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 Performance Evaluation**

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. QuickSort is known for its speed, and the goal was to harness the power of parallel processing to enhance its performance.

2. How I Parallelized the Algorithm

Dividing the Work into Separate Parts

I took a recursive divide-and-conquer 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

The results are presented in the table below, demonstrating the relationship between the application parameters and the speedup values:

#nThreads #workload #timeS #speedup

1 1386 0.003 1.0

2 1386 0.003 427.33

4 1386 0.004 320.5

8 1386 0.002 641.0

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.

Graphical Representation

A graphical representation of the speedup values is provided below, highlighting the relationship between the number of threads and the achieved speedup.

[Insert Speedup Graph Here]

4. Conclusions

Maximum Speedup Achieved

The highest speedup attained was 641.0, realized with 8 threads sorting an array of size 1386. This demonstrates a significant performance enhancement compared to the single-threaded version.

Linearity of Speedup

Speedup values are not perfectly linear (1.0, 427.33, 320.5, 641.0), which is expected due to overhead from thread management, resource contention, and varying workloads.

Reasons for Speedup Behavior

Deviation from perfect linear speedup is attributed to factors such as thread management overhead, resource contention, and workload variability.

Scalability

The algorithm exhibits scalability to a certain degree, benefiting from parallelization and improved execution time with an increasing number of threads. Further optimization and experimentation may be necessary to address performance limitations and enhance scalability.