

Parallel Monte Carlo methods for numerical integration using MPI

When does parallel execution actually help?

Carl Etornam Yevutsey · Evans Boateng · Ernest Osei Amofo

Practical Course on High-Performance Computing

Georg-August-Universität Göttingen

Problem

More samples improve precision.
Can MPI make them affordable?

Our integral

$$I_d = \int_{[-1,1]^d} \sum x_j^2 dx$$

Exact answer: $I_d = 2^d d / 3$

Monte Carlo estimator

$$\hat{I}_N = (2^d / N) \sum f(X_k)$$

X_k are uniform samples in $[-1,1]^d$

Ten times less uncertainty needs
about 100 times more samples

Why MPI?

Each sample can be evaluated independently.

Partial sums can be combined at the end.

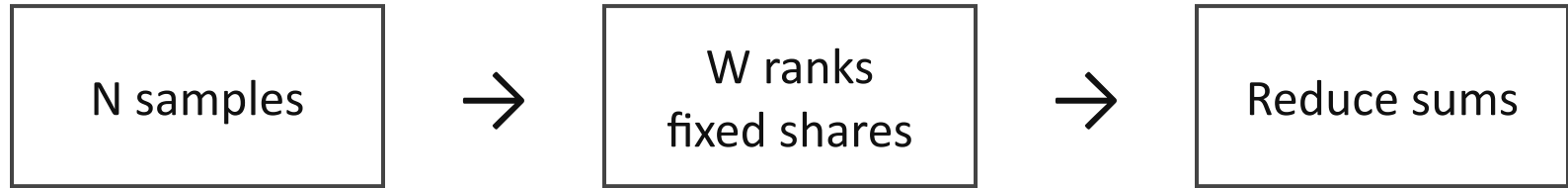
Implementations

True serial baseline



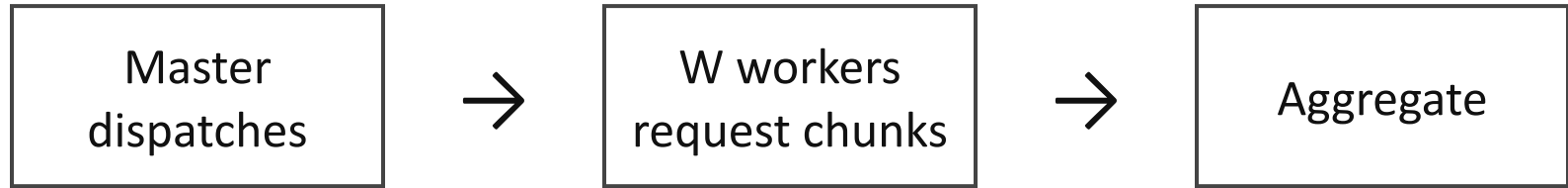
One compute thread; full work N

Static MPI



Exactly N samples across W computing ranks

Dynamic MPI



W workers plus one extra master rank

Random streams

We identify each stream by seed and worker.

SeedSequence and Philox make runs reproducible.

Short-prefix checks test duplication and rerun agreement.

Experiments

Measured environment

NovaCore

AMD Ryzen 5 7235HS
4 physical cores

DELL

Intel Core i7-12700H
14 physical cores

Two unlike Windows hosts connected over Wi-Fi

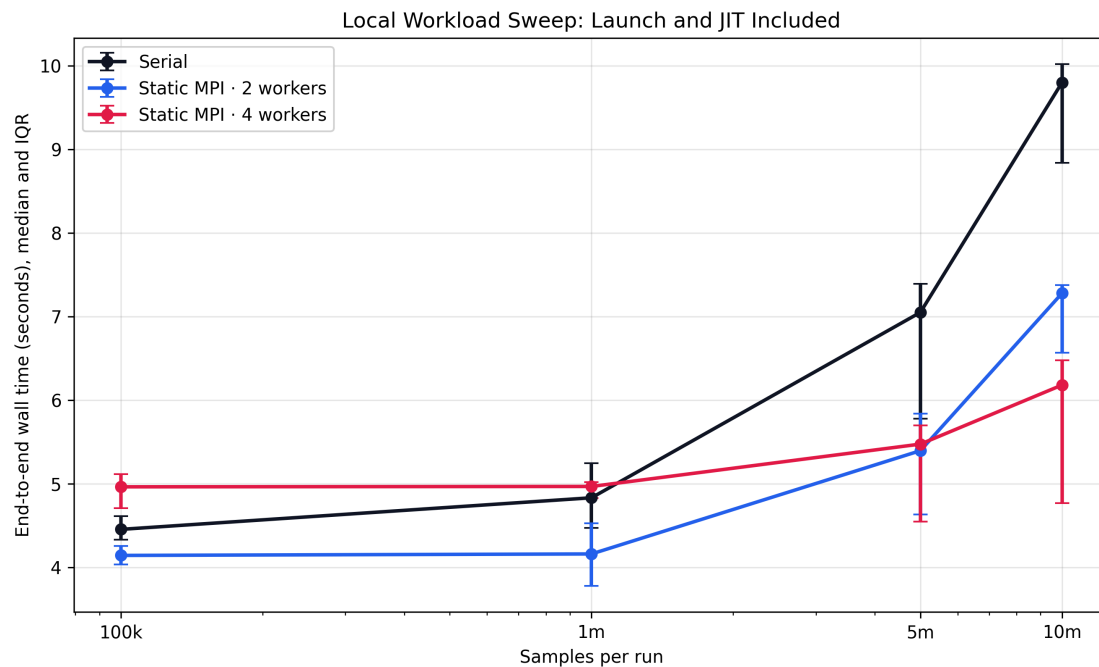
Stage 03: workload calibration

Seven repeated serial and MPI measurements.

Target: a 20–60 second serial workload.

Measure the crossover; do not assume it.

Stage 03: observed runtime



Seven runs per point · source: lower_sweep_20260926T084550Z/runs.csv · no exact crossover inferred

At 60 million samples

Serial: **22.511 s**

Static, 2 workers: **33.806 s**

Static, 4 workers: **13.622 s**

Adding workers helped at four,
but not at two

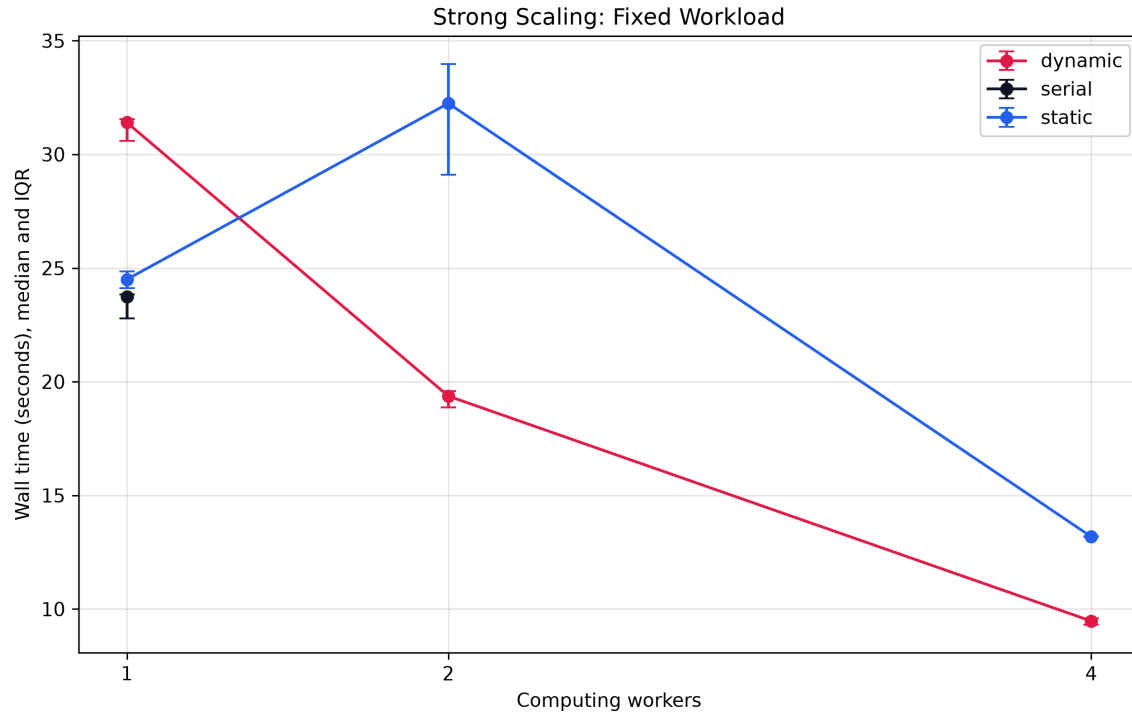
Stage 04: production design

26 configurations, seven repeats each.

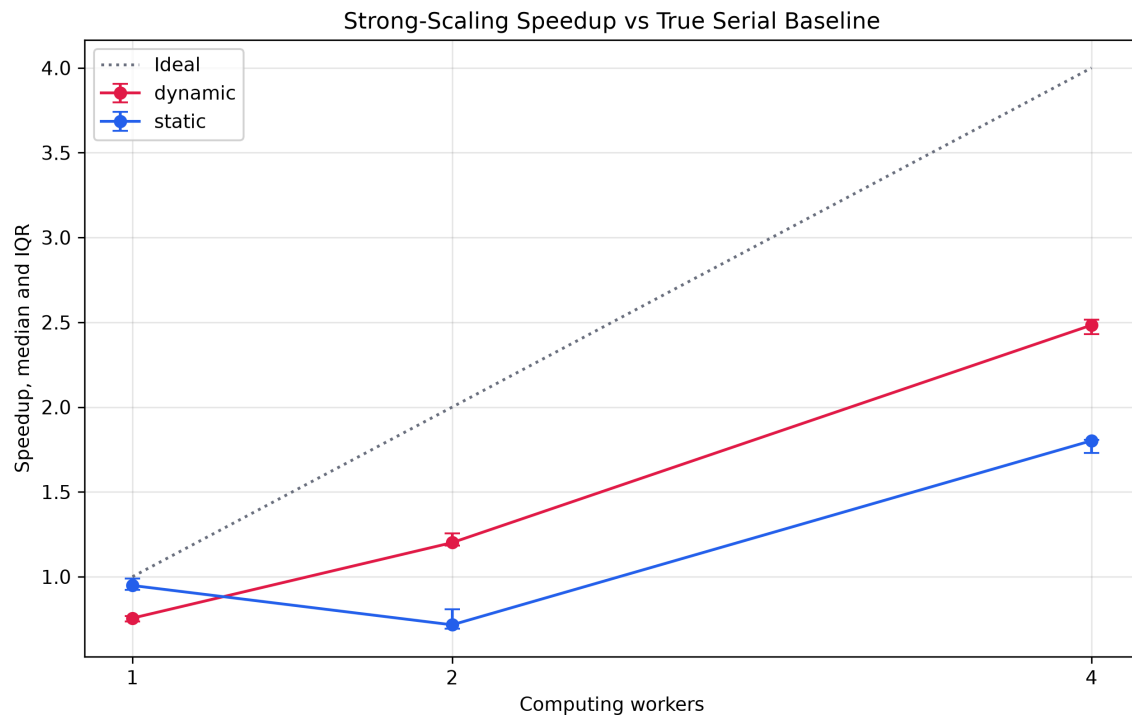
Fixed-work strong scaling and work-per-worker weak scaling.

One compute thread per MPI rank.

Strong-scaling runtime



Speedup versus true serial



Four-worker speedup

Static MPI

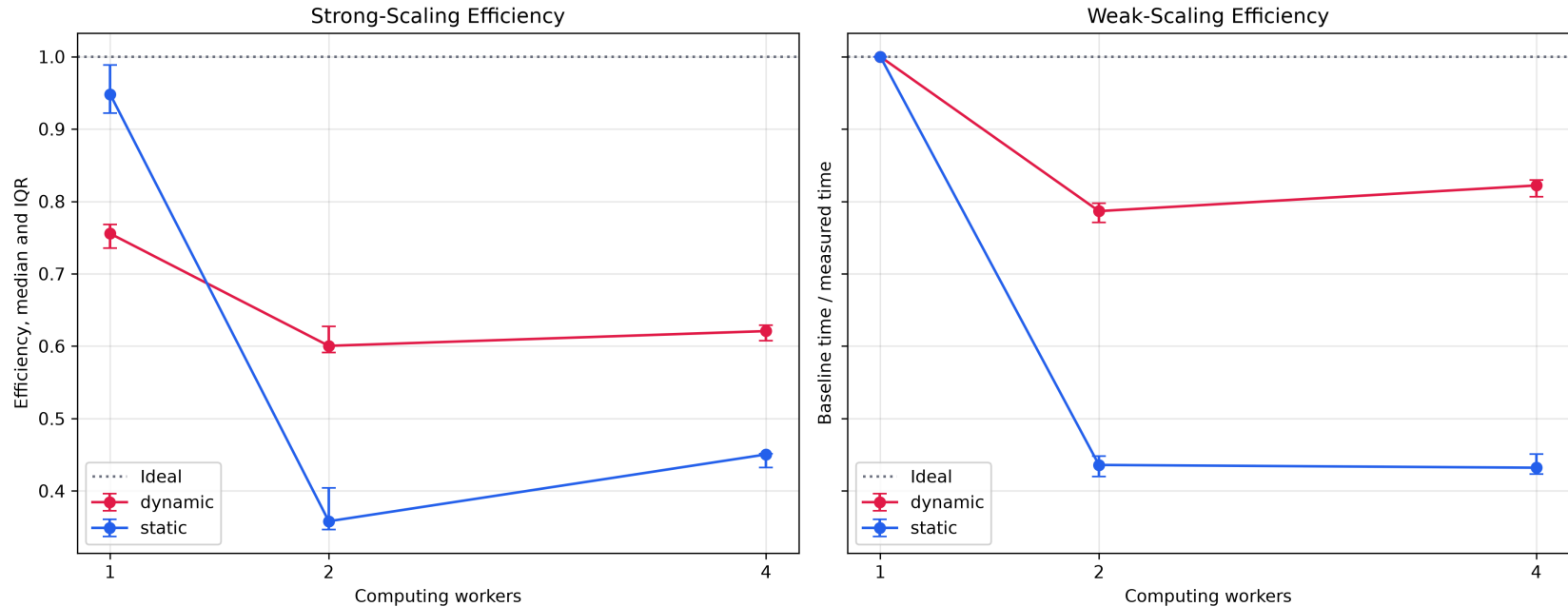
1.802×

Dynamic MPI

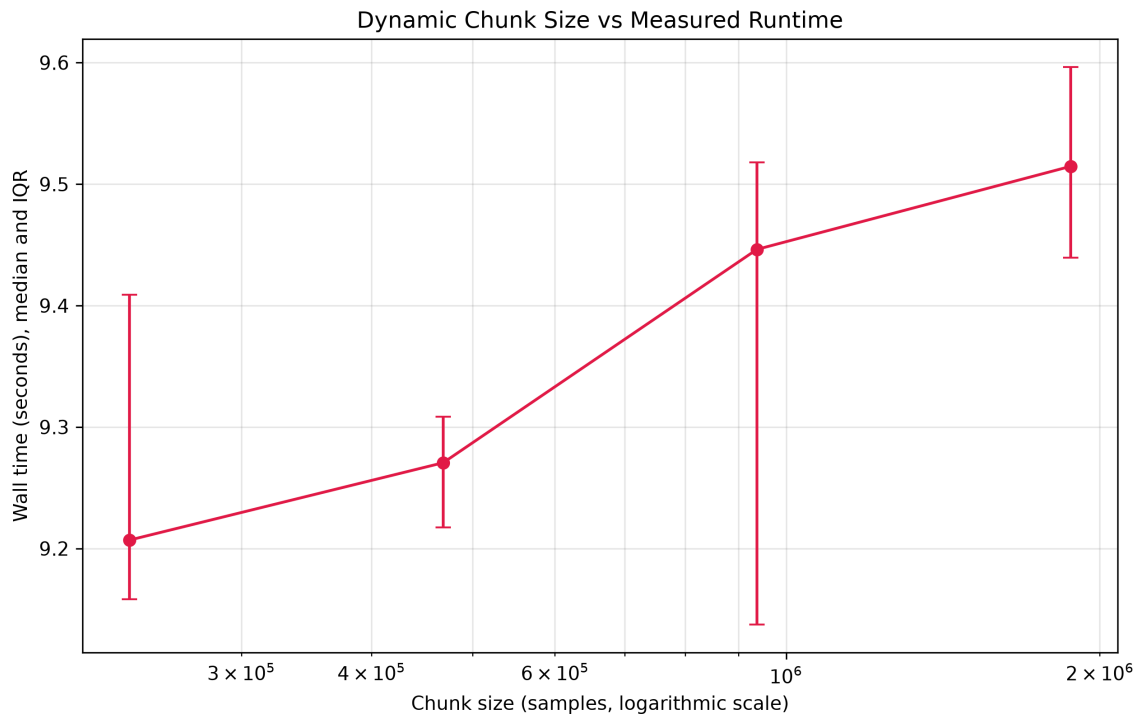
2.483×

Dynamic uses a fifth rank for its master

Efficiency stays below ideal



Chunk size matters



Scheduling stress

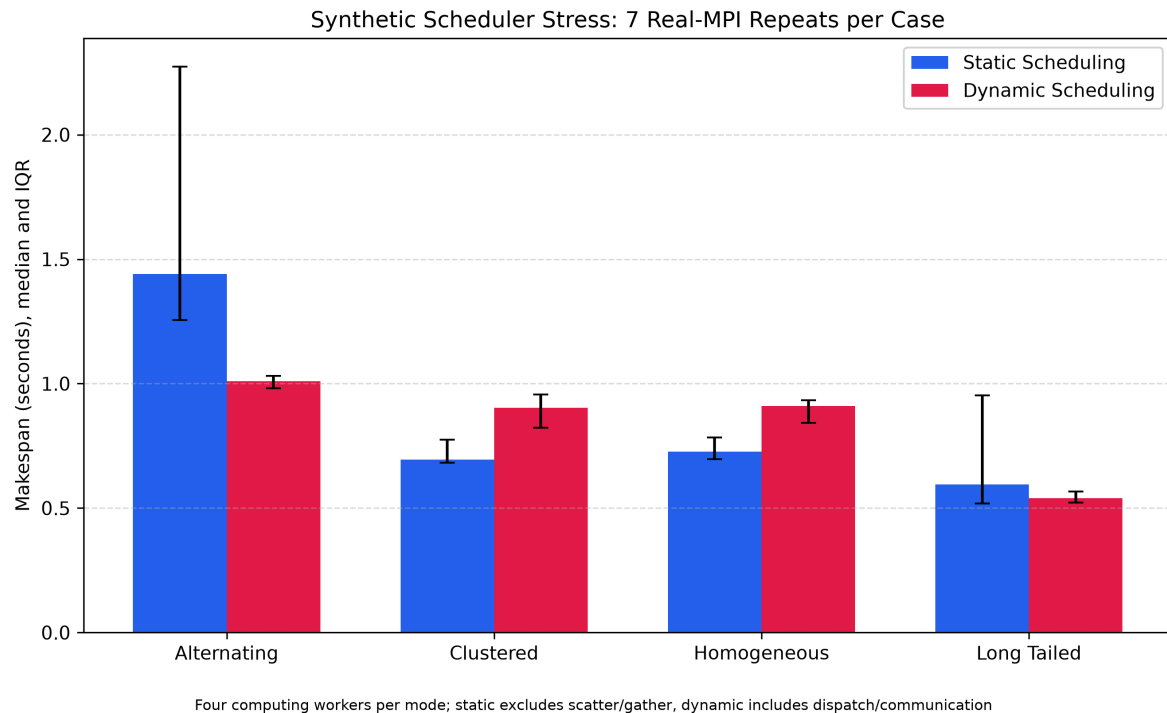
Stage 05: uneven work

Four controlled task-cost profiles.

Static assignment versus dynamic dispatch.

56 real-MPI runs in total.

Scheduling makespan



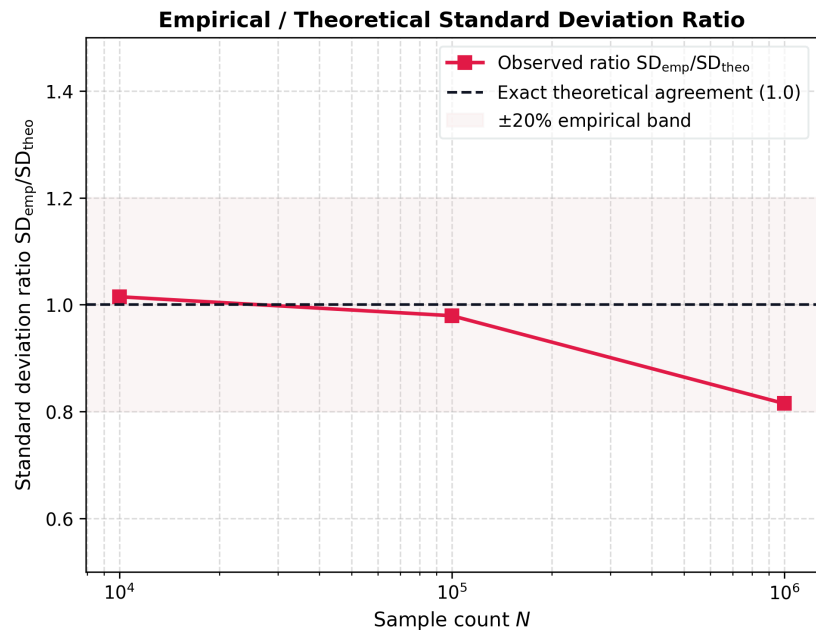
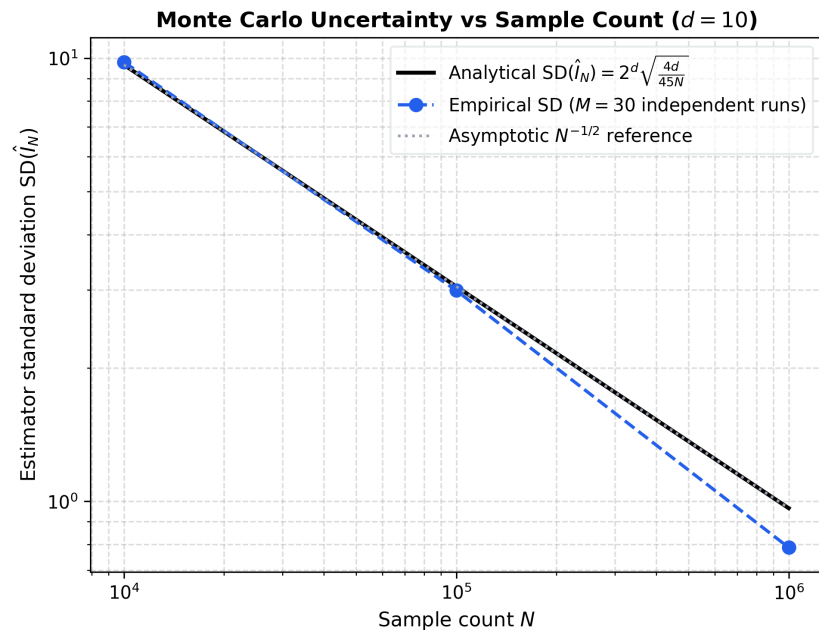
What changed with workload shape?

Dynamic was faster for alternating and long-tailed costs.

Static was faster for homogeneous and clustered costs.

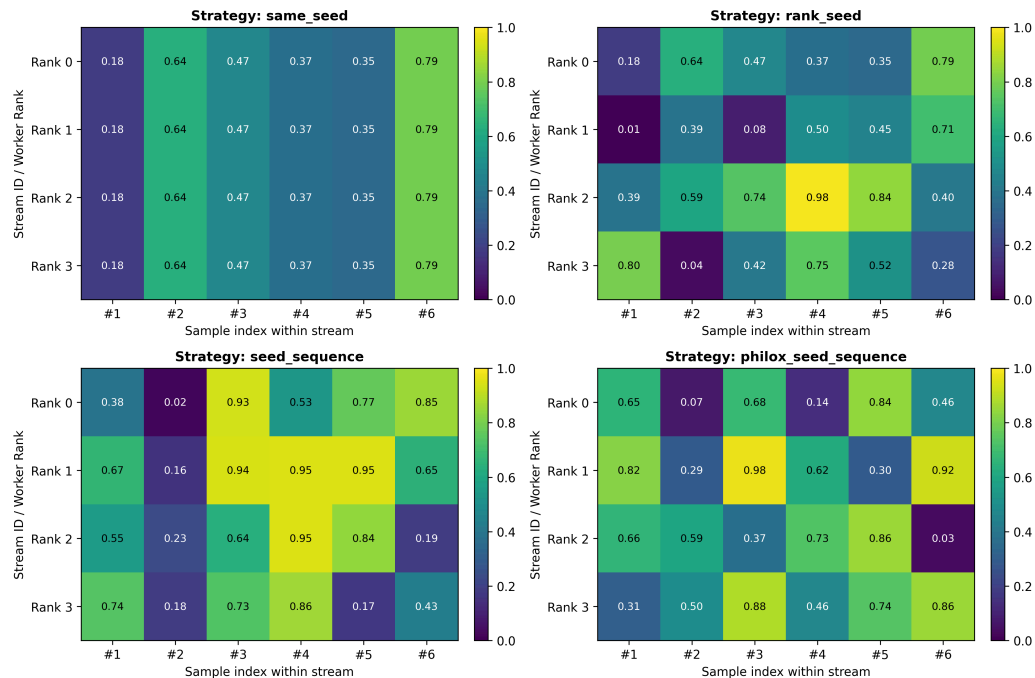
Statistical validation

Uncertainty falls with N



Finite RNG checks

Parallel Random Stream Prefix Comparison (Base Seed = 2026)
Notice identical rows in 'same_seed' (duplicated stream failure) vs distinct rows in others



Limits of this evidence

One smooth integrand and at most four computing workers.

Two Wi-Fi-connected, unreserved hosts; no CPU pinning.

Stage 06 was a separate local campaign.

Conclusion

MPI helped only after enough work amortized its overhead.

At four workers, dynamic scheduling was fastest for this setup.

The scheduler choice depends on task costs and rank allocation.

Key lessons

Use a true serial baseline and repeated wall-clock runs.

Count the extra master rank in dynamic mode.

Keep raw data, environment proof, and source hashes together.

Thank you!

Questions?