Performance Analysis of Naive vs. Karatsuba Multiplication

This document presents the performance results of naive and Karatsuba polynomial multiplication methods, both in sequential and parallel forms. The performance metrics are evaluated based on varying input sizes and thread counts. The Karatsuba algorithm, which theoretically outperforms naive multiplication for large input sizes, demonstrates significant overhead in this implementation, impacting its practical performance in these tests.

Performance Results Summary

Array Size	Threads	Naive (Sequential)	Naive (Parallel)	Karatsuba (Sequential)	Karatsuba (Para
1,000	10	4 ms	17 ms	140 ms	31 ms
10,000	5	48 ms	29 ms	4,403 ms	2,216 ms
10,000	10	46 ms	27 ms	4,500 ms	2,294 ms
100,000	10	4,044 ms	525 ms	93,279 ms	15,768 ms

Analysis

The results indicate that the Karatsuba multiplication algorithm is not currently outperforming the naive multiplication approach, particularly for the input sizes tested. The Karatsuba algorithm, which is generally optimized for very large numbers, incurs a significant overhead in this implementation, both in sequential and parallel versions. This overhead includes additional computational steps in the recursive calls and increased memory management.

1. Overhead in Karatsuba Multiplication:

- The Karatsuba method involves multiple recursive steps, which introduce computational overhead that outweighs the benefits of reduced multiplication complexity at this scale. This is especially noticeable in the sequential Karatsuba version, where the recursion depth increases latency substantially.

2. Parallel Speedup Analysis:

- Parallelizing Karatsuba does provide some improvement over the sequential version, especially as the array size increases. However, the speedup observed is still limited compared to naive parallelization due to the higher overhead associated with Karatsuba's recursive divide-and-conquer approach.

3. Threshold Adjustment for Karatsuba:

- Testing with various threshold values for switching to direct multiplication revealed that, while larger thresholds reduce recursion depth, they ultimately make Karatsuba resemble the naive multiplication approach. At smaller array sizes, Karatsuba does not fully leverage its intended efficiency gains, which are generally more pronounced with larger numbers where standard multiplication becomes computationally intensive.

Conclusion

Based on these results, the naive multiplication algorithm is currently better suited for the tested input sizes and thread configurations. For applications involving very large integers or high-degree polynomial multiplication, Karatsuba's benefits may become more apparent. Further optimization could include reducing overhead in Karatsuba's recursive implementation or exploring hybrid approaches that adaptively switch between methods based on input size and available resources.