Experimentos de Estimación de π con Programación Paralela y Secuencial

Juan Diego Collazos Mejia, Juan Sebastian Garizao Puerto, Oscar Vargas Pabon

August 17, 2025

Descripción

Este documento recoge los resultados experimentales de tres programas en C++:

- **taylorpi.cpp**: paralelo, método de Taylor (requiere iteraciones y número de hilos).
- **carlopi.cpp**: paralelo, método Monte Carlo (requiere número de lanzamientos y número de hilos).
- **montepi.cpp**: secuencial, método Monte Carlo (requiere número de lanzamientos).

Se registraron los tiempos de ejecución:

- **CPU time** medido con `clock()`.
- **Wall-clock time** medido con `time()`.

Además, se calculó el error absoluto comparando con el valor de π definido en `math.h`.

Resultados `taylorpi.cpp` (paralelo – serie de Taylor)

Iteraciones (N)	Hilos	Tiempo CPU (s)	Tiempo Wall (s)	Valor estimado π	Error absoluto
1e6	2	0.005381	0.0028035	3.14159165	1.000e-6
1e6	4	0.005848	0.0030669	3.14159165	1.000e-6
1e6	8	0.006474	0.0019328	3.14159165	1.000e-6
1e7	2	0.049728	0.0267203	3.14159255	1.000e-7
1e7	4	0.046440	0.0122906	3.14159255	1.000e-7
1e7	8	0.048715	0.0069969	3.14159255	9.999e-8
1e8	2	0.37089	0.187539885	3.14159264	9.9995425e-9
1e8	4	0.39799	0.108343164	3.14159264	9.9999759e-9
1e8	8	0.42705	0.077586926	3.14159264	9.9999133e-9

`taylorpi.cpp` in GitHub

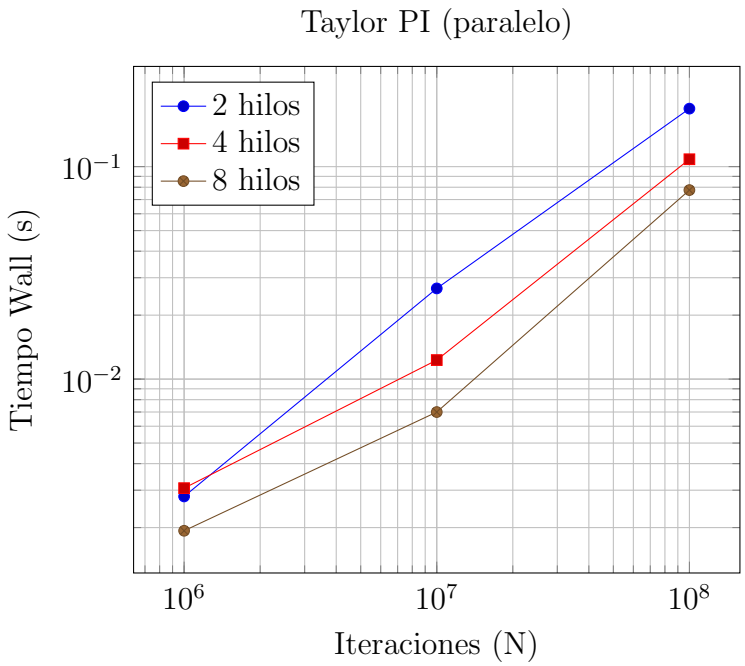

Figure 1: Tiempo de ejecución Wall para Taylor PI con distintos hilos

Observaciones:

- El tiempo Wall disminuye significativamente al aumentar los hilos.
  - Ejemplo en 10^8: pasa de $0.187s$ (2 hilos) $\rightarrow$ $0.077s$ (8 hilos).
- El tiempo CPU se mantiene relativamente constante (porque refleja el trabajo total hecho por todos los hilos).
- La precisión mejora con más iteraciones, y el error absoluto baja hasta el orden de 10^{-9}

Conclusión: la paralelización escala muy bien en este algoritmo (casi ideal).

Resultados `carlopi.cpp` (paralelo – Monte Carlo)

Iteraciones (N)	Hilos	Tiempo CPU (s)	Tiempo Wall (s)	Valor estimado π	Error absoluto
1e6	2	0.47819	0.241579655	3.142600	0.001007346
1e6	4	0.64658	0.175579842	3.140444	0.001148653
1e6	8	0.94419	0.126774304	3.140340	0.001252653
1e7	2	4.61655	2.320695875	3.141303	0.000289453
1e7	4	6.69396	1.698227549	3.141563	2.9053589e-05
1e7	8	9.39104	1.233206821	3.141708	0.000115346
1e8	2	50.4320	25.42126513	3.14185916	0.000266506
1e8	4	69.4330	17.56811539	3.14148572	0.000106933
1e8	8	92.9908	12.45675765	3.1411492	0.0004434535

`carlopi.cpp` in GitHub

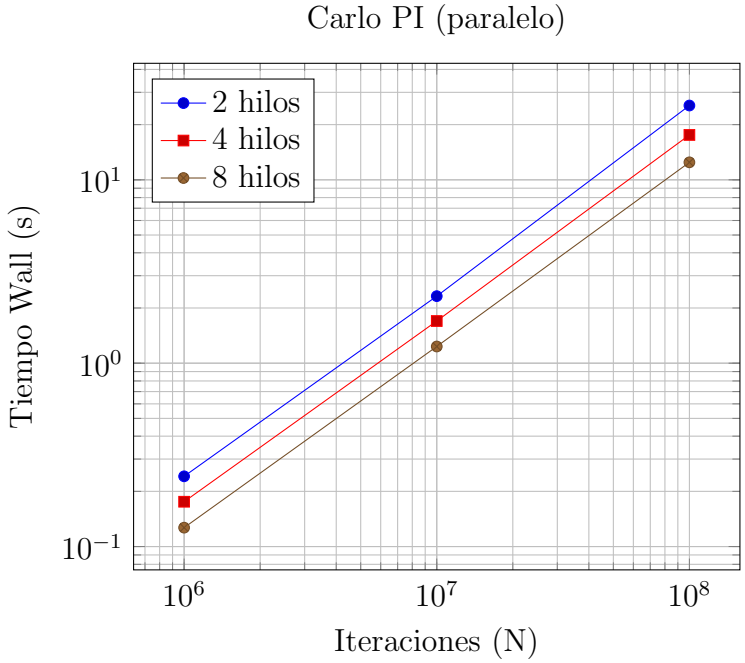

Figure 2: Tiempo de ejecución Wall para Carlo PI con distintos hilos

Observaciones:

- El tiempo Wall sí mejora con más hilos, pero no tanto como en Taylor.
  - Ejemplo en 10^8: $25.4s$ (2 hilos) $\rightarrow$ $12.4s$ (8 hilos) $\rightarrow$ $2x$ más rápido, pero no lineal.
- El tiempo CPU crece notablemente con los hilos: muestra overhead de sincronización o gestión de threads.
- La precisión mejora con las iteraciones, pero sigue limitada por la naturaleza estocástica del método.

Conclusión: paraleliza, pero con eficiencia más baja que Taylor.

Resultados `montepi.cpp` (secuencial – Monte Carlo)

Iteraciones (N)	Tiempo CPU (s)	Tiempo Wall (s)	Valor estimado π	Error absoluto
1e6	0.35890	0.3608	3.14064	0.000952654
1e7	3.79702	3.8192	3.14157	2.74536e-05
1e8	38.3042	38.5914	3.1416	8.66641e-06

`montepi.cpp` in GitHub

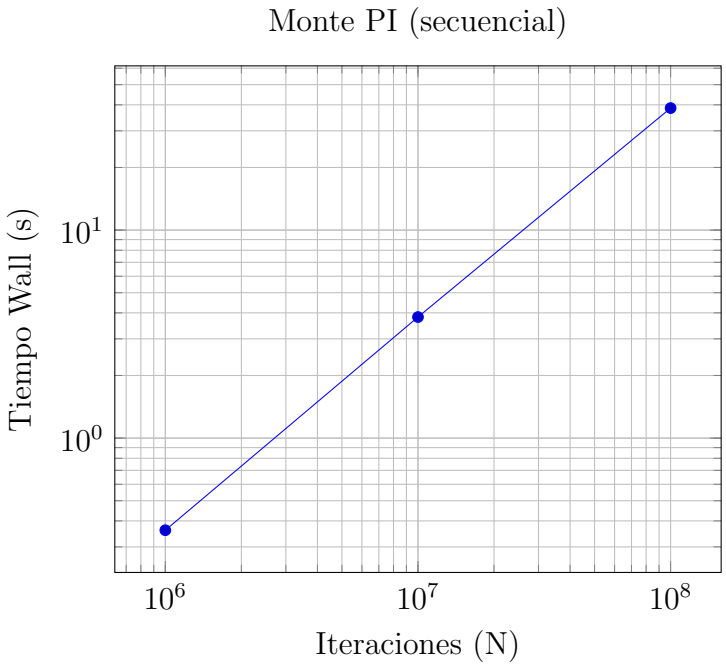

Figure 3: Tiempo de ejecución Wall para Monte PI secuencial

Observaciones:

- Escala de manera lineal con las iteraciones (tiempo se multiplica 10x cuando las iteraciones suben 10x).
- Mucho más lento que Taylor para alcanzar la misma precisión.
  - Ejemplo: en 10^8, tarda $38s$ (secuencial) vs 0.07–0.18 s (Taylor paralelo).

Conclusión: el método secuencial Monte Carlo es poco competitivo en precisión/tiempo frente a los paralelos.

Comparación General de los Métodos

1. Precisión

Taylor (paralelo):
El error absoluto decrece rápidamente con las iteraciones: llega al orden de 10^{-9} en 10^8. Esto lo convierte en el método más preciso y estable.

Carlo (paralelo, Monte Carlo):
La precisión mejora con las iteraciones, pero de manera mucho más lenta. Incluso en 10^8, el error se mantiene alrededor de 10^{-4} – 10^{-5}. Está limitado por la naturaleza aleatoria del método.

Monte (secuencial, Monte Carlo):
Similar al Carlo paralelo en precisión, pero más lento. A 10^8, obtiene error cercano a 10^{-5}.

Conclusión en precisión: Taylor domina, los métodos Monte Carlo son menos competitivos.

2. Escalabilidad y tiempos de ejecución

Taylor paralelo:
Escala casi ideal con el número de hilos. Ejemplo con 10^8 iteraciones: de 0.187 s (2 hilos) $\rightarrow$ 0.077 s (8 hilos), lo que representa aproximadamente un *speedup* real de 2.4×. El tiempo de CPU se mantiene casi constante, lo que indica una paralelización eficiente.

Carlo paralelo:
Escala peor que Taylor debido al overhead de sincronización y la generación de números aleatorios. Ejemplo con 10^8: de 25.4 s (2 hilos) $\rightarrow$ 12.4 s (8 hilos), equivalente a un *speedup* real de $\sim 2\times$. El tiempo de CPU crece considerablemente, lo que refleja menor eficiencia.

Monte secuencial:
Escala linealmente con las iteraciones, al no incluir paralelismo. En 10^8 iteraciones, tarda 38.6 s, siendo mucho más lento que ambos métodos paralelos.

Conclusión en tiempos: Taylor paralelo es el más eficiente; Carlo paralelo se beneficia del paralelismo pero no escala tan bien; Monte secuencial es el menos competitivo.

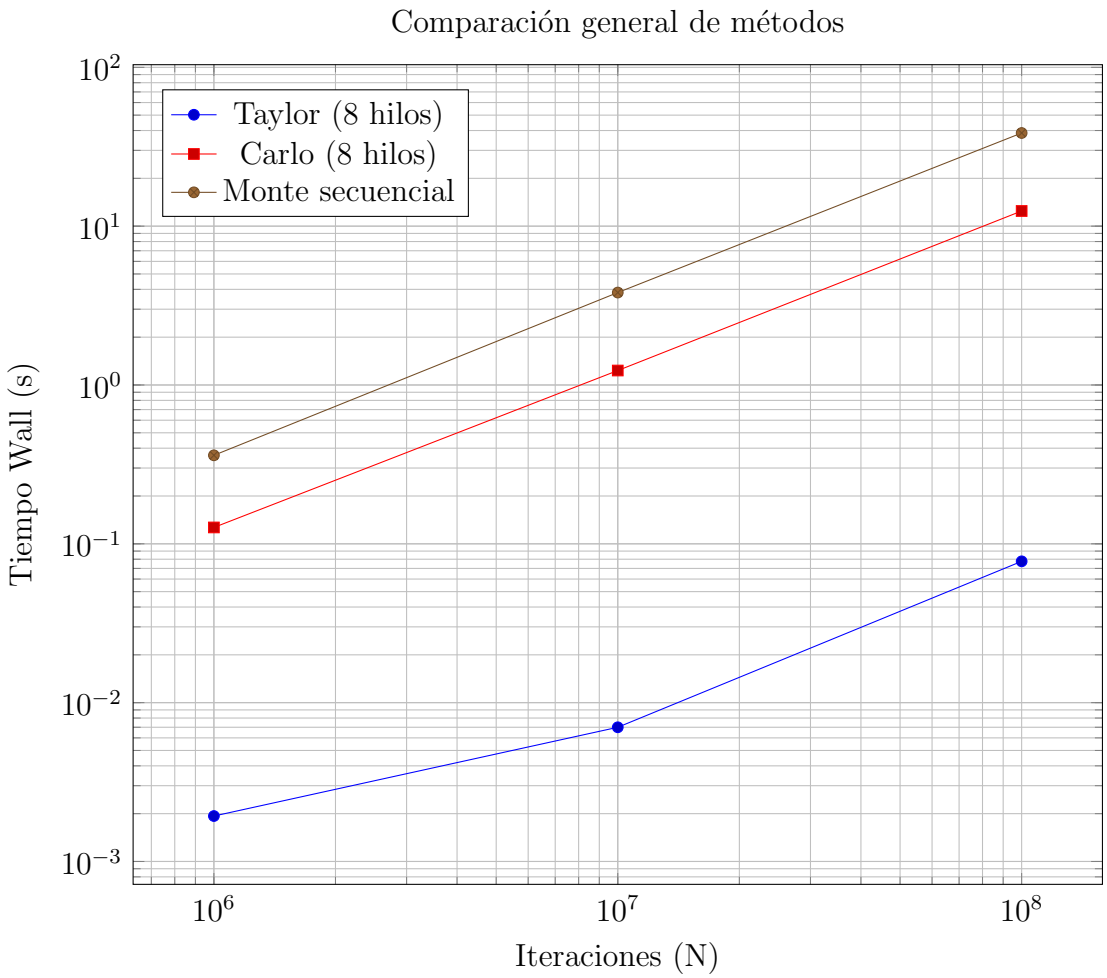

Figure 4: Comparación de tiempos Wall entre los tres métodos más representativos

Conclusión General

- **Taylor paralelo:** es el más rápido y preciso, además de escalar casi idealmente.
- **Carlo paralelo:** mejora con hilos, pero la eficiencia es más baja debido al *overhead* y la aleatoriedad.
- **Monte secuencial:** funciona, pero es ineficiente comparado con los demás.

Para aplicaciones prácticas donde se busca **precisión y eficiencia**, Taylor paralelo es claramente superior.

Carlo paralelo solo tiene sentido en entornos donde el método estocástico sea requerido, como en **simulaciones probabilísticas**.