**Performance Comparison of Sequential and Parallel Array Sum using OpenMP**

**1. Introduction**

This report presents a performance analysis of array summation using both sequential and parallel approaches. The experiment is conducted using OpenMP to parallelize the summation of a large array containing **500 million elements**.

The purpose of this study is to:

* Analyse the difference in execution time between sequential and parallel executions.
* Apply various OpenMP techniques, such as **static and dynamic scheduling**, and proper variable handling using **reduction**, **critical**, and **atomic** clauses.
* Observe the impact of different thread counts on performance.

**2. System Specifications**

The system used for testing is specified below:

|  |  |
| --- | --- |
| Specification | Details |
| Processor | Intel Core i7-8850H |
| Microarchitecture | Coffee Lake |
| Technology | 14 nm |
| Cores / Threads | 6 Cores / 12 Threads |
| Base Clock Speed | 2.60 GHz |
| Cache Memory | L1: 384 KB, L2: 1.5 MB, L3: 9 MB |

**3. Methodology**

**3.1 Code Implementation**

The experiment involves the following:

* An array of **500 million** integers initialized with incremental values.
* Calculation of the sum:
  + **Sequentially**.
  + **Parallelly** using OpenMP.
* Implementation of:
  + **Reduction** clause for efficient parallel sum.
  + **Critical** and **Atomic** sections to compare overhead.
* Testing with different thread counts (**2, 4, 8, 10, 12**) and both **Static** and **Dynamic** scheduling strategies with a chunk size of **100,000**.

**3.2 OpenMP Features Used**

* #pragma omp parallel for
* reduction(+:sum)
* #pragma omp critical
* #pragma omp atomic
* schedule(static) and schedule(dynamic, chunk\_size)

**4. Results**

**4.1 Experimental Data**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Run Number | Sequential Time (s) | Parallel Time (s) | Number of Threads | Scheduling Type (Static/Dynamic) |
| 1 | 1.16132 | 0.747892 | 2 | Static |
| 2 | 1.53184 | 0.975257 | 4 | Static |
| 3 | 1.47007 | 0.732904 | 8 | Static |
| 4 | 1.57922 | 0.342984 | 10 | Static |
| 5 | 1.40053 | 0.36392 | 12 | Static |
| 6 | 1.39619 | 0.841425 | 2 | Dynamic with ChunkSize of 100000 |
| 7 | 1.33642 | 0.463475 | 4 | Dynamic with ChunkSize of 100000 |
| 8 | 1.4769 | 0.409402 | 8 | Dynamic with ChunkSize of 100000 |
| 9 | 1.41341 | 0.367195 | 10 | Dynamic with ChunkSize of 100000 |
| 10 | 1.61726 | 0.610965 | 12 | Dynamic with ChunkSize of 100000 |
| Average | 1.438316 | 0.5855419 |  |  |

**4.2 Charts**

Line Graph

Bar Graph

**4.3 Analysis**

From the results:

* **Parallel execution** significantly reduced the execution time compared to sequential, especially with increased thread count.
* The **dynamic scheduling** helped balance workload and reduced idle time for threads.
* **Reduction clause** showed the best performance due to minimal synchronization overhead.
* **Critical** and **Atomic** sections introduced performance degradation due to thread synchronization delays.
* The ideal number of threads matched the hardware's physical and logical cores (**up to 12 threads** in this case).

**5. Discussion**

**Why use Reduction?**

* Reduction is highly optimized in OpenMP.
* Automatically manages local sums for each thread and combines them efficiently at the end.
* Minimizes locking and synchronization, making it the fastest option in most cases.

**What about Critical and Atomic?**

* **Critical**: Ensures only one thread updates the shared sum at a time (heaviest synchronization cost).
* **Atomic**: Lighter than critical but still incurs overhead on each update.
* **Reduction**: Avoids these bottlenecks entirely by combining after all threads complete their local computations.

**Scheduling Impact:**

* **Static Scheduling**: Best when workload is evenly distributed.
* **Dynamic Scheduling**: Best when workload varies or the task is very large (like this array sum), helping avoid idle threads.

**6. Challenges Faced**

During the experimentation, several challenges were encountered that affected the performance evaluation and implementation:

* **Small Array Sizes Causing Higher Parallel Overhead**:  
  Initial tests with smaller array sizes resulted in the parallel version performing worse than the sequential one. This occurred because the overhead of creating and managing threads in OpenMP was greater than the time saved through parallel execution, highlighting that parallelization benefits become more prominent with larger datasets.
* **Improper Chunk Size Selection in Dynamic Scheduling**:  
  Without specifying an appropriate chunk size in dynamic scheduling, the parallel code performed unexpectedly poorly. In some cases, execution times exceeded **17 seconds**, significantly worse than the sequential execution. This was due to unbalanced workload distribution and excessive synchronization overhead. Setting a chunk size of **1000** significantly improved load balancing and reduced execution time.

**7. Lessons Learned**

This assignment provided valuable insights into parallel programming using OpenMP:

* **Parallelization is not always faster**: Small datasets may experience degraded performance when parallelized due to thread management overhead.
* **Chunk Size Matters**: The right chunk size in dynamic scheduling is crucial for balanced workload distribution and minimizing idle threads. Careful tuning can greatly impact the overall performance.
* **Reduction is Superior for Summation**: The reduction clause is the most efficient for parallel sum operations because it minimizes synchronization costs while ensuring accuracy.
* **Profiling is Essential**: Measuring performance at different configurations (array sizes, thread counts, scheduling strategies) is necessary to identify optimal settings for any given system and workload.

**8. Conclusion**

* OpenMP parallelization effectively reduces execution time for large array sums.
* **Reduction clause** is optimal for summing operations, while **critical** and **atomic** add unnecessary overhead.
* The choice of scheduling and thread count is crucial to maximizing performance.
* Dynamic scheduling with a proper chunk size enhances load balancing.

**9. References**

* OpenMP Documentation: <https://www.openmp.org/>
* CPU-Z Software for System Information
* Experimental Results from the performed tests