Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра кібербезпеки та програмного забезпечення

**Звіт**

з виконаної лабораторної роботи № 4

Дисципліна: Паралельні та розподілені обчислення

на тему

«Паралельні обчислення в багатопроцесорних системах.

Технологія Fork-Join.»

Виконав :

студент академічної групи КІ-15

Аннаєв А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив :

Викладач

Минайленко Р. М.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кропивницкий- 2018

**Мета:** Одержати навички створення паралельних обчислень у багатопроцесорних системах.

**Завдання:**

- Створити додаток, що реалізує паралельні обчислення в багатопроцесорних системах.

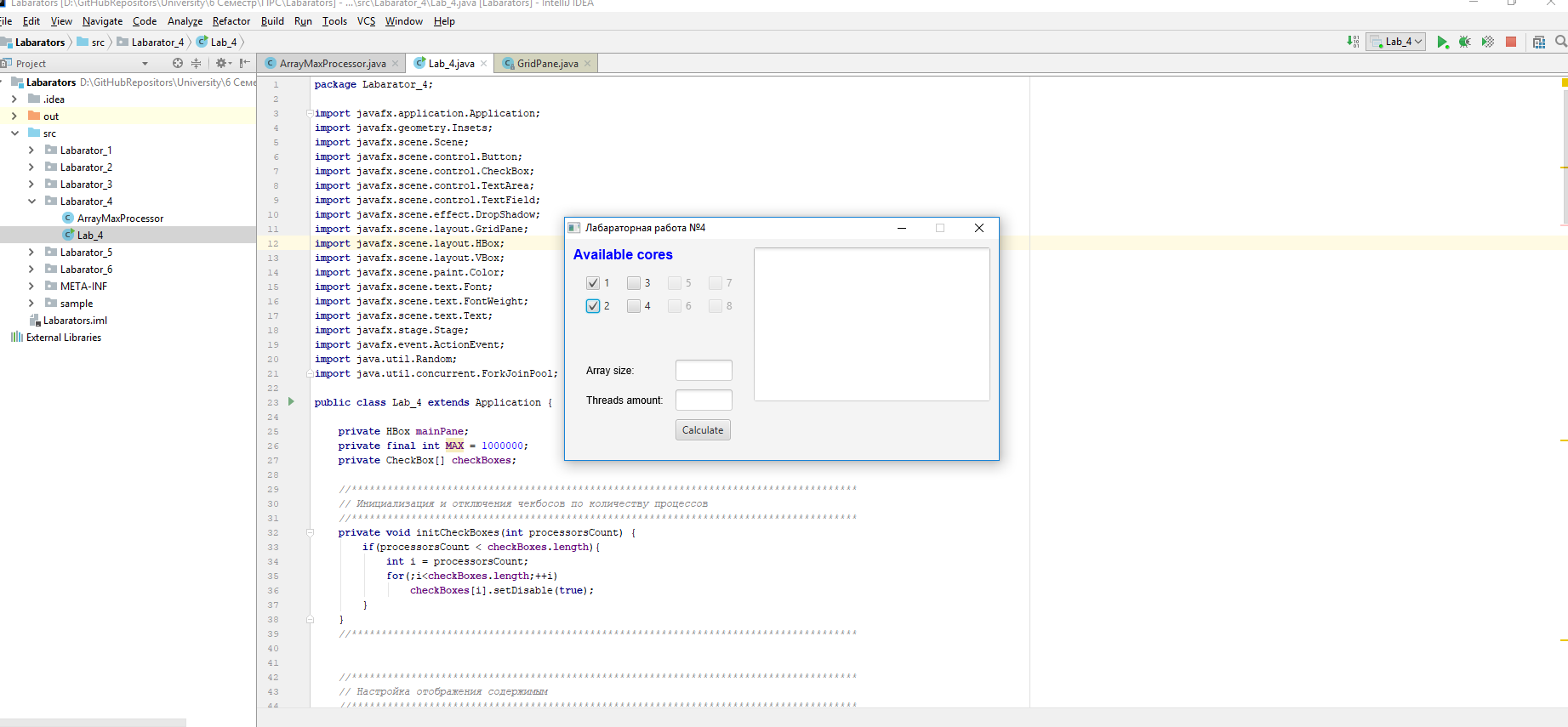
- Реалізувати паралелізм на рівні окремих процесорів.

- Зробити аналіз результатів виконання паралельних обчислень для одного і декількох процесорів.

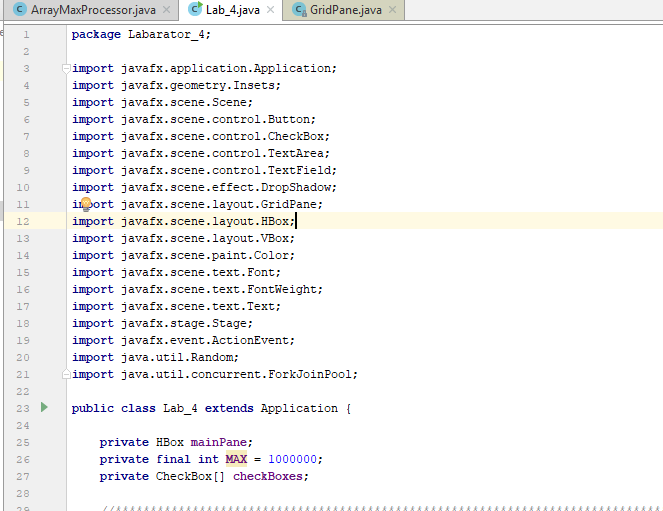
- Зробити аналіз ефективності використання великої кількості паралельних потоків обчислень.

- Зробити висновок про оптимальну кількість процесорів в обчислювальній системі з погляду співвідношення ефективності/вартості.

- Зробити висновок про оптимальну кількість потоків в обчислювальній системі для оптимальної кількості процесорів.



**public class** Lab\_4 **extends** Application {  
  
 **private** HBox **mainPane**;  
 **private final int MAX** = 1000000;  
 **private** CheckBox[] **checkBoxes**;  
  
 *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Инициализация и отключения чекбосов по количеству процессов  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private void** initCheckBoxes(**int** processorsCount) {  
 **if**(processorsCount < **checkBoxes**.**length**){  
 **int** i = processorsCount;  
 **for**(;i<**checkBoxes**.**length**;++i)  
 **checkBoxes**[i].setDisable(**true**);  
 }  
 }  
*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Настройка отображения содержимым  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private void** createLayout(){  
  
 Text processorsLabel = createLabel(**"Available cores"**, Color.***BLUE***, FontWeight.***BOLD***, 16);  
 Text arraySizeLabel = createLabel(**"Array size: "**, Color.***BLACK***, FontWeight.***NORMAL***, 12);  
 Text threadsAmountLabel = createLabel(**"Threads amount: "**, Color.***BLACK***, FontWeight.***NORMAL***, 12);  
  
 TextField tfArraySize = **new** TextField();  
 TextField tfThreadsCount = **new** TextField();  
  
 **checkBoxes** = **new** CheckBox[8];  
  
 **for**(**int** i=0; i<**checkBoxes**.**length**; ++i)  
 **checkBoxes**[i] = **new** CheckBox(String.*valueOf*(i+1));  
  
 Button calcButton = **new** Button(**"Calculate"**);  
 TextArea textArea = **new** TextArea();  
  
 GridPane grid = **new** GridPane();  
 grid.setPadding(**new** Insets(15, 15, 15, 15));  
 grid.setHgap(20);  
 grid.setVgap(10);  
 grid.add(**checkBoxes**[0], 0, 0);  
 grid.add(**checkBoxes**[1], 0, 1);  
 grid.add(**checkBoxes**[2], 1, 0);  
 grid.add(**checkBoxes**[3], 1, 1);  
 grid.add(**checkBoxes**[4], 2, 0);  
 grid.add(**checkBoxes**[5], 2, 1);  
 grid.add(**checkBoxes**[6], 3, 0);  
 grid.add(**checkBoxes**[7], 3, 1);  
  
 GridPane grid2 = **new** GridPane();  
 grid2.setPadding(**new** Insets(40, 15, 15, 15));  
 grid2.setHgap(10);  
 grid2.setVgap(10);  
 grid2.add(arraySizeLabel, 0, 0);  
 grid2.add(threadsAmountLabel, 0, 1);  
 grid2.add(tfArraySize, 1, 0);  
 grid2.add(tfThreadsCount, 1, 1);  
 grid2.add(calcButton, 1, 2);  
  
 *//HBox mainHBox = new HBox();* VBox leftVBox = **new** VBox();  
 leftVBox.getChildren().addAll(processorsLabel, grid, grid2);  
  
 VBox rightVBox = **new** VBox();  
 rightVBox.getChildren().addAll(textArea);  
  
 *// Setup shadow* DropShadow dropShadow = **new** DropShadow();  
 dropShadow.setRadius(5.0);  
 dropShadow.setOffsetX(3.0); *// Shadow offset (X and Y axis)* dropShadow.setOffsetY(3.0);  
 dropShadow.setColor(Color.***GRAY***); *// Shadow color* **mainPane** = **new** HBox();  
 **mainPane**.setSpacing(10);  
 **mainPane**.setEffect(dropShadow);  
 **mainPane**.setPadding(**new** Insets(10, 10, 10, 10));  
 **mainPane**.getChildren().addAll(leftVBox, rightVBox);  
  
 *// Action* calcButton.setOnAction((ActionEvent evn) -> {  
  
 **int** size = Integer.*valueOf*(tfArraySize.getText());  
 **int** threadsCount = Integer.*valueOf*(tfThreadsCount.getText());  
 **int** selectedProcessorsCount = 0;  
  
 *// Read amount of checked cores* **for**(**int** i=0; i<**checkBoxes**.**length**; ++i)  
 {  
 **if**(**checkBoxes**[i].isSelected())  
 selectedProcessorsCount++;  
 }  
  
 **if**(selectedProcessorsCount == 0)  
 selectedProcessorsCount = 1;  
  
textArea.appendText(**"Cores selected: "** + String.*valueOf*(selectedProcessorsCount) + **"\n"**);  
 **int**[] arr = generateArray(size);  
  
  
 ArrayMaxProcessor maxTask = **new** ArrayMaxProcessor(arr, 0, size, threadsCount);  
 ForkJoinPool forkJoinPool = **new** ForkJoinPool(selectedProcessorsCount);  
 **long** startTime = System.*currentTimeMillis*();  
 **int** maxValue = forkJoinPool.invoke(maxTask);  
 **long** elapsedMsecs = System.*currentTimeMillis*() - startTime;  
 textArea.appendText(**"Maximum value: "** + String.*valueOf*(maxValue) + **"\n"**);  
 textArea.appendText(**"Elapsed time: "** + String.*valueOf*(elapsedMsecs) + **" msecs\n\n\n"**);  
 });  
 } *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* //  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private** Text createLabel(String text, Color color, FontWeight fontWeight, **int** fontSize){  
 Text t = **new** Text();  
 t.setText(text);  
 t.setFont(Font.*font*(**"Arial"**, fontWeight, fontSize));  
 t.setFill(color);  
 **return** t;  
 }  
*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private int**[] generateArray(**int** size) {  
 **int**[] arr = **new int**[size];  
  
 Random r = **new** Random();  
  
 **for**(**int** i=0; i<size; ++i)  
 {  
 arr[i] = r.nextInt(**MAX**);  
 }  
  
 **return** arr;  
 }  
 *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Точка входа в программу  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **public static void** main(String[] args) { *launch*(args); }  
 *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** @Override  
 **public void** start(Stage stage) **throws** Exception  
 {  
 createLayout();  
  
 **int** numberOfProcessors = Runtime.*getRuntime*().availableProcessors();  
 initCheckBoxes(numberOfProcessors);  
  
 stage.setTitle(**"Лабараторная работа №4"**);  
 stage.setResizable(**false**);  
 Scene scene = **new** Scene(**mainPane**, 500, 250, Color.***TRANSPARENT***);  
 stage.setScene(scene);  
 stage.show();  
  
 }  
}



**package** Labarator\_4;  
  
**public class** ArrayMaxProcessor **extends** RecursiveTask<Integer> {  
  
 **private final int MIN\_ARR\_ELEMENTS\_PER\_THREAD** = 10;  
 **private int threadsCount**; *// Amount of availble threads* **private int**[] **numsArr**;  
 **private int arrLength**;  
 **private int begin**;  
 **private int end**;  
 **private int max**;  
 *//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Конструктор  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **public** ArrayMaxProcessor(**int**[] arr, **int** begin, **int** end, **int** threadsCount)  
 {  
 **numsArr** = arr;  
 **arrLength** = **numsArr**.**length**;  
 **this**.**threadsCount** = threadsCount;  
  
 **this**.**begin** = begin;  
 **this**.**end** = end;  
 }  
*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Создание подзадачи  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private** List<ArrayMaxProcessor> createSubtasks() {  
 List<ArrayMaxProcessor> subtasks = **new** ArrayList<ArrayMaxProcessor>();  
  
 **int** middle = (**this**.**end** - **this**.**begin**) / 2;  
 ArrayMaxProcessor subtask1 = **new** ArrayMaxProcessor(**this**.**numsArr**, **begin**, middle, **this**.**threadsCount** / 2);  
 ArrayMaxProcessor subtask2 = **new** ArrayMaxProcessor(**this**.**numsArr**, middle, **end**, **this**.**threadsCount** / 2);  
 subtasks.add(subtask1);  
 subtasks.add(subtask2);  
 **return** subtasks;  
 }  
*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Получение максимального значения массива  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** **private int** getMax()  
 {  
 **int** max = 0;  
  
 **for**(**int** i=**begin**; i<**end**; ++i)  
 **if**(max < **numsArr**[i])  
 max = **numsArr**[i];  
  
 **return** max;  
 }  
*//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\** @Override  
 **protected** Integer compute(){  
 **int** length = **end** - **begin**;  
  
 **if**(**threadsCount** >= 2) {  
 **if** (length > **MIN\_ARR\_ELEMENTS\_PER\_THREAD**) {  
 **threadsCount** -= 2;  
 List<ArrayMaxProcessor> subtasks =**new** ArrayList<ArrayMaxProcessor>();  
 subtasks.addAll(createSubtasks());  
  
 **for**(ArrayMaxProcessor subtask : subtasks){  
 subtask.fork();  
 }  
  
 **int** result = 0;  
 **for**(ArrayMaxProcessor subtask : subtasks) {  
 **int** v = subtask.join();  
 **if**(v > result)  
 result = v;  
 }  
 **threadsCount** += 2;  
 **return** result;  
 }  
 **else** {  
 **return** getMax();  
 }  
 }  
 **else** {  
 **return** getMax();  
 }  
 }  
}

