Homework 1 Report

KAI HOGAN

This report investigates the performance and accuracy of parallel methods for estimating using OpenMP. Two approaches were implemented and evaluated: a deterministic numerical integration method and a stochastic Monte Carlo method. Both were compared against a serial baseline to assess accuracy and scalability across varying thread counts. Initial parallel implementations employed critical sections for shared variable updates, which resulted in poor performance. A separate implementation replaced critical regions with atomic operations, yielding measurable improvements at low to moderate thread counts. However, as the number of threads increased beyond eight, performance did not scale due to thread contention and poor scaling. The deterministic integration method consistently produced more accurate results than the Monte Carlo approximation, while the Monte Carlo method executed more quickly. Overall, the study highlights the trade-offs between synchronization mechanisms in OpenMP and the need for reduction-based strategies to achieve efficient parallel performance in embarrassingly parallel numerical computations.

CCS Concepts: • **Computing methodologies → Parallel algorithms**; *Shared memory algorithms*; • **Theory of computation** → *Design and analysis of algorithms.*

ACM Reference Format:
Kai Hogan. 2025. Homework 1 Report. *ACM Trans. Graph.* 1, 1, Article 1 (October 2025), 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Findings

1.1 Serial vs Parallel (Deterministic integration):

For the deterministic integration method, the serial version performed much better than the parallel implementation when using 8 threads with 100 and 1,000 steps. With 10,000 steps, the parallel version showed a slight speedup, though the improvement was not significant. At 100,000 steps, however, the parallel code ran about four times faster than the serial version. For all higher thread counts, the serial implementation consistently outperformed the parallel one at every step size. Still, the performance gap between the two versions became smaller as the number of steps increased.

1.2 Parallel(Deterministic integration) vs Parallel(Monte Carlo)

For smaller step sizes, the Monte Carlo method consistently ran faster than the deterministic integration method. However, as the step size increased, the difference in runtime between the two approaches began to shrink. With 16 and 24 threads at 100,000 steps, both methods produced nearly identical execution times. Despite its speed advantage, the Monte Carlo method was far less accurate than deterministic integration. With only 100 guesses, its estimate of pi could be off by as much as 0.2. Even at higher numbers of guesses, the error remained around 0.05. In contrast, the deterministic integration method was never off by more than 0.01 and became increasingly accurate as the step size grew.

Threads	Steps	Serial Time (s)	Parallel Time (s)	Speedup	Efficiency
8	100000	0.00407	0.00121	3.37×	42%
16	100000	0.00418	0.02395	0.17×	1%
24	100000	0.00415	0.02052	0.20×	0.8%
28	100000	0.00441	0.03326	0.13×	0.5%

Fig. 1. Execution times Deterministic integration (Parallel vs Serial)

Threads	Steps	Integration π	Error (Integration)	Monte Carlo π	Error (Monte Carlo)
8	100	3.1419368579	0.0003442044	3.3200000000	0.1784073465
	1,000	3.1416035449	0.0000108914	3.1680000000	0.0264073465
	10,000	3.1415929980	0.0000003445	3.1268000000	0.0147926535
	100,000	3.1415926645	0.0000000110	3.1409600000	0.0006326535
16	100	3.1419368579	0.0003442044	3.0800000000	0.0615926535
	1,000	3.1416035449	0.0000108914	3.1400000000	0.0015926535
	10,000	3.1415929980	0.0000003445	3.1180000000	0.0235926535
	100,000	3.1415926645	0.0000000110	3.1366000000	0.0049926535
24	100	3.1419368579	0.0003442044	3.0400000000	0.1015926535
	1,000	3.1416035449	0.0000108914	3.0560000000	0.0855926535
	10,000	3.1415929980	0.0000003445	3.1220000000	0.0195926535
	100,000	3.1415926645	0.0000000110	3.1349600000	0.0066326535
28	100	3.1419368579	0.0003442044	3.0400000000	0.1015926535
	1,000	3.1416035449	0.0000108914	3.0400000000	0.1015926535
	10,000	3.1415929980	0.0000003445	3.1112000000	0.0303926535
	100,000	3.1415926645	0.0000000110	3.1347200000	0.0068726535

Fig. 2. Monte Carlo accuracy vs Deterministic integration accuracy

1.3 Critical Section vs Atomic

With only 8 threads, the atomic implementation only slightly outperformed the critical section version. As the number of threads increased, the performance gap widened, with the atomic version running significantly faster. Both approaches introduced serialization, but the atomic operation only serialized updates to the pi variable, whereas the critical section serialized access to an entire region of code.

Author's Contact Information: Kai Hogan, khogan@uoregon.edu.

ACM 1557-7368/2025/10-ART1
https://doi.org/XXXXXXX.XXXXXXX

1.4 Conclusion

The experiments demonstrate clear trade-offs between parallelization strategies and numerical methods for estimating pi. Deterministic integration consistently produced highly accurate results, while the Monte Carlo method offered faster execution at the cost of lower precision. Overall, achieving both high accuracy and efficient parallel performance requires careful selection of synchronization mechanisms and workload size.

Threads	**Step Count**	**Parallel Time (Atomic)**	**Parallel Time (Critical)**
8	100–100000	0.0004–0.001s	0.0005–0.001s
16	100–100000	0.0007–0.0009s	0.007 - 0.025s
24	100–100000	~0.013–0.02s	0.020 - 0.022s
28	100–100000	~0.02–0.03s	0.020 - 0.040s

Fig. 3. Execution times (Critical Section vs Atomic Operation)