The content of the report (.doc, .docx, .pdf or other open format files)

1. Description of the problem.
2. How you have parallelized the algorithm:

how you divide the work into separate parts?

how you allocte the subworks (parts) between threads?

how you synchronized the threads?

1. Describe an experiment:

system where it was performed (e.g., MIF cluster);

results achieved (relation between app parameters and speed up values must be shown as graphical form);

1. What are the conclusions:

what maximum speedup has been achieved?

is the speedup linear /perfect/, very good, not bad little, nothing?

if the speed up is not "perfect" what (you think) is a reason for this

does the experimental data shows if the algorithm is scalable?

**Parallel QuickSort Algorithm**

1. Description of the Problem

The objective of this project was to parallelize the QuickSort algorithm for efficient sorting of an array in a multi-threaded environment.

2. How I Parallelized the Algorithm

Dividing the Work into Separate Parts

I took a recursive approach, which involved the following steps:

Starting with the original array.

Selecting a pivot element and dividing the array into two subarrays: elements less than the pivot and elements greater than the pivot.

Recursively applying the same process to the two subarrays.

Allocating Subworks between Threads

I used Java's Fork-Join framework to create and manage a pool of threads for handling subarray sorting tasks. The framework efficiently distributed tasks among available threads, making optimal use of resources.

3. Description of the Experiment

Experimental System

The experiments were conducted on the MIF computing cluster, providing a multi-core computing environment suitable for parallel algorithm testing.

Results Achieved:

In this work I used different workloads to see the different speedups.

#nThreads #workload #timeS #speedup

1 1000 0.011 1.0

2 1000 0.01 116.54545454545456

4 1000 0.008 160.25

8 1000 0.009 142.44444444444446

------------------------

1 5000 0.018 1.0

2 5000 0.017 75.41176470588235

4 5000 0.02 64.1

8 5000 0.018 71.22222222222223

------------------------

1 15000 0.028 1.0

2 15000 0.036 35.611111111111114

4 15000 0.038 33.73684210526316

8 15000 0.037 34.64864864864865

Execution time was measured in seconds (#timeS), and the speedup (#speedup) was calculated relative to the single-threaded version. Speedup values significantly exceeded 1, indicating substantial performance improvement with increased threads.

Graph

4. Conclusions

Maximum Speedup Achieved

The highest speedup attained was 160.25, realized with 4 threads sorting an array of size 1000. So it demonstrates a significant performance enhancement compared to the single-threaded version.

Linearity of Speedup

Speedup values are not perfectly linear across all three tests, which is expected due to overhead from thread management, resource contention, and varying workloads.

Reasons for Speedup Behavior

Speedup can occur due to factors such as thread management overhead, resource contention, and workload variability.

Scalability

The algorithm can be updated, benefiting from parallelization and improved execution time with an increasing number of threads.