**Performance Comparison of Sequential and Parallel Fibonacci Computation using OpenMP**

**Introduction**

This report presents a performance analysis of Fibonacci sequence computation using both sequential and parallel approaches. The experiment is conducted using OpenMP to parallelize Fibonacci computation for many terms.

The purpose of this study is to:

* Analyze the difference in execution time between sequential and parallel executions.
* Apply various OpenMP techniques, such as static and dynamic scheduling, and proper variable handling.
* Observe the impact of different thread counts and chunk sizes on performance.

**Methodology**

**Code Implementation**

The experiment involves the following:

* Computing the Fibonacci sequence up to **N = 10,000,000** terms.
* Execution in both **sequential and parallel modes**.
* Parallelizing loop-based Fibonacci computation using OpenMP.
* Testing with different thread counts **(4, 6, 8, 10, 12)** and both **static and dynamic scheduling** strategies.

**OpenMP Features Used**

* #pragma omp parallel for schedule(static, chunk\_size) for optimized parallel execution.
* #pragma omp parallel for schedule(dynamic, chunk\_size) to test workload distribution.
* omp\_set\_num\_threads(num\_threads) to control the number of threads.
* omp\_get\_wtime() to measure execution time accurately.

**Results: Experimental Data**

**Execution Time for Sequential vs. Parallel Fibonacci Computation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run Number | Sequential Time (s) | Parallel Time (s) | Number of Threads | Scheduling Type |
| 1 | 0.079970 | 0.214807 | 4 | Static |
| 2 | 0.032254 | 0.062875 | 6 | Static |
| 3 | 0.033165 | 0.060533 | 8 | Static |
| 4 | 0.034544 | 0.058788 | 10 | Static |
| 5 | 0.033281 | 0.061115 | 12 | Static |
| 6 | 0.032936 | 0.199988 | 4 | Dynamic (Chunk Size: 6) |
| 7 | 0.034313 | 0.192392 | 6 | Dynamic (Chunk Size: 6) |
| 8 | 0.033371 | 0.228526 | 8 | Dynamic (Chunk Size: 6) |
| 9 | 0.034151 | 0.256167 | 10 | Dynamic (Chunk Size: 6) |
| 10 | 0.035378 | 0.228645 | 12 | Dynamic (Chunk Size: 6) |

**Comparison of Static and Dynamic Scheduling in Fibonacci Computation**

|  |  |  |
| --- | --- | --- |
| Number of Threads | Static - Parallel Time (s) | Dynamic - Parallel Time (s) (Chunk Size: 6) |
| 4 | 0.214807 | 0.199988 |
| 6 | 0.062875 | 0.192392 |
| 8 | 0.060533 | 0.228526 |
| 10 | 0.058788 | 0.256167 |
| 12 | 0.061115 | 0.228645 |

**Why Dynamic Scheduling is Not Suitable for Fibonacci Computation?**

Dynamic scheduling is **not suitable for Fibonacci computation** because it introduces:

* **High Overhead**: Frequent task allocation increases execution time instead of improving performance.
* **Data Dependencies**: Fibonacci calculations require the previous two terms, which dynamic scheduling does not handle efficiently.
* **Poor Cache Performance**: Non-contiguous memory access leads to excessive cache misses.
* **Synchronization Costs**: Threads must frequently synchronize, reducing parallel efficiency.

Static scheduling was preferred because it **ensures better workload distribution** and **minimizes synchronization overhead**, leading to improved performance.

**Analysis**

From the results:

* **Sequential execution performed better** than dynamic scheduling due to dependency issues.
* **Static scheduling provided better performance** in parallel execution by reducing overhead.
* **Dynamic scheduling introduced significant overhead**, making it inefficient for Fibonacci computation.

**Conclusion**

This study demonstrated that **OpenMP parallelization does not always improve performance** for dependent computations like Fibonacci. Static scheduling performed significantly better than dynamic scheduling due to **lower overhead and better cache utilization**. The results highlight the importance of selecting **appropriate scheduling strategies** based on the nature of the problem being parallelized.