# Assignment 1

Shayan Shabanzadeh

## Experiment Design

We measured how parallelism affects both (i) **runtime** and (ii) **accuracy** when approximating $\pi$ using three implementations:

- **Integration (atomic):** trapezoidal rule with a shared accumulator updated via *atomic* operations.
- **Integration (critical):** same, but the accumulator update is protected by a *critical* section.
- **Monte Carlo (MC):** throw random points and count hits inside the unit quarter-circle.

For each method we varied:

- **Threads:** $T \in \{1, 2, 4, 8, 16, 32, 64, 128\}$.
- **Work budget:** `steps` (integration) or `guesses` (MC) in $\{10^5, 10^6, 10^7, 10^8\}$.

For every configuration we computed:

$$\textbf{Runtime} = \text{mean wall time (s)}, \qquad \textbf{MAE} = |\hat{\pi} - \pi|.$$

Figures use actual thread counts on an x–axis with $\log_2$ scaling for readability; the data points are the real thread counts.

## Results: Runtime (What We Saw)

**MC scales; integration does not.** The *single* most important observation is that MC speedups are substantial with more threads, while both integration variants slow down due to per–iteration sharing.

**Monte Carlo (MC).** For the largest budget (`guesses` $= 10^8$), mean time falls from **1.3701 s** (1 thread) to **0.03823 s** (128 threads) — about **35.8× speedup**. Speedups are visible at all budgets:

- $10^5$: $0.001\,349\,\text{s} \rightarrow 0.001\,143\,\text{s}$ (1→128); small absolute gain because the job is tiny.
- $10^6$: $0.012\,968\,\text{s} \rightarrow 0.000\,328\,\text{s}$ (1→128); ∼**39.6**×.
- $10^7$: $0.134\,291\,\text{s} \rightarrow 0.017\,369\,\text{s}$ (1→128); ∼**7.7**×.
- $10^8$: $1.370\,114\,\text{s} \rightarrow 0.038\,228\,\text{s}$ (1→128); ∼**35.8**×.

*Why:* MC trials are independent (no shared updates), so threads progress without blocking.

**Integration (atomic).** Here, every step updates a shared accumulator via an atomic. Contention grows with $T$; stalls dominate for large budgets. For `steps` $= 10^8$, runtime *increases* from **0.6857 s** (1 thread) to **13.5628 s** (128 threads): a **19.8× slowdown**. Similar trend at other budgets:

- $10^5$: $0.000\,784\,\text{s}$ (1) $\rightarrow$ $0.021\,986\,\text{s}$ (128) *(overhead dominates when work is tiny)*.
- $10^6$: $0.006\,928\,\text{s}$ (1) $\rightarrow 0.132\,540\,\text{s}$ (128) ↑.
- $10^7$: $0.068\,570\,\text{s}$ (1) $\rightarrow 1.344\,519\,\text{s}$ (128) ↑.
- $10^8$: $0.685\,736\,\text{s}$ (1) $\rightarrow 13.562\,825\,\text{s}$ (128) ↑.

**Integration (critical).** The critical section forces *full* serialization at the update, so it behaves even worse than atomics. For `steps` $= 10^8$, time grows from **1.0561 s** (1) to **39.8585 s** (128): a **37.8× slowdown**. Other budgets show the same pattern:

- $10^5$: $0.001\,126\,\text{s}$ (1) $\rightarrow 0.047\,521\,\text{s}$ (128).
- $10^6$: $0.010\,311\,\text{s}$ (1) $\rightarrow 0.434\,496\,\text{s}$ (128).
- $10^7$: $0.104\,327\,\text{s}$ (1) $\rightarrow 4.099\,531\,\text{s}$ (128).
- $10^8$: $1.056\,109\,\text{s}$ (1) $\rightarrow 39.858\,507\,\text{s}$ (128).

**Summary of runtime.** MC: *parallel speedups* (best at large budgets). Integration with per–iteration sharing: *parallel slowdowns* that worsen with $T$ and budget. The runtime plot (left) visually summarizes these numeric trends.

## Results: Accuracy (What We Saw)

**Accuracy is governed by the work budget, not the thread count.** Increasing `steps`/`guesses` from $10^5$ to $10^8$ consistently reduces MAE by 1–2 orders of magnitude. For a *fixed* budget, errors fluctuate slightly with $T$ but show *no systematic dependence* on threads.

**Monte Carlo (MC).** At 1 thread, MAE improves from **1.678e-3** (1e5) to **2.033e-5** (1e8), matching the familiar $O(1/\sqrt{N})$ behavior. Holding budget fixed at 1e8, MAE across threads stays in the same $O(10^{-5})$ band:

- 1e8 guesses: $2.033 \cdot 10^{-5}$ (1) vs. $3.927 \cdot 10^{-5}$ (128).

**Integration (atomic).** At 1 thread, MAE shrinks from **1.415e-3** (1e5) to **1.831e-5** (1e8). For a fixed budget of 1e8, MAE remains $O(10^{-5})$ across threads:

- 1e8 steps: $1.831 \cdot 10^{-5}$ (1) vs. $4.062 \cdot 10^{-5}$ (128).

**Integration (critical).** At 1 thread, MAE goes from **1.399e-3** (1e5) to **1.674e-5** (1e8). For 1e8 steps across threads, it stays $O(10^{-5})$:

- 1e8 steps: $1.674 \cdot 10^{-5}$ (1) vs. $5.308 \cdot 10^{-5}$ (128).

**Summary of accuracy.** More samples $\Rightarrow$ better accuracy. Threads simply change how *fast* we can reach a given budget (and for integration with sharing, they actually make it slower); they do not directly change the error for a fixed budget. The error plot (right) shows essentially flat curves vs. thread count for each budget.

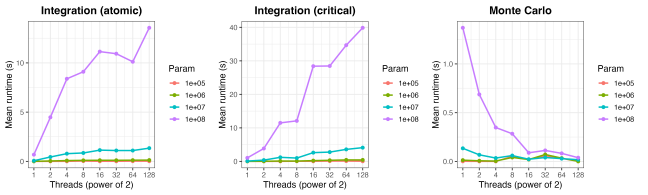


Figure 1: Runtime vs. threads ($\log_2$ ticks). MC accelerates with threads (e.g., 1e8 guesses: 1.37 s $\rightarrow$ 0.038 s); integration with atomics/critical slows down strongly as contention rises (e.g., 1e8 steps atomic: 0.686 s $\rightarrow$ 13.56 s; critical: 1.06 s $\rightarrow$ 39.86 s).

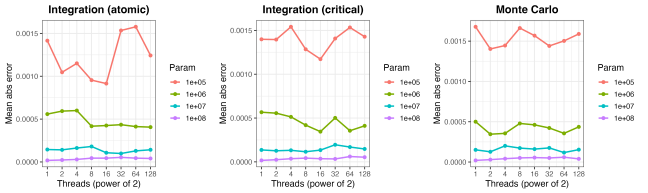


Figure 2: Mean absolute error (MAE) vs. threads. Curves are essentially flat for a fixed budget, confirming error depends on total samples, not thread count (e.g., MC 1e8: $2.0\times10^{-5}$–$3.9\times10^{-5}$ across $T$).

## Why These Results Happen (Mechanistic Explanation)

*MC is embarrassingly parallel:* each trial is independent; no shared state $\Rightarrow$ no contention. *Integration with per–iteration sharing:* every step touches a single shared accumulator. With **atomics**, the cache line and RMW pipeline serialize many updates; with a **critical** section, only one thread can update at a time, producing the strongest slowdown. Hence:

MC: scale with $T$ vs. Integration: stall with $T$.

Accuracy follows the mathematics of the estimator (MC variance $\propto 1/N$; trapezoidal error decreases with finer discretization). Threads do not change that estimator, only the throughput of samples.