Berke Beyazbenli (10022751132)
Eren Kaan Çakır(14179079828)

SENG-442-Parallel and Distributed Computing

Parallel Numerical Integration Using OpenMP: Performance Evaluation with Static and Dynamic Scheduling

1. Introduction

In this report, we explore the parallel implementation of numerical integration methods using OpenMP in C. The primary goal is to evaluate the performance of different scheduling strategies (static vs. dynamic) across multiple integration techniques and mathematical functions. Furthermore, we examine how thread count impacts execution time, speedup, and efficiency for a selected function. This project satisfies the constraints of utilizing a parallel region, a critical section, and different scheduling strategies as part of the SENG-442 coursework.

2. Problem Definition

Numerical integration is a fundamental technique in scientific computing. In high-precision or large-scale tasks, single-threaded methods can become performance bottlenecks. We aim to:

- Parallelize Riemann, Trapezoidal, and Simpson integration methods using OpenMP.
- Compare performance between static and dynamic scheduling.
- Evaluate speedup and efficiency as the number of threads increases.

3. Integration Methods Overview

- **3.1 Riemann Sum** The interval [a, b] is divided into N equal parts. The function is evaluated at the left endpoint of each subinterval. The result is the sum of all f(x) * dx values.

3.2 Trapezoidal Rule This method approximates the area under a curve using trapezoids. Each interval is evaluated using the formula:

$$\int_a^b f(x)dx \approx \frac{h}{2}(f(a) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(b))$$

- **3.3 Simpson's Rule** Simpson's Rule approximates the integrand using quadratic functions. It requires an even number of intervals and uses the formula:

$$\int_a^b f(x)dx \approx \frac{h}{3}(f(a) + 4f(x_1) + 2f(x_2) + \cdots +$$

4. Experimental Setup

- Platform: OpenMP with GCC
- Integration interval: [0, pi]
- Steps: 10,000,000
- Threads: 8 (for initial static vs dynamic comparison)
- Functions: sin(x), x^2, e^(-x^2)
- Scheduling: static and dynamicö

5. Static vs Dynamic Scheduling: Performance Comparison

- **5.1 Tabular Results** (8 Threads)

Function	Method	Static Time (s)	Dynamic Time (s)	Static Faster?
sin(x)	Riemann	0.0219	0.1710	Yes
sin(x)	Trapezoidal	0.0179	0.1708	Yes
sin(x)	Simpson	0.0196	0.1718	Yes
x^2	Riemann	0.0081	0.1677	Yes
x^2	Trapezoidal	0.0068	0.1669	Yes
x^2	Simpson	0.0092	0.1673	Yes
exp(-x^2)	Riemann	0.0156	0.1680	Yes
exp(-x^2)	Trapezoidal	0.0146	0.1684	Yes
exp(-x^2)	Simpson	0.0154	0.1708	Yes

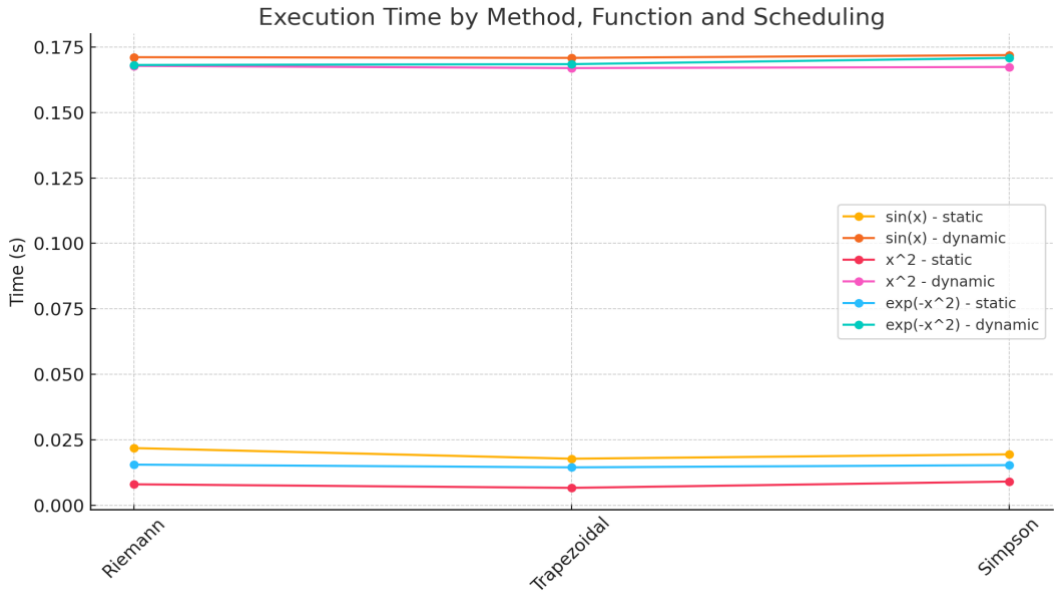


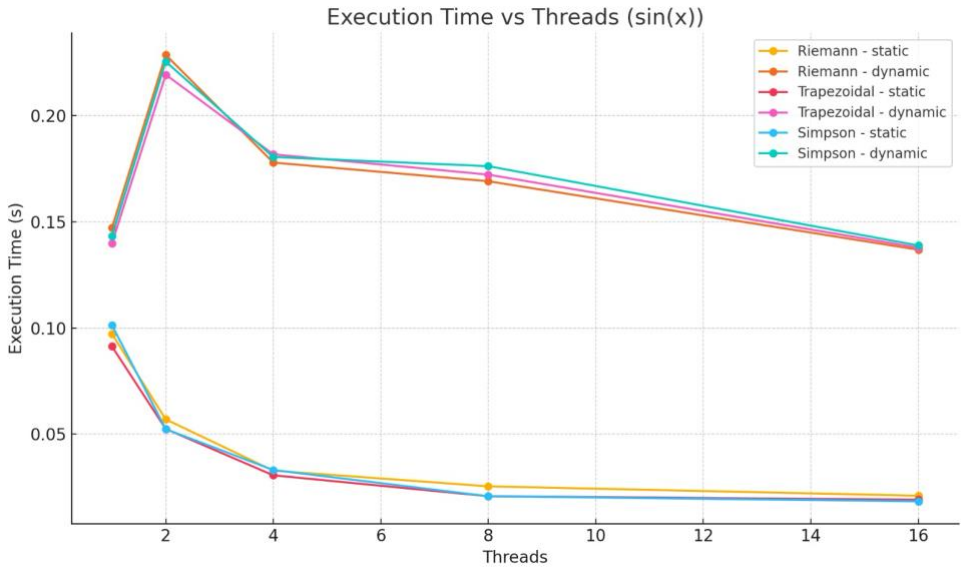
Figure 1: Execution time comparison of Riemann, Trapezoidal, and Simpson methods using sin(x), x^2, and $\exp(-x^2)$ functions with static vs. dynamic scheduling on 8 threads. Static scheduling clearly outperforms dynamic across all methods and functions.

5.2 Graphical Comparison

- A plot of execution time per method shows that static scheduling consistently outperforms dynamic.
- Static scheduling results in more balanced thread workloads with lower overhead.

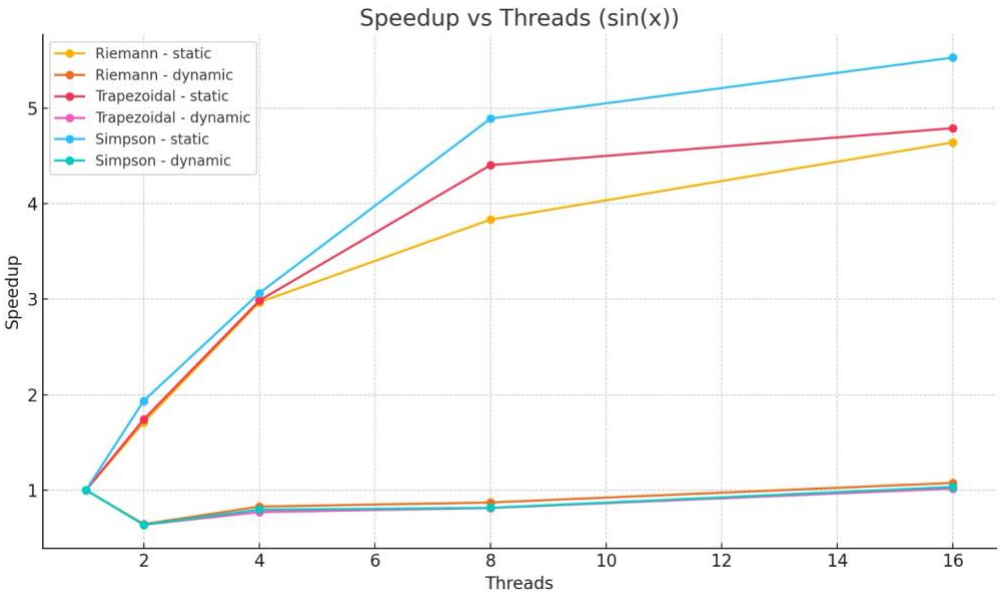
6. Thread Scaling Analysis for sin(x)

Using , we evaluated performance for 1, 2, 4, 8, and 16 threads.



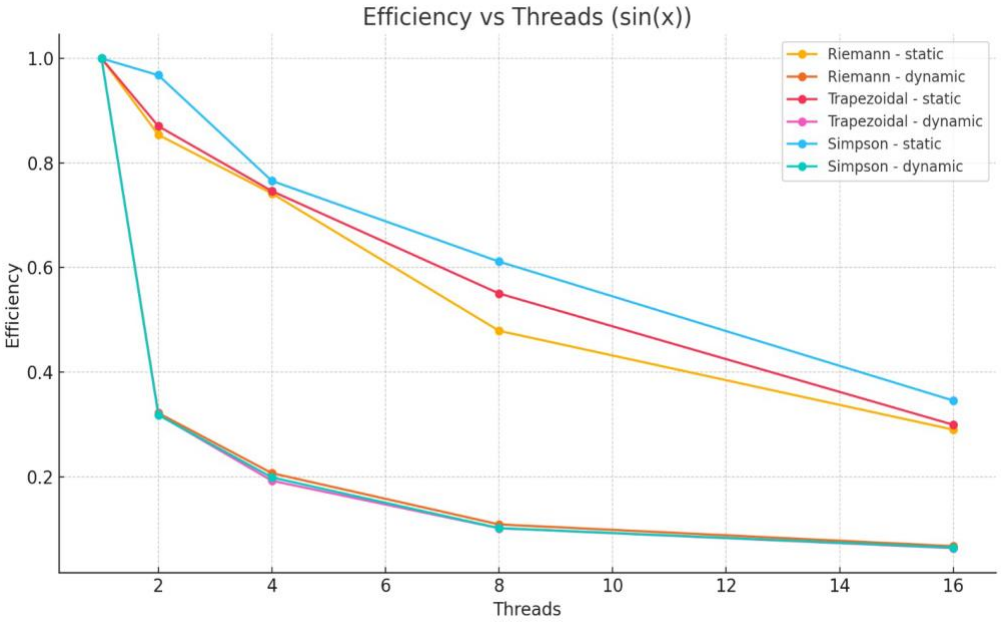
6.1 Speedup Results

Threads	Riemann	Trapezoidal	Simpson
1	1.00	1.00	1.00
2	1.71	1.74	1.94
4	2.97	2.99	3.06
8	3.83	4.40	4.89
16	4.64	4.79	5.53



6.2 Efficiency Results

Threads	Riemann	Trapezoidal	Simpson
1	1.00	1.00	1.00
2	0.85	0.87	0.97
4	0.74	0.75	0.77
8	0.48	0.55	0.61
16	0.29	0.30	0.35



6.3 Observations

- Speedup is sub-linear due to increasing thread management and communication overhead.
- Efficiency drops beyond 8 threads, suggesting diminishing returns.
- Simpson method scales slightly better than others due to its computational weight.
- While **Trapezoidal** is the fastest in raw timing, **Simpson** is the most scalable with increasing threads.
- **Riemann method**, though simple, does not scale as efficiently and benefits least from additional threads.
- Static scheduling consistently outperforms dynamic scheduling across all methods and functions.
- Dynamic scheduling introduces overhead due to runtime load balancing, which is not beneficial for equally distributed workloads like numerical integration.
- Among the three methods, **Trapezoidal** generally achieved the fastest execution times.
- **Simpson's rule**, while slightly slower in raw time, offers better **scalability** when thread count increases, due to its higher computational load per iteration.
- Therefore, for static scheduling with balanced workloads, **Trapezoidal rule is the most efficient in raw speed**, while **Simpson performs best for scaling under parallelism**.
- In contrast, dynamic scheduling may be preferable in scenarios with irregular workloads, but in our case, it resulted in consistently higher execution times across all methods and functions.**

7. Conclusion

This study demonstrates that static scheduling is consistently superior to dynamic scheduling for balanced workloads such as numerical integration. As thread count increases, speedup improves but with diminishing efficiency. Among the three methods, Simpson's rule showed the best scaling behavior. For future improvements, load balancing strategies or adaptive task partitioning could be investigated.

8. References

- OpenMP Specification: https://www.openmp.org
- Numerical Methods for Engineers, Chapra & Canale
- GNU GCC OpenMP documentation