Assignment 2 part B

Due date: July 3rd at 11:59pm Waterloo time

ECE 454 / 750: Distributed Computing

Instructor: Dr. Wojciech Golab wgolab@uwaterloo.ca
Assignment TA: Mohammad Hamdaqa
mhamdaqa@uwaterloo.ca

A few house rules

Collaboration:

- groups of 1, 2 or 3 students
- undergraduate and graduate students may be in the same group

Submission:

- electronic submission (one per group) using drop box in LEARN
- submit two files: tarball with code and design doc (max 1 page) in one of the following formats: pdf, doc, docx, rtf, ppt, or pptx

Evaluation:

- part B of A2 is worth 2/3 of the total weight of A2
- if in doubt, please post questions in Piazza!

Learning objective

Objective: to understand the relative merits of shared memory parallel computing versus distributed cluster computing.

In this assignment you will experiment with shared memory parallel computing, and later on in the course we will study distributed cluster computing frameworks.

In the end we will see that smaller can be much more efficient.

Overview

In this assignment you will be counting (in fact enumerating) triangles in an undirected graph.

Triangle counting is a fundamental graph analytics problem and has been studied intensively in the context of community detection in social networks, spam detection, link recommendation, and detection of protein interactions.

Your goal in this assignment is to produce a solution to the problem that satisfies the following properties:

- It runs in a **single multi-threaded Java process**.
- It is efficient in terms of running time versus input size.
- It scales well with the number of available cores.

Starter code

- Starter code is provided for you at the following URL: http://ece.uwaterloo.ca/~wgolab/ece454750a2.tar.gz
- The tarball includes the following:
  - A build.xml file to build your code using the ant tool.
  - Code under src/, including the main Java class: ece454750s15a2/TriangleCount.java
  - A sample input file: samplegraph.txt
  - A Linux shell script to package your code for electronic submission: maketarball.sh
- To run the code do the following from bash in on ecelinux:
  - mkdir ece454750a2 ; cd ece454750a2
  - wget https://ece.uwaterloo.ca/~wgolab/ece454750a2.tar.gz
  - tar -xf ece454750a2.tar.gz
  - ant ; cat output.txt

Group membership

- The TriangleCountImpl class contains a method called getGroupMembers.
- Modify the implementation of this method so that it returns the Nexus IDs of all the members of your group.
- The grader may rely on the output of this method to record your assignment grades correctly.

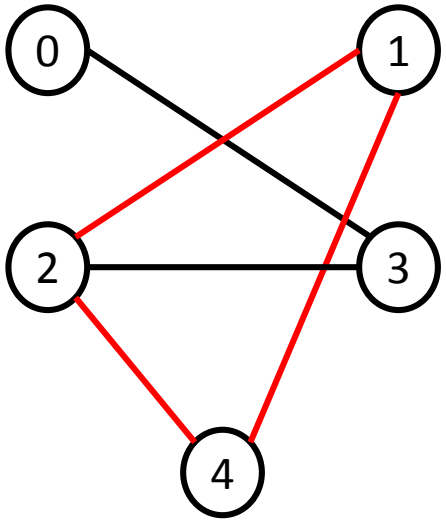
Some more house rules

- Any additional code you write should be placed in the ece454750s15a2 package under src/.
- Write your solution by modifying the TriangleCountImpl class.
- Do not modify the Triangle and TriangleCount classes.
- You may use Java 1.6, or Scala 2.11.6. The provided build.xml file cross-complies Java code to version 1.6 automatically.
- The code must work on ecelinux / eceubuntu.
- The grader will execute your code from the directory where you run the ant tool to build the code. Example command line for ecelinux / eceubuntu:

```
java -cp "a2.jar:/opt/scala-2.11.6/lib/scala-library.jar"
ece454750s15a2.TriangleCount -ncores 2 -if graph.txt -of output.txt
```

Input file: samplegraph.txt

```
5 vertices, 5 edges
0: 3
1: 2 4
2: 1 3 4
3: 2 0
4: 2 1
```



Correct output:
1 2 4

Output format

- The starter code mostly takes care of the output format for you. The output is written to a file whose name is determined by the -of argument to TriangleCount.
- The output file contains one line per triangle. Each triangle is specified as a triplet of vertices separated by spaces and listed in ascending order (e.g., 1 2 4).
- Your task is to return the triangles as a List object from the enumerateTriangles method of class TriangleCountImpl.
- The order in which your method implementation outputs the triangles does not matter, but each triangle should be output only once.

Additional inputs

- Larger inputs for testing are provided at http://ece.uwaterloo.ca/~wgolab/graphs.tar.gz
- You can specify the graph input file using the -if argument to TriangleCount.
- Start experimenting with the smallest graphs, which contain 1000 vertices and various numbers of edges: graph1K_A.txt, graph1K_B.txt, graph1K_C.txt
- Once your code is in good shape move on to larger graphs (10K, 100K, and 1M vertices).
- Note that some of the graphs contain no triangles at all!
- Some of the graphs are quite large (100MB+) and therefore you should not include them in your electronic submission.

How not to solve the problem

The starter code implements the following algorithm:

```
for each vertex x
        for each neighbor y of x
                for each neighbor z of y
                        if x < y < z and x is a neighbor of z then
                                output x y z
                        end if
                end for
        end for
end for
```

How to solve the problem

Step 1:

- Implement an efficient single-threaded solution (i.e., when the program runs with -ncores 1).
- **Treat this as a separate execution path in your code, and do not use any synchronization as that will cost you performance.**
- Try to improve efficiency compared to the naive implementation in the previous slide by using appropriate data structures.

How to solve the problem

Step 2:

- Once you're happy with your single-threaded implementation, **parallelize it**.
- Draw inspiration from the following (bad and good) examples of parallel word counting, which will be covered in tutorial:
  http://www.cs.umd.edu/class/fall2013/cmsc433/examples/wordcount/WordCountParallelBad.java
  http://www.cs.umd.edu/class/fall2013/cmsc433/examples/wordcount/WordCountParallel.java

How to solve the problem

Hints for step 2:

- When a program contains a loop, one way to parallelize it is to **execute different iterations of the loop in different threads**.
- This is only possible if different iterations can be executed independently, meaning that iteration i+1 does not depend on the output of iteration i.
- You decide which loop or loops to parallelize, and how to divide the iterations into separate tasks.
- Use thread-safe data structures from the java.util.concurrent package.

Assessment

- We will test your single-threaded implementation (-ncores 1) and your parallel implementation separately.
- Marks will be awarded mostly on the basis of performance (efficiency, scalability).
- Measurements of running time will exclude the cost of reading data from disk to main memory, as well as the cost of writing output to disk.
- A design document of at most 1 page in length is required.