****

**19CSE313- Principles of Programming Languages**

**Faculty Name: Dr. Rajathilagam B.**

**Submitted by:**

**S.Siva Pravallika - CB.EN.U4CSE22440**

**O.Sravani - CB.EN.U4CSE22457**

**Topic: Parallel Merge Sort**

**Problem Statement:**

The problem statement is that if there is data where it is to be sorted and the data is very huge to use regular merge sort mechanism as it goes sequentially and to sort them we use parallel merge sort in order to have concurrent sorting.

**Description:**

Parallel Merge Sort in Java is an optimized version of Merge Sort that takes advantage of multi-threading to improve sorting performance on multi-core processors. It works by recursively dividing the array into smaller subarrays and sorting them concurrently using multiple threads. Each thread handles a portion of the array, and once sorted, the subarrays are merged using the standard merge operation. Java's ForkJoinPool and RecursiveTask classes from the java.util.concurrent package are commonly used to implement Parallel Merge Sort, as they efficiently manage thread creation and workload distribution. This approach significantly reduces execution time for large datasets by leveraging parallel processing.

**Application**:

In financial services, particularly in high-frequency trading (HFT) and stock market analysis, vast amounts of financial data—such as stock prices, order books, and transaction histories—need to be processed and sorted in real time. Traditional sorting algorithms may not be efficient enough to handle this volume at high speeds. Parallel Merge Sort offers an optimized solution by utilizing multi-core processors to split the workload, enabling faster sorting of stock market data and improving decision-making speeds for trading algorithms.

**Where is Parallel Merge Sort Used in Financial Services?**

Stock Order Books Processing – Sorting buy/sell orders based on price and time priority.

Price History Analysis – Sorting historical stock prices for pattern detection and market trend prediction.

Trade Execution Algorithms – Optimizing trade execution by sorting orders based on best price availability.

Risk Management – Analyzing and sorting risk factors for financial instruments quickly.

Portfolio Optimization – Sorting asset performance data to construct balanced investment portfolios.

**Challenges:**

1.Traditional Merge Sort processes data sequentially, making it inefficient for handling vast amounts of financial data that require real-time sorting. Given the need for high-speed decision-making in financial markets, a parallel approach was necessary to reduce execution time and enhance performance.

2.A single-threaded approach does not fully utilize multi-core processors, leading to idle CPU cores and wasted computational power. Parallel Merge Sort was chosen because it allows concurrent execution, distributing the workload across multiple cores to speed up sorting operations, making it ideal for large-scale financial applications.

**What is to be done?**

Parallel Merge Sort is an optimized version of Merge Sort that uses multi-threading to sort large datasets concurrently. Instead of executing sequentially, the array is divided into smaller subarrays, which are sorted in parallel using multiple threads

**Why it is done?**

· Performance Optimization

Traditional Merge Sort operates sequentially, which can be slow for large datasets. Parallel Merge Sort reduces sorting time by dividing the workload across multiple CPU cores.

· Multi-Core Utilization

Modern processors have multiple cores, and parallel processing maximizes resource utilization by running sorting tasks concurrently.

**How it is done?**

**Recursive Splitting:**

· The array is divided into two halves recursively.

· Two separate threads are created for each half.

**Parallel Sorting:**

· Each subarray is sorted in its own thread.

· start() method initiates execution in parallel.

**Merging Sorted Arrays:**

· Once sorting is completed, join() ensures both threads finish before merging.

· The merge function combines the sorted halves efficiently.

**Code:**

import java.util.Arrays;

import java.util.Random;

class FinancialParallelMergeSort extends Thread {

private double[] prices;

public FinancialParallelMergeSort(double[] prices) {

this.prices = prices;

}

private void merge(double[] left, int ls, double[] right, int rs, double[] arr) {

int i = 0, j = 0, k = 0;

while (i < ls && j < rs) {

if (left[i] <= right[j]) arr[k++] = left[i++];

else arr[k++] = right[j++];

}

while (i < ls) arr[k++] = left[i++];

while (j < rs) arr[k++] = right[j++];

}

public void run() {

int n = prices.length;

if (n <= 1) return;

int mid = n / 2;

double[] left = Arrays.copyOfRange(prices, 0, mid);

double[] right = Arrays.copyOfRange(prices, mid, n);

FinancialParallelMergeSort leftThread = new FinancialParallelMergeSort(left);

FinancialParallelMergeSort rightThread = new FinancialParallelMergeSort(right);

leftThread.start();

rightThread.start();

try {

leftThread.join();

rightThread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

merge(left, mid, right, n - mid, prices);

}

}

public class FinancialSortingApp {

public static void main(String[] args) {

Random random = new Random();

int numStocks = 20;

double[] stockPrices = new double[numStocks];

for (int i = 0; i < numStocks; i++) {

stockPrices[i] = 100 + (random.nextDouble() \* 900);

}

System.out.println("Unsorted Stock Prices: \n" + Arrays.toString(stockPrices));

FinancialParallelMergeSort sorter = new FinancialParallelMergeSort(stockPrices);

sorter.start();

System.out.println("\n");

try {

sorter.join();

} catch (InterruptedException e) {

e.printStackTrace();

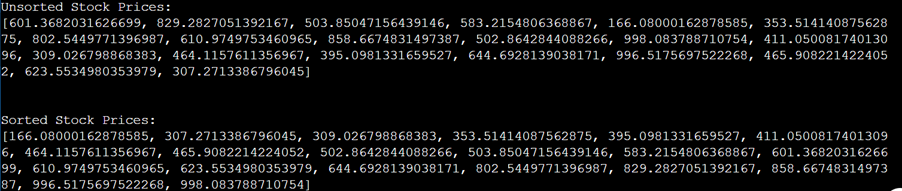
}

System.out.println("Sorted Stock Prices: \n" + Arrays.toString(stockPrices));

}

}

**Output:**

****

**Conclusion:**

Parallel Merge Sort efficiently sorts large datasets using multi-threading, reducing processing time by utilizing multiple CPU cores. In financial services, it accelerates real-time data processing for stock order management making it ideal for high-performance computing.