****

**De La Salle University • College of Computer Studies**

**Concurrent Merge Sort Application**

(Design and Evaluation of Its Performance)

Name (last name first) : Fernandez, Ryan Austin

Poblete, Clarisse Felicia M.

San Pedro, Marc Dominic

Tan, Johansson E.

Section : G01

Date of Submission : January 26, 2016

**I. Source Code**

**Driver.java**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.File;

public class Driver {

public static int numElem = 8000000;

public static int[] arr = new int[numElem];

public static void main(String args[]) throws Exception {

//args is the power of 2

int power = Integer.parseInt(args[0]);

ThreadPool threadPool = ThreadPool.init((int)Math.pow(2,power));

long ave = 0;

int[] storage = new int[numElem];

BufferedReader br = null;

try {

br = new BufferedReader(new FileReader(new File("nums.txt")));

} catch(Exception e) {

e.printStackTrace();

}

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

String s = br.readLine();

try {

storage[i] = Integer.parseInt(s);

} catch(Exception e) {

System.out.println(s);

}

}

br.close();

for(int i = 0; i < 5; i++) {

threadPool.instance().reset();

for(int j = 0; j < numElem; j++) {

arr[j] = storage[j];

}

long time = System.currentTimeMillis();

MergeSorter m = threadPool.instance().getMergeSorter(0,numElem - 1);

m.mergeSort();

time = System.currentTimeMillis() - time;

ave += time;

System.out.println("Time: " + (time / 1000.0) + "s");

}

System.out.println("Average Time: " + (ave / 5000.0) + "s");

}

}

**MergeSorter.java**

public abstract class MergeSorter implements Runnable{

protected int start;

protected int end;

public MergeSorter(int start, int end) {

this.start = start;

this.end = end;

}

public void mergeSort() {

if(start < end) {

sort();

merge();

}

}

protected abstract void sort();

protected int mid() {

return start + (end - start) / 2;

}

private void merge() {

merge(start, mid(), end);

}

protected void merge( int s, int m, int e ) {

int size1 = m - s + 1;

int size2 = e - m;

int[] left = new int[size1];

int[] right = new int[size2];

int i;

int j;

int k;

for( i = 0; i < size1; i++ ) {

left[i] = Driver.arr[ s + i ];

}

for( j = 0; j < size2; j++ ) {

right[j] = Driver.arr[ m + 1 + j ];

}

i = j = 0;

for( k = s; k <= e; k++ ) {

if( j == size2 || ( i != size1 && left[i] < right[j] ) ){

Driver.arr[k] = left[i];

i++;

} else {

Driver.arr[k] = right[j];

j++;

}

}

}

public void run() {

mergeSort();

}

}

**LinearSorter.java**

public class LinearSorter extends MergeSorter {

public LinearSorter(int start, int end) {

super(start,end);

}

protected void sort() {

sort( start, mid() );

sort( mid() + 1, end );

}

protected void sort(int start, int end) {

if( start < end ) {

int m = start + ( end - start ) / 2;

sort( start, m );

sort( m + 1, end );

merge( start, m, end );

}

}

}

**ParallelSorter.java**

public class ParallelSorter extends MergeSorter{

private Thread left;

private Thread right;

public ParallelSorter(int start, int end) {

super(start,end);

}

protected void sort() {

try {

left = new Thread(ThreadPool.instance()

.getMergeSorter(start, mid()));

right = new Thread(ThreadPool.instance()

.getMergeSorter(mid() + 1, end));

left.start();

right.start();

left.join();

right.join();

} catch(Exception e) {

e.printStackTrace();

}

}

}

**ThreadPool.java**

public class ThreadPool {

private static ThreadPool instance = null;

private volatile int resources;

private int resourceCount;

private ThreadPool(int resources) {

resourceCount = this.resources = resources;

}

public static ThreadPool init(int resources) throws Exception {

if( instance == null ) {

instance = new ThreadPool(resources);

return instance;

} else {

throw new Exception("Thread pool already initialized.");

}

}

public static ThreadPool instance() throws Exception {

if(instance == null) {

throw new Exception("Thread pool not initialized.");

} else {

return instance;

}

}

public void reset() {

resources = resourceCount;

}

public synchronized MergeSorter getMergeSorter(int start, int end) {

if( resources >= 2 ) {

resources -= 2;

return new ParallelSorter(start,end);

} else {

return new LinearSorter(start,end);

}

}

}

**II. Analysis**

**Appendix A: Comparative Results**