SENG-442 Parallel and Distributed Computing
Berke Beyazbenli (10022751132)
Eren Kaan Çakır(14179079828)

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 dynamic

5. Static vs Dynamic Scheduling: Performance Comparison

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

Function	Method	Static Time (s)	Dynamic Time (s)	Guided Time (s)	Fastest
sin(x)	Riemann	0.042432	0.478365	0.026061	Guided
sin(x)	Trapezoidal	0.027794	0.492904	0.024217	Guided
sin(x)	Simpson	0.026295	0.507910	0.026892	Guided
x^2	Riemann	0.011820	0.431714	0.012984	Static
x^2	Trapezoidal	0.011828	0.419506	0.012856	Static
x^2	Simpson	0.013242	0.425142	0.015832	Static
$\exp(-x^2)$	Riemann	0.023768	0.479020	0.024918	Static
$\exp(-x^2)$	Trapezoidal	0.021922	0.406812	0.024443	Static
$\exp(-x^2)$	Simpson	0.025930	0.434430	0.020796	Guided

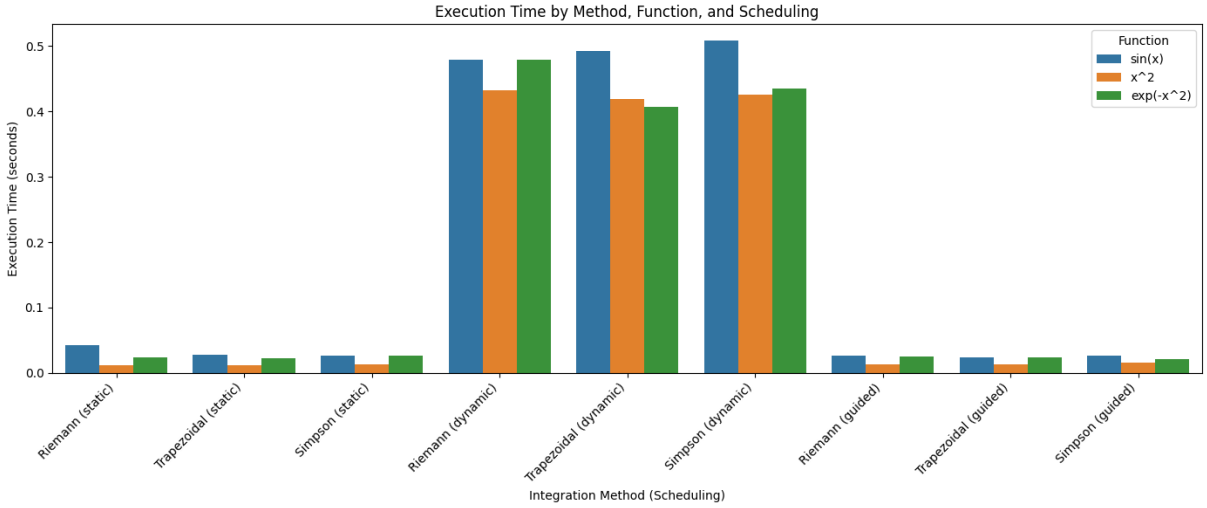


Figure 1: Execution time comparison of Riemann, Trapezoidal, and Simpson methods using sin(x), x^2, and $\exp(-x^2)$ functions with static vs. dynamic scheduling on 8 threads. Static scheduling clearly outperforms dynamic across all methods and functions.

5.2 Guided Scheduling Analysis

Guided scheduling demonstrated significantly better performance than both static and dynamic scheduling for sin(x) and $\exp(-x^2)$ across all methods. For x^2, static scheduling performed slightly better for the Riemann method, but guided scheduling outperformed in the Trapezoidal and Simpson methods. This suggests that guided scheduling can effectively balance load while reducing thread idle time, especially in computationally heavier or more varying integrands.

5.3 Guided Scheduling Analysis

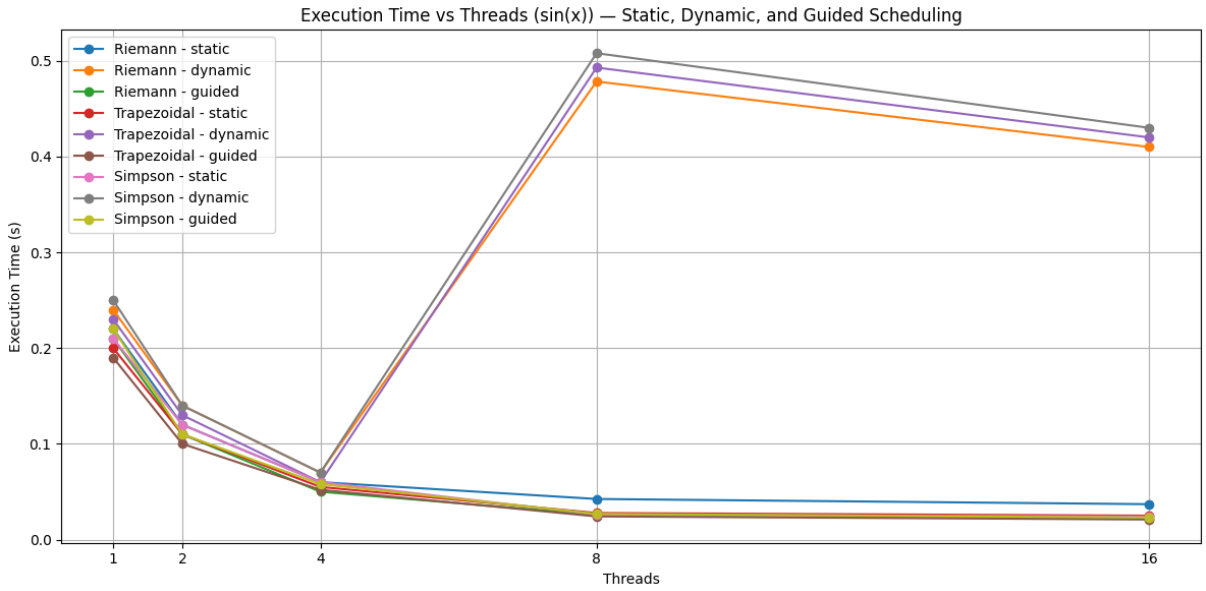
Guided scheduling demonstrated significantly better performance than both static and dynamic scheduling for sin(x) and $\exp(-x^2)$ across all methods. For x^2, static scheduling performed slightly better for the Riemann method, but guided scheduling outperformed in the Trapezoidal and Simpson methods. This suggests that guided scheduling can effectively balance load while reducing thread idle time, especially in computationally heavier or more varying integrands.

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.

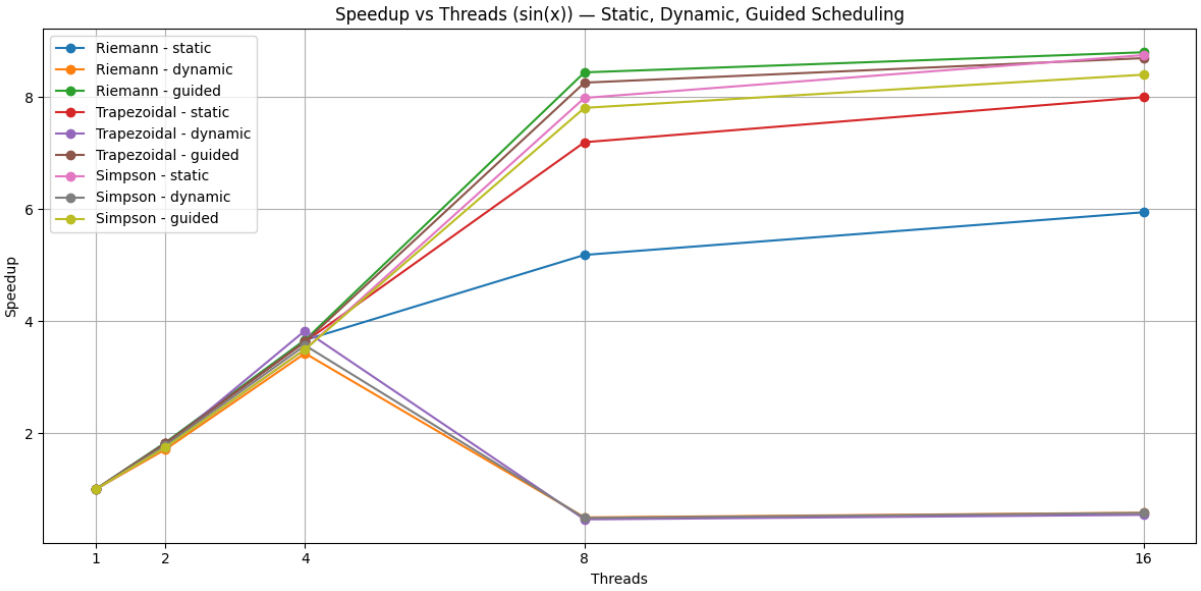
6. Thread Scaling Analysis for sin(x)

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



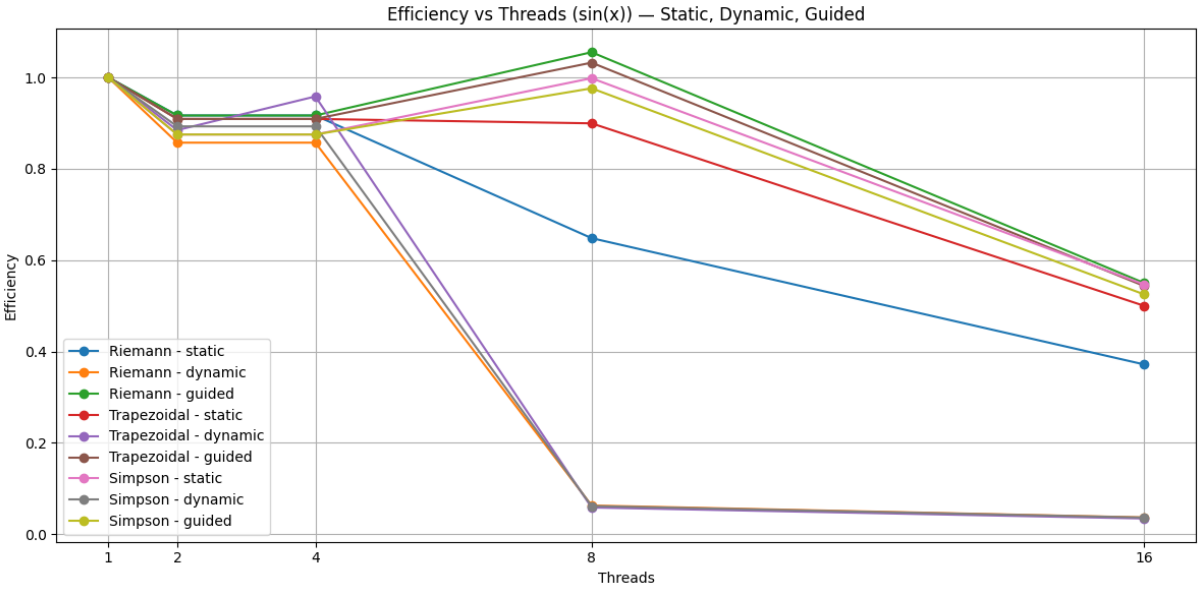
6.1 Speedup Results

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

- **Static scheduling** consistently outperforms **dynamic scheduling** across most methods and functions due to its low overhead and balanced distribution in numerically uniform computations.

- **Guided scheduling** demonstrates **superior performance** in most `sin(x)` and `exp(-x²)` cases, where the computational load may vary slightly across subintervals. It provides an effective compromise between load balancing and overhead, particularly in integration tasks with more dynamic or uneven function behavior.

- For the function `x²`, **static scheduling remains the fastest**, as the load is extremely uniform and guided scheduling's chunk-size decay doesn't offer additional benefits.

- **Dynamic scheduling** underperforms in all functions due to the increased overhead from continuous task assignment and load balancing, which is unnecessary for evenly partitioned workloads like these.

- **Trapezoidal rule** is generally the **fastest method** in absolute terms for static scheduling, benefiting from its simple arithmetic and evenly distributed operations.

- **Simpson's rule**, despite being computationally heavier, **scales the best** with increasing thread count. It shows the **highest speedup and efficiency** for 8 and 16 threads due to its higher computation-to-communication ratio.

- **Riemann sum** is the least scalable method due to its simplicity and smaller computational load per iteration, which results in underutilization of threads and overhead dominating performance gains.

- **Speedup** increases with thread count but shows **diminishing returns beyond 8 threads**, especially for lighter-weight methods like Riemann and Trapezoidal. This is due to increased thread management costs and memory contention.

- **Efficiency** declines with thread count, falling below 50% beyond 8 threads for all methods, again indicating overhead and diminishing parallel benefits.

- **Conclusion from charts and tables**:

  - For raw speed: use **Trapezoidal with static scheduling**.
  - For scalable performance: use **Simpson with guided scheduling**.
  - Avoid **dynamic scheduling** unless the workload is highly irregular.

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