CSCI 654-01—Foundations of Parallel Computing
Programming Project 2

Prof. Alan Kaminsky—Fall Semester 2018
Rochester Institute of Technology—Department of Computer Science

---

Overview

Write a sequential program and a parallel program in Java using the Parallel Java 2 Library. Run the programs on a cluster parallel computer to learn about cluster parallel programming and weak scaling.

Help with your project: I am willing to help you with the design of your project. I am willing to help you debug your project if the code isn't working. However, for help with design or debugging issues *you must come see me in person.* Either visit me during office hours or make an appointment. I will not help you with design or debugging issues via email. If it's the evening of the project deadline and I have gone home, you are on your own. Plan and work ahead so there will be plenty of time for me to help you if necessary.

Largest Triangle

Given a group of two-dimensional points, we want to find the **largest triangle,** namely three distinct points that are the vertices of a triangle with the largest area. The area of a triangle is $(s(s-a)(s-b)(s-c))^{1/2}$, where a, b, and c are the lengths of the triangle's sides and $s = (a+b+c)/2$. The length of a triangle's side is the Euclidean distance between the side's two endpoints.

You will write sequential and cluster parallel versions of a program that finds the largest triangle in a group of points. The program's command line argument is a *constructor expression* for a

PointSpec object from which the program obtains the points' (*x*,*y*) coordinates. For more information about constructor expressions, see *Big CPU, Big Data* Chapter 15.

Here is the source file for interface PointSpec: PointSpec.java

Here is the source file for class Point, used by interface PointSpec: Point.java

Your programs must use interface PointSpec and class Point. You are not allowed to change these in any way.

Here is the source file for class RandomPoints, which implements interface PointSpec and generates points at random locations in a square region: RandomPoints.java

To test your programs, you may use class RandomPoints or your own class that properly implements interface PointSpec. When I grade your programs, I might or might not use class RandomPoints to specify the points.

Program Input and Output

The program's command line argument is a constructor expression for a PointSpec object from which the program obtains the points' (*x*,*y*) coordinates. The first point is at index 0, the second point is at index 1, and so on.

The program must print four lines of output, as in this example:

```
$ java pj2 LargestTriangleSeq "RandomPoints(100,100,142857)"
9 95.951 7.8408
12 98.248 97.938
80 2.3838 77.670
4295.3
```

The first line contains the index of the largest triangle's first vertex point, a space character, the X coordinate of the first vertex point, a space character, the Y coordinate of the first vertex point, and a newline. The line must be printed with this statement:

```
System.out.printf ("%d %.5g %.5g%n", index, x, y);
```

The second line contains the index and coordinates of the largest triangle's second vertex point. The third line contains the index and coordinates of the largest triangle's third vertex point. The first three lines must be printed in ascending order of the index.

The fourth line contains the largest triangle's area. The line must be printed with this statement:

```
System.out.printf ("%.5g%n", area);
```

The program must print the triangle with the largest area. If more than one triangle has the same largest area, the program must print the triangle with the smallest first index. If more than one triangle has the same largest area and smallest first index, the program must print the triangle with the smallest second index. If more than one triangle has the same largest area and smallest first index and smallest second index, the program must print the triangle with the smallest third index.

Software Requirements

1. The sequential version of the program must be run by typing this command line:
2. `java pj2 jar=<jarfile> LargestTriangleSeq "<pointspec>"`
    a. `<jarfile>` is the name of a Java Archive file containing the compiled class files for your project.
    b. `<pointspec>` is a constructor expression for a point specification object.

    Note: This means that the program's class must be named `LargestTriangleSeq`, this class must not be in a package, and this class must extend class edu.rit.pj2.Task.

    Note: On the `tardis` computer, you must include the `jar=` option. For further information, see Parallel Java 2 on the RIT CS Parallel Computers.

3. The parallel version of the program must be run by typing this command line:
4. `java pj2 jar=<jarfile> workers=<K> LargestTriangleClu "<pointspec>"`
    a. `<jarfile>` is the name of a Java Archive file containing the compiled class files for your project.
    b. `<K>` is the number of worker tasks.
    c. `<pointspec>` is a constructor expression for a point specification object.

    Note: This means that the program's class must be named `LargestTriangleClu`, this class must not be in a package, and this class must extend class edu.rit.pj2.Job.

    Note: On the `tardis` computer, you must include the `jar=` option. For further information, see Parallel Java 2 on the RIT CS Parallel Computers.

5. If the command line does not have the required number of arguments, if any argument is erroneous, or if any other error occurs, the program must print an error message on the console and must exit. The error message must describe the problem. The wording of the error message is up to you.
6. The program must print on the console the information specified above under Program Input and Output.

    *Note: If your program's output does not conform **exactly** to Software Requirements 1 through 4, **you will lose credit** on your project.* See the Grading Criteria below.

Software Design Criteria

1. The programs must follow the cluster parallel programming patterns studied in class.
2. The programs must be designed using object oriented design principles as appropriate.
3. The programs must make use of reusable software components as appropriate.
4. Each class or interface must include a Javadoc comment describing the overall class or interface.
5. Each constructor or method within each class or interface must include a Javadoc comment describing the overall constructor or method, the arguments if any, the return value if any, and the exceptions thrown if any.

   Note: See my Java source files which we studied in class for the style of Javadoc comments I'm looking for.

   Note: If your program's design does not conform to Software Design Criteria 1 through 5, ***you will lose credit*** *on your project.* See the Grading Criteria below.

Submission Requirements

Your project submission will consist of a ZIP file containing the Java source file for every class and interface in your project. Put all the source files into a ZIP file named "`<username>.zip`", replacing `<username>` with the user name from your Computer Science Department account. On a Linux system, the command for creating a ZIP file is:

```
zip <username>.zip *.java
```

DO NOT include the Parallel Java 2 Library in your ZIP file!

Send your ZIP file to me by email at ark@cs.rit.edu. Include your full name and your computer account name in the email message, and include the ZIP file as an attachment.

When I get your email message, I will extract the contents of your ZIP file into a directory. I will copy the PointSpec.java and Point.java source files into the directory where I extracted your files (overwriting any PointSpec.java or Point.java source files in your ZIP file). I will set my Java class path to include the directory where I extracted your files, plus the Parallel Java 2 Library. I will compile all the Java source files using the JDK 1.7 compiler. I will then send you a reply message acknowledging I received your project and stating whether I was able to compile all the source files. If you have not received a reply within one business day (i.e., not counting weekends), please contact me. Your project is not successfully submitted until I have sent you an acknowledgment stating I was able to compile all the source files.

The submission deadline is Wednesday, October 17, 2018 at 11:59pm. The date/time at which your email message arrives in my inbox will determine whether your project meets the deadline.

You may submit your project multiple times up until the deadline. I will keep and grade only the most recent successful submission. There is no penalty for multiple submissions.

If you submit your project before the deadline, but I do not accept it (e.g. I can't compile all the source files), and you cannot or do not submit your project again before the deadline, the project will be late (see below). ***I STRONGLY advise you to submit the project several days BEFORE the deadline, so there will be time to deal with any problems that might arise in the submission process.***

Grading Criteria

I will grade your project by:

- (10 points) Evaluating the design of your sequential and parallel programs, as documented in the Javadoc and as implemented in the source code.
  - All of the Software Design Criteria are fully met: 10 points.
  - Some of the Software Design Criteria are not fully met: 0 points.
- (10 points) Running your sequential and parallel programs. There will be ten test cases, each worth 1 point. For each test case, if both programs run using the command lines in Requirements 1 and 2 and both programs produce the correct output, the test case will get 1 point, otherwise the test case will get 0 points. "Correct output" means "output fulfills *all* the Software Requirements *exactly*."
- (10 points) Weak scaling performance. I will run your parallel program on the `tardis` cluster, on a certain test case with 6000 points, with $K = 1$ worker task, three times, and take the smallest running time T_1. I will run your parallel program on the `tardis` cluster, on a certain test case with 9500 points, with $K = 4$ worker tasks, three times, and take the smallest running time T_2. (My program's running times are about 60 seconds on these test cases.)
  - Correct output and $T_1 \leq 70$ seconds and $T_2 \leq 70$ seconds: 10 points.
  - Correct output and $T_1 \leq 80$ seconds and $T_2 \leq 80$ seconds: 8 points.
  - Correct output and $T_1 \leq 90$ seconds and $T_2 \leq 90$ seconds: 6 points.
  - Correct output and $T_1 \leq 100$ seconds and $T_2 \leq 100$ seconds: 4 points.
  - Correct output and $T_1 \leq 110$ seconds and $T_2 \leq 110$ seconds: 2 points.
  - Otherwise: 0 points.
- (30 points) Total.

When I run your programs, the Java class path will point to the directory with your compiled class files, plus the Parallel Java 2 Library. I will use JDK 1.7 to run your program.

I will grade the test cases based *solely* on whether your program produces the correct output as specified in the above Software Requirements. *Any* deviation from the requirements will result in

a grade of 0 for the test case. This includes errors in the formatting (such as extra spaces), output lines not terminated with a newline, and extraneous output not called for in the requirements. The requirements state *exactly* what the output is supposed to be, and there is no excuse for outputting anything different. If any requirement is unclear, please ask for clarification.

If there is a defect in your program and that same defect causes multiple test cases to fail, I will deduct points for *every* failing test case. The number of points deducted does *not* depend on the size of the defect; I will deduct the same number of points whether the defect is 1 line, 10 lines, 100 lines, or whatever.

After grading your project I will put your grade and any comments I have in your encrypted grade file. For further information, see the Course Grading and Policies and the Encrypted Grades.

Test Cases

The grading test case commands and correct printouts were as follows. For test cases 1–2, the wording of the error message was up to you, as long as it described the problem. For test cases 3–10 and the scaling test cases, the output was deemed correct if the largest triangle's area was correct (regardless of which points comprised that triangle).

This ZIP archive contains the source files for the point spec classes I used in the test cases: p2.zip

IMPORTANT NOTE
Running on a `tardis` cluster node, each test case 1–10 should have taken about 2 seconds, and the scaling test cases should have taken about 60 seconds. I set a 10-second timeout on test cases 1–10 and a 120-second timeout on the scaling test cases. If the program timed out, it indicates a bad design.

IMPORTANT NOTE
I have accurately recorded in your grade file what I observed during my testing. If you run the test cases on your machine and do not get the same results as I, then there is a flaw in your design somewhere that causes inconsistent results (perhaps due to incorrect parallelization), and I am not going to change your grade for the original submission. You may still resubmit your project.

```
1.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq
    Usage: java pj2 LargestTriangleSeq <ctor>
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu
    Usage: java pj2 [workers=<K>] LargestTriangleClu <ctor>

2.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "BadPoints(10)"
    LargestTriangleSeq: Cannot construct point spec "BadPoints(10)"
    Usage: java pj2 LargestTriangleSeq <ctor>
```

```
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "BadPoints(10)"
    LargestTriangleClu: Cannot construct point spec "BadPoints(10)"
    Usage: java pj2 [workers=<K>] LargestTriangleClu <ctor>

3.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "CirclePoints(700,10)"
    230 -4.7387 8.8060
    463 -5.2823 -8.4910
    697 9.9964 -0.26925
    129.90
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "CirclePoints(700,10)"
    230 -4.7387 8.8060
    463 -5.2823 -8.4910
    697 9.9964 -0.26925
    129.90

4.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "CirclePoints(800,20)"
    263 -9.4971 17.601
    530 -10.450 -17.053
    796 19.990 -0.62822
    519.61
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "CirclePoints(800,20)"
    263 -9.4971 17.601
    530 -10.450 -17.053
    796 19.990 -0.62822
    519.61

5.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "LinePoints(700)"
    0 0.0000 0.0000
    1 1.0000 0.0000
    2 2.0000 0.0000
    0.0000
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "LinePoints(700)"
    0 0.0000 0.0000
    1 1.0000 0.0000
    2 2.0000 0.0000
    0.0000

6.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "LinePoints(800)"
    0 0.0000 0.0000
    1 1.0000 0.0000
    2 2.0000 0.0000
    0.0000
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "LinePoints(800)"
    0 0.0000 0.0000
    1 1.0000 0.0000
    2 2.0000 0.0000
    0.0000
```

```
7.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "RandomPoints(700,10,142857)"
    344 0.0057381 0.23250
    560 0.25923 9.9400
    634 9.9853 0.58002
    48.394
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "RandomPoints(700,10,142857)"
    344 0.0057381 0.23250
    560 0.25923 9.9400
    634 9.9853 0.58002
    48.394

8.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "RandomPoints(800,20,285714)"
    519 19.882 19.950
    606 0.029248 17.119
    697 19.551 0.0078466
    197.48
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "RandomPoints(800,20,285714)"
    519 19.882 19.950
    606 0.029248 17.119
    697 19.551 0.0078466
    197.48

9.  $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "WigglePoints(20)"
    0 0.0000 0.0000
    5 1.5708 1.0000
    17 5.3407 -0.80902
    3.3058
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "WigglePoints(20)"
    0 0.0000 0.0000
    5 1.5708 1.0000
    17 5.3407 -0.80902
    3.3058

10. $ java pj2 timelimit=10 jar=p2.jar debug=none workers=1
LargestTriangleSeq "WigglePoints(40)"
    0 0.0000 0.0000
    11 1.7279 0.98769
    34 5.3407 -0.80902
    3.3364
    $ java pj2 timelimit=10 jar=p2.jar debug=none workers=4
LargestTriangleClu "WigglePoints(40)"
    0 0.0000 0.0000
    11 1.7279 0.98769
    34 5.3407 -0.80902
    3.3364

Scaling test case
    $ java pj2 timelimit=120 jar=p2.jar debug=makespan workers=1
LargestTriangleClu "SquarePoints(1500)"
    1500 1500.0 0.0000
    3000 1500.0 1500.0
```

```
    4637 0.0000 1363.0
    1.1250e+06
    $ java pj2 timelimit=120 jar=p2.jar debug=makespan workers=4
LargestTriangleClu "SquarePoints(2375)"
    0 0.0000 0.0000
    2542 2375.0 167.00
    7125 0.0000 2375.0
    2.8203e+06
```

Late Projects

If I have not received a successful submission of your project by the deadline, your project will be late and will receive a grade of zero. You may request an extension for the project. There is no penalty for an extension. See the Course Policies for my policy on extensions.

Plagiarism

Programming Project 2 must be entirely your own individual work. I will not tolerate plagiarism. If in my judgment the project is not entirely your own work, you will automatically receive, as a minimum, a grade of zero for the assignment. See the Course Policies for my policy on plagiarism.

Resubmission

If you so choose, you may submit a revised version of your project after you have received the grade for the original version. However, if the original project was not successfully submitted by the (possibly extended) deadline or was not entirely your own work (i.e., plagiarized), you are not allowed to submit a revised version. Submit the revised version via email in the same way as the original version. I will accept a resubmission up until 11:59pm Wednesday 24-Oct-2018. You may resubmit your project multiple times up until the deadline; I will keep and grade only the most recent successful resubmission; there is no penalty for multiple resubmissions. I will grade the revised version using the same criteria as the original version, then I will subtract 3 points (10% of the maximum possible points) as a resubmission penalty. The revised grade will replace the original grade, even if the revised grade is less than the original grade.