OS PROJECT REPORT

SERIAL VS OPEN MP VS PTHREADS

**Members:**

* k214869 Yasir Raza
* k213225 Qamber Raza

**Introduction:**

The choice of an appropriate sorting algorithm can significantly impact the performance and efficiency of data processing tasks. In this project, we aim to compare the performance of three popular sorting algorithms: Merge Sort, Bubble Sort, and Quick Sort, across different implementations.

**Sorting Algorithms Overview:**

Bubble Sort is a simple comparison-based algorithm that iteratively compares adjacent elements and swaps them if they are in the wrong order. With a time, complexity of O(n^2), it is one of the least efficient sorting algorithms.

Merge Sort is a divide-and-conquer algorithm that recursively splits the dataset into smaller halves, sorts them individually, and then merges the sorted halves to obtain the final sorted array. It has a time complexity of O (n log n) and is known for its stability and consistent performance.

Quick Sort is a highly efficient sorting algorithm with an average time complexity of O(n log n). It follows a divide-and-conquer approach by selecting a pivot element, partitioning the dataset based on the pivot, and recursively sorting the subarrays.

**Analysis:**

Performance Comparison (10 elements) in seconds

(Inserted values are average of 4 values)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Serial | Open MP | Pthread |
| Bubble | 0.003264 | 0.003383 | 0.005854 |
| Merge | 0.003527 | 0.003954 | 0.005118 |
| Quick | 0.003067 | 0.004319 | 0.004965 |

Performance Comparison (100 elements) in seconds

(Inserted values are average of 4 values)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Serial | Open MP | Pthread |
| Bubble | 0.059462 | 0.043466 | 0.004514 |
| Merge | 0.039391 | 0.037389 | 0.004358 |
| Quick | 0.035167 | 0.029171 | 0.004144 |

Performance Comparison (1000 elements) in seconds

(Inserted values are average of 4 values)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Serial | Open MP | Pthread |
| Bubble | 0.278064 | 0.152383 | 0.169854 |
| Merge | 0.135271 | 0.100894 | 0.151183 |
| Quick | 0.091673 | 0.083613 | 0.089651 |

**Findings:**

It is evident from our experiments that Quick Sort consistently outperforms both Merge Sort and Bubble Sort across all implementations. This outcome aligns with the theoretical expectations as Quick Sort has an average time complexity of O(n log n), making it more efficient for large datasets compared to Bubble Sort (O(n^2)) and Merge Sort (O(n log n)).

Furthermore, when comparing the serial implementations, Quick Sort showcases significantly faster execution times than Merge Sort and Bubble Sort. This performance advantage is due to Quick Sort's partitioning strategy, which allows it to quickly converge on the final sorted array. Merge Sort, while still efficient, requires additional merging steps to combine the sorted subarrays, leading to slightly slower execution times. Bubble Sort, with its quadratic time complexity, demonstrates the poorest performance among the three algorithms for larger datasets.

Comparing pthread and OpenMP implementations, we found that both techniques achieved similar performance improvements for Merge Sort and Quick Sort. However, OpenMP, with its simpler syntax and implicit thread management, offers a more convenient approach to parallel programming. It eliminates the need for explicit thread creation and synchronization, resulting in more concise and readable code. On the other hand, pthread provides finer control over thread creation and management, allowing for more flexibility in complex scenarios

**Future Work:**

1. Explore optimization techniques for the sorting algorithms, such as adaptive pivot selection in Quick Sort or optimized merging strategies in Merge Sort.
2. Investigate hybrid sorting algorithms that combine the strengths of different algorithms for improved performance.
3. Explore alternative parallelization strategies beyond pthread and OpenMP, such as MPI.
4. Conduct experiments with different hardware to analyze the impact of parallelization on various architectures, such as multi-core CPUs or GPUs.
5. Investigate the scalability of the parallelized sorting algorithms by testing them with larger datasets and varying numbers of threads or processes.

**Conclusion:** The speed of sorting in C using serial, OpenMP, or pthreads can depend on various factors such as the size of the dataset being sorted, the hardware configuration of the system, the specific sorting algorithm used, and the implementation of the parallelization technique. In general, parallelization techniques like OpenMP and pthread can speed up sorting by dividing the work among multiple threads, taking advantage of the multiple cores available on modern CPUs.

Therefore, the answer to whether serial, OpenMP, or pthread is faster for sorting in C depends on the specific use case and the factors mentioned above. It's best to benchmark and compare the performance of each approach for a specific application and dataset size to determine which one is the fastest.