

# Aceleración de la técnica de Monte Carlo para integración numérica usando memoria compartida y memoria distribuida

Edwin Urbina Quiroz  
Ervin Villalta Cáceres  
Isao Núñez Okubo

Universidad de Costa Rica

December 4, 2025

# Contenido

- 1 Introducción
- 2 Implementación secuencial
- 3 Versión interactiva
- 4 Paralelización (espacios)
- 5 Conclusiones

# Motivación del proyecto

- La integración numérica tradicional se vuelve muy costosa en altas dimensiones.
- La técnica de Monte Carlo reduce este costo usando muestreo aleatorio.
- Objetivo del proyecto:
  - Implementar la integración de Monte Carlo multidimensional.
  - Acelerar el método usando memoria compartida y memoria distribuida.
  - Estudiar la escalabilidad del método.
- Nota: en clase se vio una versión de “Monte Carlo de punto medio”, pero en el proyecto se implementa la variante estándar por muestreo uniforme directo, que es la adecuada para parallelización.

## Idea general del método

- Se desea calcular una integral multidimensional:

$$I = \int_{[a,b]^d} f(\mathbf{x}) d\mathbf{x}.$$

- Se reescribe como valor esperado:

$$I = V \mathbb{E}[f(\mathbf{X})], \quad V = (b - a)^d.$$

- Aproximación por Monte Carlo:

$$\hat{I}_N = V \frac{1}{N} \sum_{i=1}^N f(\mathbf{X}_i).$$

- Ventaja: el error decrece como  $N^{-1/2}$  y no depende de la dimensión.

# Función objetivo utilizada

- Se toma como función de prueba:

$$f(\mathbf{x}) = \exp\left(-\sum_{i=1}^d x_i^2\right).$$

- Aparición en:
  - física estadística,
  - probabilidad multivariada,
  - integrales gaussianas en altas dimensiones.
- Es una función suave y bien comportada para estudiar convergencia y error.

## Generación de puntos aleatorios

- Se usa el generador `std::mt19937` (Mersenne Twister).
- Se genera un uniforme en  $[0, 1]$ :

$$r = \frac{\text{generador}()}{\text{generador}.max()}.$$

- Se escala al intervalo  $[a, b]$ :

$$x_i = a + (b - a) r.$$

- El proceso se repite para cada dimensión y para cada punto.

## Código principal del método

```
for (int i = 0; i < N; i++) {  
    std::vector<double> punto(dimensiones);  
    for (int d = 0; d < dimensiones; d++) {  
        double r = double(generador()) /  
                   double(generador.max());  
        punto[d] = lim_inf + (lim_sup - lim_inf)*r;  
    }  
  
    double valor_final = func(punto);  
    suma_final += valor_final;  
    suma_final2 += valor_final*valor_final;  
}
```

# Cálculo de la integral y del error

- Promedio:

$$\hat{f} = \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}_i).$$

- Varianza:

$$\hat{\sigma}^2 = \mathbb{E}[f^2] - (\mathbb{E}[f])^2 \approx \frac{1}{N} \sum f(\mathbf{x}_i)^2 - \hat{f}^2.$$

- Integral estimada:

$$I \approx V\hat{f}.$$

- Error estadístico:

$$\Delta I = V \sqrt{\frac{\hat{\sigma}^2}{N}}.$$

## Versión con argumentos de línea de comandos

- El programa permite elegir parámetros desde la terminal:

```
./mc --li 0 --ls 1 --d 3 --n 5000000
```

- Parámetros:

- $--li$ : límite inferior.
- $--ls$ : límite superior.
- $--d$ : número de dimensiones.
- $--n$ : número de puntos  $N$ .

- Esta versión facilita:

- barrer distintos valores de  $N$ ,
- cambiar dimensión  $d$ ,
- automatizar experimentos para medir tiempos.

# Memoria compartida (espacio reservado)

**Aquí debe ir la parte de memoria compartida.**

Sugerencias de contenido:

- Explicar la idea de paralelizar el ciclo de Monte Carlo usando hilos.
- Mostrar brevemente el uso de OpenMP (por ejemplo, `#pragma omp parallel for`).
- Comentar el uso de `reduction` para `suma_final` y `suma_final2`.
- Presentar tiempos de ejecución con 1, 2, 4, 8 hilos y el speedup observado.

# Memoria distribuida (espacio reservado)

**Aquí debe ir la parte de memoria distribuida.**

Sugerencias de contenido:

- Describir cómo se reparte el total de  $N$  puntos entre varios procesos.
- Explicar el uso de MPI: cada proceso calcula una parte de la suma.
- Mencionar funciones como MPI\_Reduce para combinar resultados.
- Discutir el impacto de la comunicación entre procesos.

# Escalabilidad (espacio reservado)

**Aquí debe ir el análisis de escalabilidad.**

Sugerencias de contenido:

- Definir speedup:  $S(p) = T(1)/T(p)$ .
- Definir eficiencia:  $E(p) = S(p)/p$ .
- Mostrar alguna tabla o gráfica con tiempos para distintos números de hilos/procesos.
- Comparar comportamiento en memoria compartida vs. distribuida.

# Conclusiones del proyecto

- El método de Monte Carlo es adecuado para integrales multidimensionales.
- La implementación secuencial sirve como referencia para el análisis de rendimiento.
- La parallelización en memoria compartida y distribuida permite reducir significativamente el tiempo de cómputo.
- El estudio de la escalabilidad muestra los límites prácticos de la aceleración.

Gracias

¡Preguntas?