard-Jones en Python
def lennard_jones(r_ij, epsilon, sigma, r_c):
    """
    Calcula el potencial de Lennard-Jones para una distancia r_ij
    y devuelve 0 si r_ij > r_c (potencial truncado).
    """
    if r_ij <= r_c:
        # Fórmula: 4 * epsilon * [ (sigma/r)^12 - (sigma/r)^6 ]
        term1 = (sigma / r_ij)**12
        term2 = (sigma / r_ij)**6
        U = 4 * epsilon * (term1 - term2)
        return U
    else:
        # Potencial truncado a cero si supera el radio de corte
        return 0.0
```

La **energía potencial total** del sistema (Utotal) en un instante dado es la suma de las energías de todos los pares únicos de partículas que se encuentran dentro de este radio de corte. Matemáticamente, esto es lo que calcula el bucle principal de la función `calculate_forces_and_potential_parallel`:

$$U_{total} = \sum_{i=1}^{N-1} \sum_{j=i+1, r_{ij} \le r_c}^{N} U(r_{ij})$$

En python sería

```
def energia_total(N, posiciones, epsilon, sigma, r_c):
    """
    Calcula la energía potencial total U_total de un sistema de N partículas.
    posiciones: lista de posiciones (listas o arrays) de las partículas.
    """
    U_total = 0.0
    for i in range(N-1):
        for j in range(i+1, N):
            # Calcular distancia rij entre partículas i y j
            rij = distancia(posiciones[i], posiciones[j])
            if rij <= r_c:
                # Sumar la energía Lennard-Jones si rij <= r_c
                U_total += lennard_jones(rij, epsilon, sigma, r_c)
    return U_total

def distancia(pos1, pos2):
    """Calcula la distancia euclidiana entre dos posiciones."""
    dx = pos2[0] - pos1[0]
    dy = pos2[1] - pos1[1]
    dz = pos2[2] - pos1[2]
    return (dx**2 + dy**2 + dz**2)**0.5
```

2. Fuerza Intermolecular

La fuerza es la derivada negativa de la energía potencial. A partir de la ecuación de Lennard-Jones, la **magnitud de la fuerza** (Fij) que una partícula ejerce sobre otra es:

$$F(r_{ij}) = \frac{24\epsilon}{r_{ij}^2}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \cdot r_{ij} = \frac{24\epsilon}{r_{ij}}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

O lo podemos interpretar de la siguiente manera:

```
def energia_total(N, posiciones, epsilon, sigma, r_c):
    """
    Calcula la energía potencial total U_total de un sistema de N partículas.
    posiciones: lista de posiciones (listas o arrays) de las partículas.
    """
    U_total = 0.0
    for i in range(N-1):
        for j in range(i+1, N):
            # Calcular distancia rij entre partículas i y j
            rij = distancia(posiciones[i], posiciones[j])
            if rij <= r_c:
                # Sumar la energía Lennard-Jones si rij <= r_c
                U_total += lennard_jones(rij, epsilon, sigma, r_c)
    return U_total

def distancia(pos1, pos2):
    """Calcula la distancia euclidiana entre dos posiciones."""
    dx = pos2[0] - pos1[0]
    dy = pos2[1] - pos1[1]
    dz = pos2[2] - pos1[2]
    return (dx**2 + dy**2 + dz**2)**0.5
```

Esta es la magnitud. La **fuerza como vector** (Fij) se obtiene multiplicando esta magnitud por la
dirección que une las partículas. En el código, esto se implementa eficientemente calculando
`Vector force_vec = {force_mag * dr.x, ...}`, donde `dr` es el vector de distancia.

La **fuerza total** sobre una partícula i (Ftotal,i) es la suma vectorial de las fuerzas ejercidas por todas las demás partículas j que se encuentren dentro de su radio de corte:

→

→

$$F_{total,i} = \sum_{j \neq i,\, r_{ij} \leq r_c} F_{ij}$$

3. Dinámica y Evolución Temporal

Para mover las partículas en el tiempo, se utiliza el

algoritmo de Velocity Verlet, un integrador numérico que resuelve la segunda ley de Newton (F=ma). Este método es robusto y conserva bien la energía.

→

→

El algoritmo se implementa en las funciones `update_positions` y `update_velocities` siguiendo estas dos ecuaciones para cada paso de tiempo Δt:

1. **Actualización de Posición:** Se calcula la nueva posición r(t+Δt) a partir de la posición, velocidad y aceleración actuales (a(t)=F(t)/m).
   r(t+Δt)=r(t)+v(t)Δt+21a(t)Δt2

   →

   →

   →

   →

   →

   →

   →

2. **Actualización de Velocidad:** Se calcula la nueva velocidad v(t+Δt) usando un promedio de la aceleración vieja (a(t)) y la nueva (a(t+Δt), calculada con las fuerzas en las nuevas posiciones).
   v(t+Δt)=v(t)+21[a(t)+a(t+Δt)]Δt

→

→

→

→

→

→

→

4. Geometría de la Simulación: Condiciones de Frontera Periódicas

Para simular un material "a granel" y evitar que las partículas se escapen o choquen con paredes, la caja de simulación se trata como si se repitiera infinitamente en todas las direcciones.

Cuando se calcula la distancia entre dos partículas i y j, no se usa su distancia absoluta, sino la distancia a la **imagen más cercana** de la otra partícula. Esto se implementa en el código para cada componente del vector de distancia (ej. para x) con la siguiente lógica: si la distancia Δx=xi−xj es mayor que la mitad de la caja (L/2), se le resta el tamaño de la caja (L). Si es menor que −L/2, se le suma L.

Esto equivale a la fórmula matemática:

→

→

→

$$\Delta x' = \Delta x - L \cdot \text{round}\left(\frac{\Delta x}{L}\right)$$