ITCS 4145/5145 Assignment 1
Using the Seeds Pattern Programming Framework
Workpool Pattern

Author: B. Wilkinson and C. Ferner. Modification date: Jan 28, 2014

Seeds Framework

The purpose of this assignment is to become familiar with the Seeds pattern programming framework.[1] This framework was developed at UNC-Charlotte by Jeremy Villalobos as part of his PhD research into pattern programming. The framework is Java-based. The programmer first selects a particular pattern for his application. Various patterns are available including workpool, pipeline and synchronous stencil, and others can be created. We shall use the workpool pattern in this assignment as it is very general and applicable to many problems. In the workpool pattern, a master node sends out tasks to slave workers. The slaves perform computations and return results to the master, which then produces the final results. To achieve dynamic load balancing, the master keeps a queue of tasks. When slave returns a result of task, it is given another task from the queue until all talks are completed.

To use the Seeds workpool, the programmer must implement a Java interface with three principal methods:

- The diffuse method – used by the master to distribute pieces of the data to the slaves.
- The compute method – used by the slaves for the actual computation
- The gather method – used by the master to gather the results

An additional "data count" method is used to tell the framework how many pieces of data will be computed. The programmer might implement a few other methods depending upon the application, notably an initialization method and a method to compute the final result. No message passing routines are needed by the programmer - the diffuse method will create a DataMap object with data to be passed to the slaves, and similarly the compute method will create a DataMap object with data to the passed back to the master. The framework takes care of the message passing and self deploys on a local computer, a cluster, or a geographically distributed Grid platform when the application is launched. Deployment is done using a second "bootstrapping" class with a main method. This class is mostly written for each pattern and the programmer simply fills in site-specific details (paths, etc.) prior to running.

There are three versions of the Java-based Seeds framework currently implemented:

- Full JXTA P2P networking version suitable for a fully distributed network of computers and requiring an Internet connection even in just running on a single computer.
- A simplified JXTA P2P version called the "NoNetwork" version for running on a single computer and not requiring an Internet connection but otherwise similar to the full JXTA P2P version.
- Multicore version implemented with threads for more efficient execution on a single multicore computer or shared memory multiprocessor. It does not require an Internet connection.

The two JXTA versions can use the same application module source code and bootstrap code, and run in the same fashion with similar logging output. The multicore version also uses the same application

[1] Originally the framework was called Parallel Grid Application Framework (pgaf) - "Seeds" comes from the mechanism of "seeding" computers with the framework folder during self-deployment.

module source code but the bootstrap code is slightly different. In each version of the framework, only one Seeds library is different - seeds.jar, seedsNoNetwork.jar, and seedsMulticore.jar.

For this assignment, we will use the multicore version of the framework on a single PC. An Internet connection is not need to execute Seeds although it is needed to download the software. The Eclipse IDE will be used to provide tools to detect for coding errors. The assignment can be done on your own computer (recommended) or a lab computer, Windows, Mac, or Linux.

Sample Code

Fully working sample programs are provided on the course home page for:

Monte Carlo pi ("PiApprox"")
MatrixAddition ("MatrixAdd")
Matrix Multiplication ("MatrixMult")
Numerical Integration ("NumIntegration")

compressed into two zip files: ProjectSource.zip (network and no network versions) ProjectSourceMulticore.zip (multicore version) with the required directory structure (src.edu.uncc.grid.example.workpool….)

Project Contents:

- PiApprox:
  - MonteCarloPiModule.java
  - RunMonteCarloPi.java
- MatrixAdd:
  - MatrixAddModule.java
  - RunMatrixAddModule.java
- MatrixMult:
  - MatrixMultModule.java
  - RunMatrixMultModule.java
- NumIntegration:
  - NumericalIntegrationModule.java
  - RunNumericalIntegrationModule.java
- SeedsTemplate:
  - TemplateModule.java
  - RunTemplateModule.java

PART 1 Install Software and Execute Sample Code (70%)

Task 1: Install Java

First you need Java JDK.[2] To determine whether you have Java and if so, its version, type:

java –d64 –version

at the command line.[3] **–d64** will establish whether you are running 32-bit Java or a 64 bit Java.

[2] The assignment has been tested with JDK 1.7.0 on Windows XP and 7 computers, and JDK 1.6.0 on Mac OS X 10.6.8.

If you do not already have Java JDK installed, obtain it from:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

32-bit and 64-bit OS. You will need to determine whether you are running a 32-bit OS or a 64 bit OS.[4] A 32-bit OS will require a 32-bit Java.

Task 2: Install Eclipse

The Eclipse IDE will be used to build and run the Seeds code. If you do not already have Eclipse installed, obtain it from:

http://www.eclipse.org

Download the version of "Eclipse IDE for Java Developers" suitable for your platform and version of Java. It may take some time to download because of its size. Note the 32-bit version of Eclipse only works with 32-bit Java. The 64-bit version of Eclipse requires 64-bit OS and 64-bit Java. There is no installation wizard for Eclipse, so once you have downloaded the compressed Eclipse file, uncompress it and place the Eclipse folder in a suitable place in your file system. Eclipse can be placed anywhere for execution, but typically in C:\Program Files on a Windows XP/7 machine or /usr/local/bin on Linux. The following instructions assume a Windows system.

Eclipse is started by double clicking the Eclipse executable found in the Eclipse folder. It is convenient to create a shortcut of the Eclipse executable on the desktop.

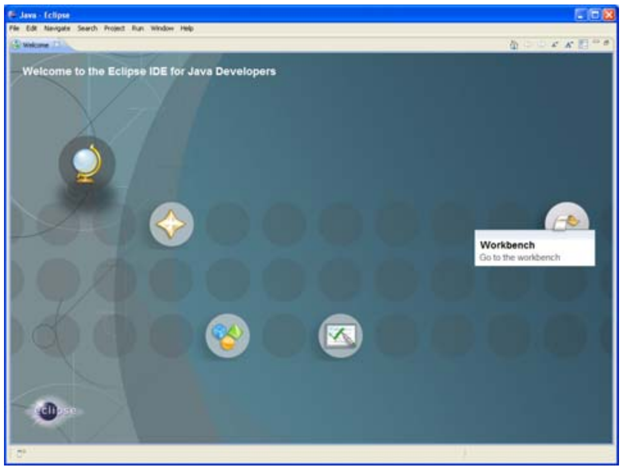

Java - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to the Eclipse IDE for Java Developers
Workbench
Go to the workbench

Figure 1 Welcome screen and workbench for Eclipse IDE for Java Developers

Starting the Eclipse executable will ask you the location for the workspace where your project code will be placed. At this point you will create a workspace for the assignment (Task 3 next). The appropriate Seeds software will also be placed there.

[3] A Windows command line console window can be started with Win key + r, enter cmd, and press Enter. On a mac, a command line console window can be started from Applications > Utilities > Terminal.
[4] Windows system: Start > right click (My) Computer > Properties. If you do not see x64 Edition, under system, it is a 32-bit OS. Mac: Apple Menu> About This Mac. All Mac OS X => 10.5 are 64 bit OS's.

Task 3 Create Eclipse Workspace

Create a folder called **Assign1** (placed anywhere you like) and an Eclipse workspace folder within this directory called "workspaceMulticore". Once we add the Seeds software and project code later, the directory structure and important files to know are given below:

```
… --->Eclipse                                    //wherever Eclipse is located
      eclipse.exe                                        //click on to start Eclipse
      …

--->Assign1
   --->workspaceMultiCore                                // used to hold Seeds projects
            --->seedsMulticore                   // appropriate Seeds framework
                                                         // (sometimes called pgaf)
                --->lib                                  // Seeds libraries directory
                Availableservers.txt                     // holds information of computers used

            --->PiApprox                         // Monte Carlo pi project
                --->bin> edu>uncc>grid>example>workpool>         // Class files, empty until code compiled
                --->src>edu>uncc>grid>example>workpool>          // Java source files
                                  MonteCarloPiModule.java        //Monte Carlo pi application code
                                  RunMonteCarloPiModule.java     //Bootstrap class for Monte Carlo pi code

                …                                        // other projects
```

Figure 2 Software directory structure

Task 4 Adding the multicore Seed libraries

Obtain the multicore version of the Seeds framework from a link "under Assignment 1 on the course home. There is no installation wizard so once you have downloaded the compressed file, uncompress it and place it in the location shown in Figure 2.

Task 5 Adding the Monte Carlo π Code

The Monte Carlo algorithm for computing π is well known and given in many parallel programming texts including the course textbook (Wilkinson and Allen 2005]. It is a so-called embarrassingly parallel application that is particularly amenable to parallel implementation but the Monte Carlo algorithm for computing π is used here more for demonstration purposes than as a good way to compute π. (It is actually a poor way to approximate π because it converges too slowly; however, the approach can lead to more important Monte Carlo applications.) A circle is formed within a square as shown in Figure 3. The circle has unit radius and the square has sides 2 x 2. The ratio of the area of the circle to the area of the square is given by $\pi(1^2)/(2 \times 2) = \pi/4$. Points within the square are chosen randomly and a score is kept of how many points happen to lie within the circle. The fraction of points within the circle will be π/4, given a sufficient number of randomly selected samples. Typically only the first quadrant of the circle is used, i.e. points between 0 and 1. In the given code, the master process will send a different random number to each of the slaves. Each slave uses that number as the starting seed for their random number generator. The Java Random class nextDouble method returns a number uniformly distributed between 0 and 1.0 (excluding 0 and 1). Each slave then gets the next two random numbers as the coordinates of a point (x,y) using nextDouble. If the point is within the circle (i.e. $x^2 + y^2 < 1$), it increments a counter that is counting the number of points within the circle. This is repeated for 1000 points. Each slave returns its accumulated count. The gatherData method performed by the master accumulates the slave results. A separate method, getPi, executed within the bootstrap module, computes the final approximation for π using the accumulated total.

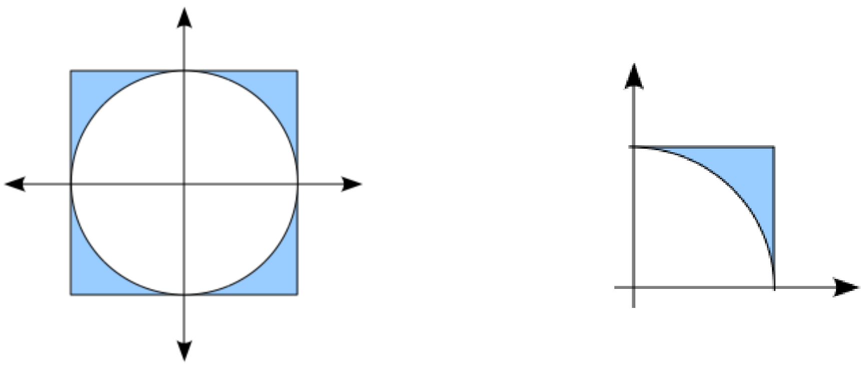


Figure 3: Unit radius circle within a 2x2 square and the upper quadrant. The ratio of the area of the circle to the area of the square is π/4. The same is true for the just the upper quadrant.

Download Program. Download the sample source code for the Monte Carlo π workpool from the provided sample source code on the course home page. Once you have downloaded the compressed file, uncompress it and place the PiApprox folder and place it in the location shown in Figure 2. There are two Java programs:

MonteCarloPiModule.java
RunMonteCarloPiModule.java

MonteCarloPiModule.java. MonteCarloPiModule.java implements the interface for the workpool is given in Figure 4:

```
package edu.uncc.grid.example.workpool;
import java.util.Random;
import java.util.logging.Level;
import edu.uncc.grid.pgaf.datamodules.Data;
import edu.uncc.grid.pgaf.datamodules.DataMap;
import edu.uncc.grid.pgaf.interfaces.basic.Workpool;
import edu.uncc.grid.pgaf.p2p.Node;

public class MonteCarloPiModule extends Workpool {
        private static final long serialVersionUID = 1L;
        private static final int DoubleDataSize = 1000;
        double total;
        int random_samples;
        Random R;
        public MonteCarloPiModule() {
                R = new Random();
        }
        public void initializeModule(String[] args) {
                total = 0;
                Node.getLog().setLevel(Level.WARNING); // reduce verbosity for logging
                random_samples = 3000; // set number of random samples
        }
        public Data Compute (Data data) {
                DataMap<String, Object> input = (DataMap<String,Object>)data; //input gets data produced by DiffuseData()
                DataMap<String, Object> output = new DataMap<String, Object>(); // output will emit partial answers by method
                Long seed = (Long) input.get("seed"); // get random seed
                Random r = new Random();
                r.setSeed(seed);
                Long inside = 0L;
                for (int i = 0; i < DoubleDataSize ; i++) {
                        double x = r.nextDouble();
                        double y = r.nextDouble();
                        double dist = x * x + y * y;
                        if (dist <= 1.0) {
```

```
                    ++inside;
                }
            }
            output.put("inside", inside);     // store partial answer to return to GatherData()
            return output;
        }
        public Data DiffuseData (int segment) {
            DataMap<String, Object> d =new DataMap<String, Object>();
            d.put("seed", R.nextLong());
            return d; // returns a random seed for each job unit
        }
        public void GatherData (int segment, Data dat) {
            DataMap<String,Object> out = (DataMap<String,Object>) dat;
            Long inside = (Long) out.get("inside");
            total += inside; // aggregate answer from all the worker nodes.
        }
        public double getPi() { // returns value of pi based on the job done by all the workers
            double pi = (total / (random_samples * DoubleDataSize)) * 4;
            return pi;
        }
        public int getDataCount() {
            return random_samples;
        }
}
```

Figure 4 MonteCarloPiModule.java

In MonteCarloPiModule.java, two important classes are imported called Data and DataMap. Data is used to pass data between the master and slaves. Data is cast into DataMap within the methods and used within the methods.[5] DiffuseData method (executed by the master) returns a random seed for each job. The Compute method (executed by slaves) picks up the random number from DiffuseData to initialize its random number generator and proceeds to choose 1000 randomly selected points counting how many are within the circle. That number is returned to the GatherData method, which accumulates all the answers from the slaves. getPi (executed in the bootstrap class) computes the final value for π.

RunMonteCarloPiModule.java. RunMonteCarloPiModule.java deploys the Seeds pattern and runs the workpool. The code for the multicore version of the framework is given in Figure 5:

```
package edu.uncc.grid.example.workpool;

import java.io.IOException;
import net.jxta.pipe.PipeID;
import edu.uncc.grid.pgaf.Anchor;
import edu.uncc.grid.pgaf.Operand;
import edu.uncc.grid.pgaf.Seeds;
import edu.uncc.grid.pgaf.p2p.Types;

public class RunMonteCarloPiModule {
        public static void main(String[] args) {
            try {
                MonteCarloPiModule pi = new MonteCarloPiModule();
                Thread id = Seeds.startPatternMulticore( new Operand( (String[])null, new Anchor( args[0],
                        Types.DataFlowRole.SINK_SOURCE), pi ), 4 );
                id.join();
                System.out.println( "The result is: " + pi.getPi() ) ;
            } catch (SecurityException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

Figure 5 RunMonteCarloPiModule.java –Thread-based multicore version

[5] The reason for two classes is for implementation convenience. For more details of Data and DataMap, see course note. DataMap extends Java HashMap.

Several classes are imported, PipeID, seeds-specific Anchor, Operand, Seeds, and Types. An instance of MonteCarloPiModule is first created. The Thread object is the thread managing the source and sink threads for the pattern. The programmer can monitor when the pattern is done computing by checking id.isAlive() or can just wait for the pattern to complete using id.join(). Args[0] should be the local host name.

The Seeds method startPatternMulticore starts the workpool pattern on the computer. It requires as a single argument an Operand object. Creating an Operand object requires three arguments. The first is a String list of argument that will be submitted to the host. The second is an Anchor object specifying the nodes that should have source and sink nodes (the master in this case) which in this provided as the string argument of main (first command line argument, args[0]). The third argument is an instance of MonteCarloPiModule previously created. As mentioned above, to run this code, we will need to provide one command line argument, the name of the local host.

Task 6 Executing Monte Carlo π Code on the multicore version of Seeds

The program can be executed on the command line or through an IDE. We choose here to use Eclipse.

Step 1. Open Eclipse

Start Eclipse and go to the workbench. Select the workspace you created called **workspaceMulticore**.

Step 2 Create a new project

Go to **File** > **New** > **Project** and select **Java Project**, or **File** > **New** > **Java Project**. Provide a name for the project, say PiApprox. If you click **Finish** at this time, you will see a new project created on the left panel as shown on the right image of Figure 6.

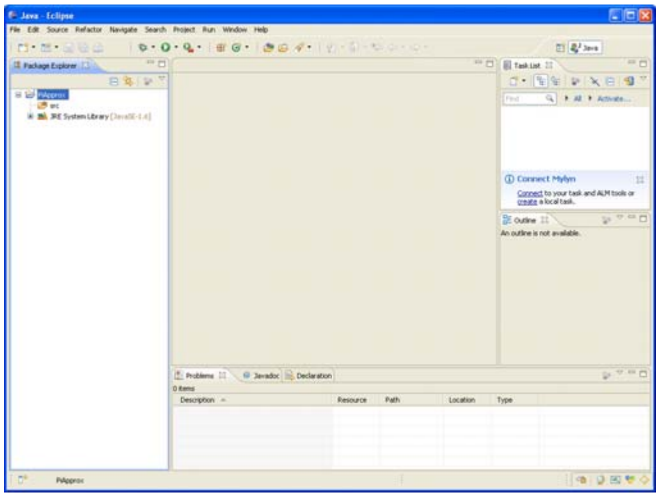
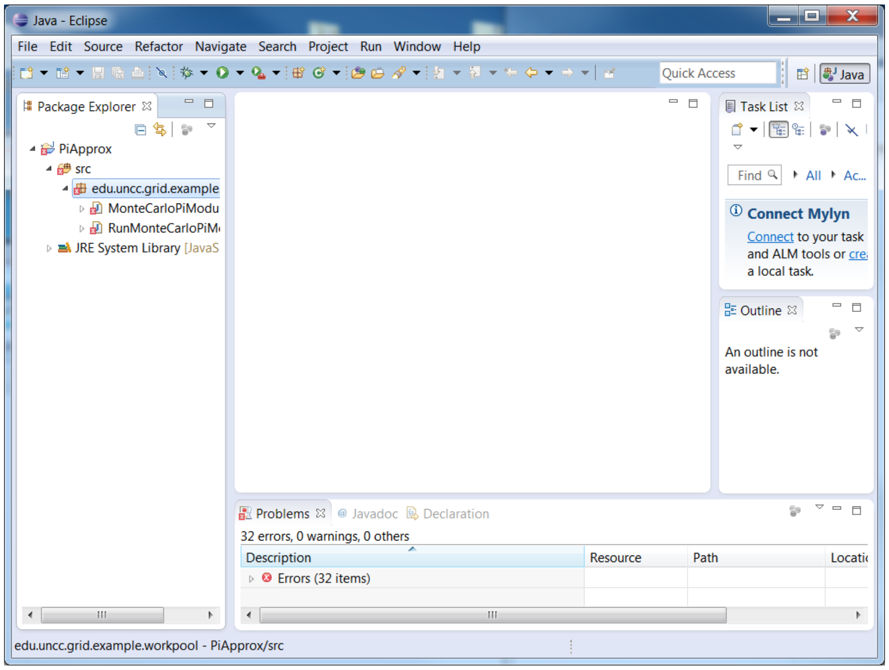


Figure 6 Creating the PiApprox Java project

Step 3 Add source code

At this point you should already see the two source files **MonteCarloPiModule.java** and **RunMonteCarloPiModule.java** within **src/edu/uncc/grid/example/workpool**, as shown in Figure 7. Select **File > Refresh** if needed. If you do not see the src files, you will need to add them, by for example dragging and dropping the source file directory into the project navigator window replacing the existing src directory. In any event, you will have unresolved errors as we have not yet added the paths to the libraries.



```
PiApprox
  src
    edu.uncc.grid.example
      MonteCarloPiModu
      RunMonteCarloPiM
  JRE System Library [JavaS
```

Connect Mylyn
Connect to your task and ALM tools or cre a local task.

An outline is not available.

32 errors, 0 warnings, 0 others

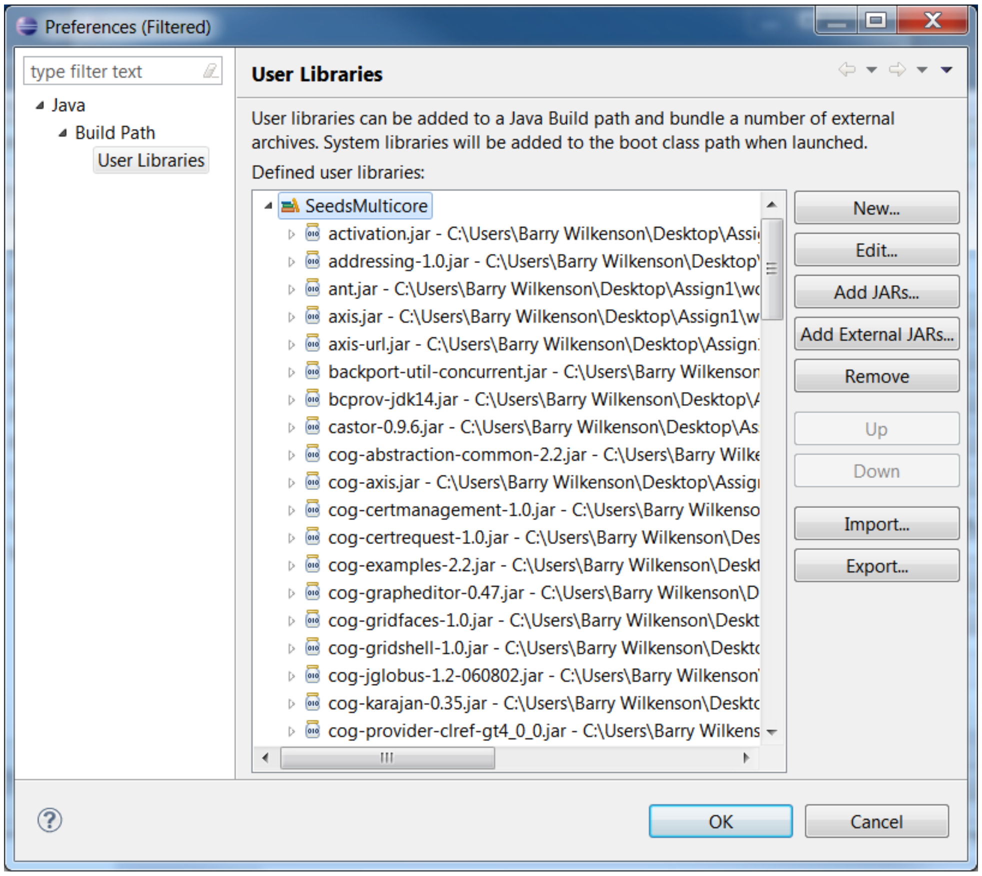
Description	Resource	Path	Locati
Errors (32 items)			

edu.uncc.grid.example.workpool - PiApprox/src

Figure 7 Source files

Step 4 Add build path to Seeds libraries

Right click the **PiApprox** folder and select **Properties**. Select **Java Build Path** > **Libraries tab** > **Add library** > **User Library** and **Next**. Since you have not yet provided any named user libraries you will not see any user libraries. Select **User Library** again and **New** and provide a name for the libraries, in this case say “SeedsMulticore” and click **OK**. Click on **Add External Jars** and navigate through your file system to the **Assign1/workspaceMulticore/seedsMulticore/lib** folder. Select all the jars inside lib (**control-A**). Click **Open**. Finally you should see something like Figure 8. Click **OK**, and get back to the workbench (**Finish** > **OK**). At this point, all unresolved references should vanish.


Preferences (Filtered)

type filter text

Java

Build Path

User Libraries

User Libraries

User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

```
SeedsMulticore
  activation.jar - C:\Users\Barry Wilkenson\Desktop\Assi
  addressing-1.0.jar - C:\Users\Barry Wilkenson\Desktop\
  ant.jar - C:\Users\Barry Wilkenson\Desktop\Assign1\wc
  axis.jar - C:\Users\Barry Wilkenson\Desktop\Assign1\w
  axis-url.jar - C:\Users\Barry Wilkenson\Desktop\Assign
  backport-util-concurrent.jar - C:\Users\Barry Wilkenson
  bcprov-jdk14.jar - C:\Users\Barry Wilkenson\Desktop\A
  castor-0.9.6.jar - C:\Users\Barry Wilkenson\Desktop\As
  cog-abstraction-common-2.2.jar - C:\Users\Barry Wilke
  cog-axis.jar - C:\Users\Barry Wilkenson\Desktop\Assign
  cog-certmanagement-1.0.jar - C:\Users\Barry Wilkenso
  cog-certrequest-1.0.jar - C:\Users\Barry Wilkenson\Des
  cog-examples-2.2.jar - C:\Users\Barry Wilkenson\Deskt
  cog-grapheditor-0.47.jar - C:\Users\Barry Wilkenson\D
  cog-gridfaces-1.0.jar - C:\Users\Barry Wilkenson\Deskt
  cog-gridshell-1.0.jar - C:\Users\Barry Wilkenson\Deskto
  cog-jglobus-1.2-060802.jar - C:\Users\Barry Wilkenson
  cog-karajan-0.35.jar - C:\Users\Barry Wilkenson\Deskto
  cog-provider-clref-gt4_0_0.jar - C:\Users\Barry Wilkens
```

Figure 8 Seeds libraries in build path

For subsequent Seeds projects, you will be able to simply select the named Seeds libraries. Note each of the three versions of Seeds have different Seeds libraries and should be named differently.

Step 5 Command Line Arguments

Before you can run the program, you will need to provide a command line argument that is read by the bootstrap class **RunMonteCarloPiModule**. Go to **Run** > **Run Configurations** > **Java Application** (Figure 9).

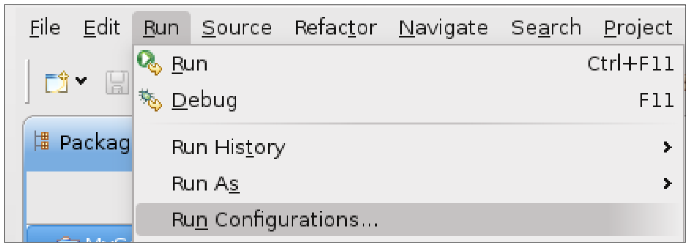


Figure 9 Run configurations

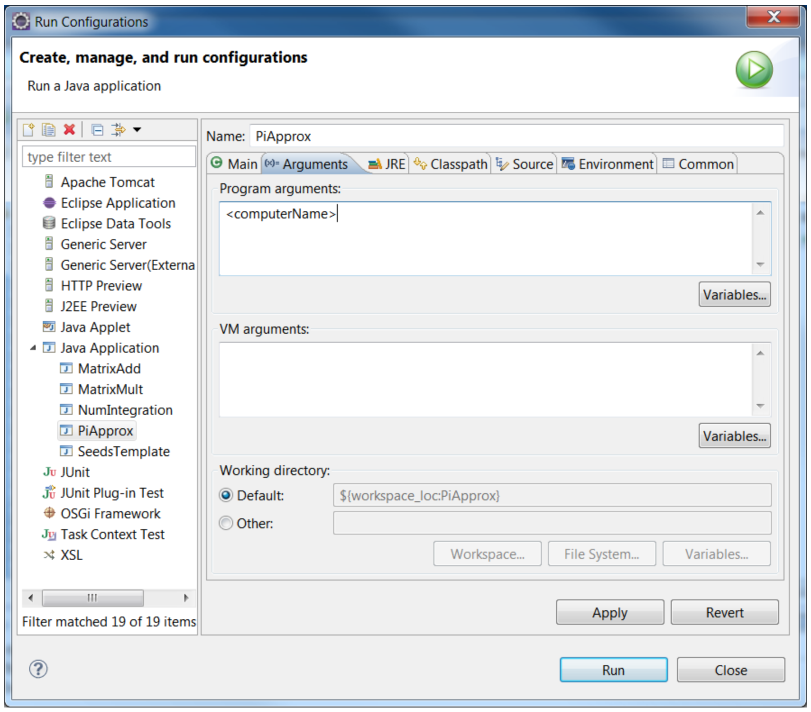
A Java Application configuration called **RunMonteCarloPiModule** should already be present. If so, rename it to **PiApprox**, otherwise create a named configuration by clicking on the leftmost icon for **New**

launch configuration at the top left of **Run Configurations** to create a new configuration named **PiApprox**. Select the **PiApprox** configuration.

Main class. In the main tab, confirm the main class is **edu.uncc.grid.example.workpool.RunMonteCarloPiModule**, otherwise set the main class to that.

Arguments. Click the tab named **(x)=Arguments.** For the multicore version of Seeds, the bootstrap class is written to accept one argument, the name of your computer. The name of your computer can be found by typing hostname on the command line. Enter this name as the program argument (shown as <computerName> in Figure 10).

IMPORTANT. Do NOT use the computer name that you will see from "View system information" or similar, which can have additional characters added to the name. You MUST use the name returned by the hostname command.



Name: PiApprox

Program arguments:

<computerName>

VM arguments:

Working directory:

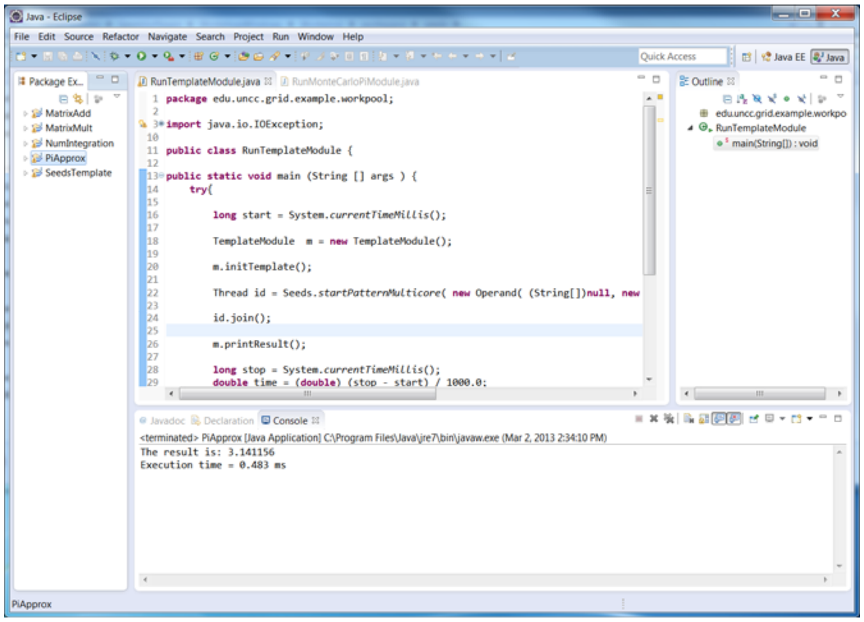
Default: ${workspace_loc:PiApprox}

Other:

Figure 10 Program command line argument

Step 6 Run program

Click "Run" to run the project. You should see the project run immediately with output in the console window (Figure 11).



```
package edu.uncc.grid.example.workpool;

import java.io.IOException;

public class RunTemplateModule {

public static void main (String [] args ) {
    try{

        long start = System.currentTimeMillis();

        TemplateModule  m = new TemplateModule();

        m.initTemplate();

        Thread id = Seeds.startPatternMulticore( new Operand( (String[])null, new

        id.join();

        m.printResult();

        long stop = System.currentTimeMillis();
        double time = (double) (stop - start) / 1000.0;
```

```
<terminated> PiApprox [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 2, 2013 2:34:10 PM)
The result is: 3.141156
Execution time = 0.483 ms
```

Figure 11 PiApprox output

How many random numbers were tried by the π approximation program?

Issues running program: If you do not get the expected output, see posted FAQs on the course home page for known issues.

Task 7 Correcting a Flaw in Monte Carlo π Code

There is a flaw in the Monte Carlo π code. Although it produces the correct answer, the use of a random number to start each random number sequence in each slave using the same random function causes each sequence to be interrelated. Modify the code to fix this problem and execute the code. Provide the code, a full explanation, and results in your write up.

Task 8 Matrix Addition

Sample code is given for matrix addition. Rather than each slave adding single elements of the matrices, each slave adds one row of **A** with one row of **B** to create one row of **C**, as shown in Figure 12:

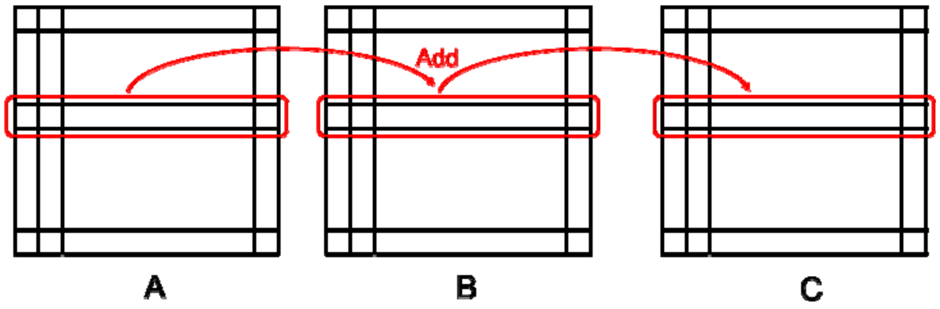


Figure 12 Matrix addition with rows added together

The workpool is shown in Figure 13:

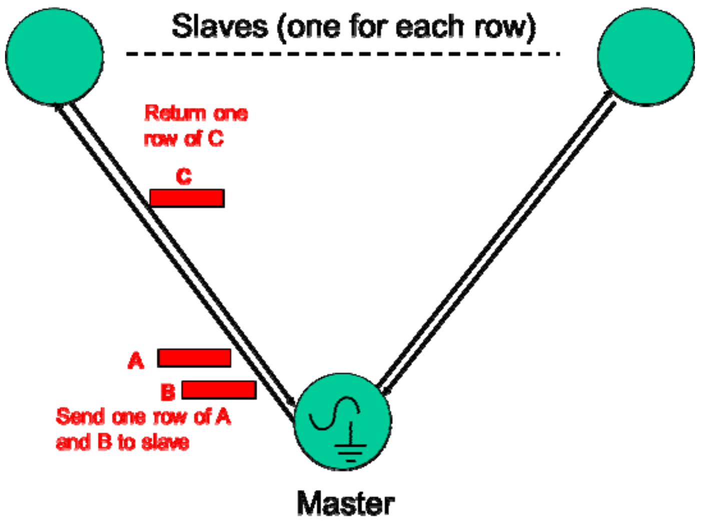


Figure 13 Workpool for matric addition

The program is written for 3 x 3 matrices and 3 slaves.

Repeat the process you used for running PiApprox to execute the matrix addition sample programs and report on the results. Test the code with the following matrices:

Matrix A
1 2 3
4 5 6
7 8 9

Matrix B
9 8 7
6 5 4
3 2 1

Make sure you show the numeric results that you get in your write up as a screenshot.

Task 9 Multiplication

Sample code is given for matrix multiplication. In this case each slave computes one result element as shown in Figure 14

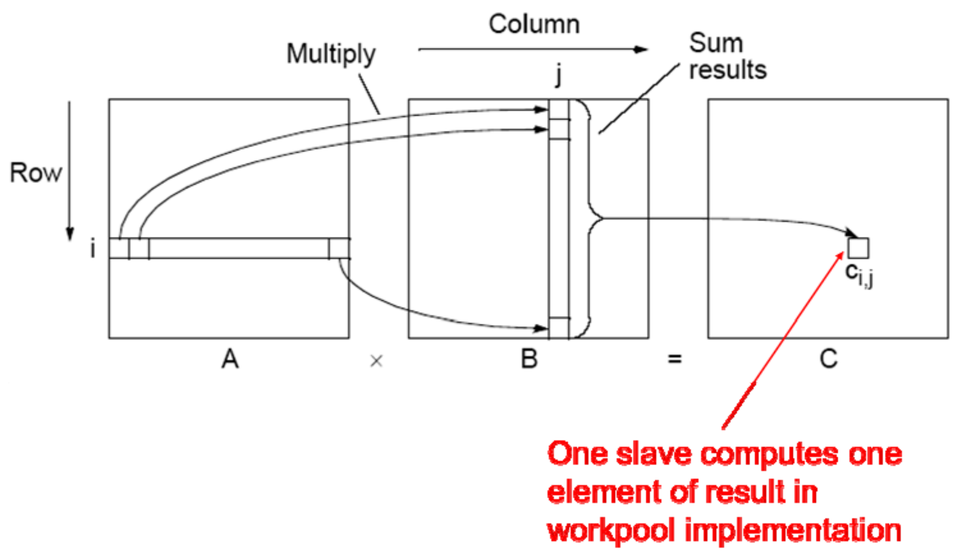

Multiply
Column
j
Sum results
Row
i
$c_{i,j}$
A
×
B
=
C
One slave computes one element of result in workpool implementation

Figure 14 Matrix multiplication

The workpool is shown in Figure 15:

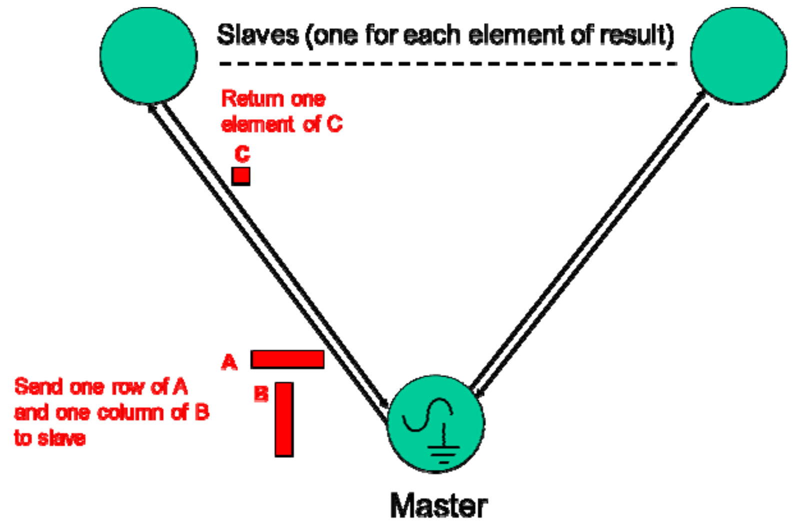

Slaves (one for each element of result)
Return one element of C
C
A
B
Send one row of A and one column of B to slave
Master

Figure 15 Matrix multiplication workpool

The program is written for 3 x 3 matrices and 9 slaves.

Execute the matrix multiplication sample programs and report on the results. Test the code with the following matrices:

```
Matrix A          Matrix B
 1 2 3             9 8 7
 4 5 6             6 5 4
 7 8 9             3 2 1
```

Make sure you show the numeric results that you get in your write up as a screenshot.

PART 2 Writing Your Own Code (30%)

Modify the sample matrix multiplication program to multiple two *N* x *N* matrices using the *block matrix multiplication algorithm* shown in Figure 16:

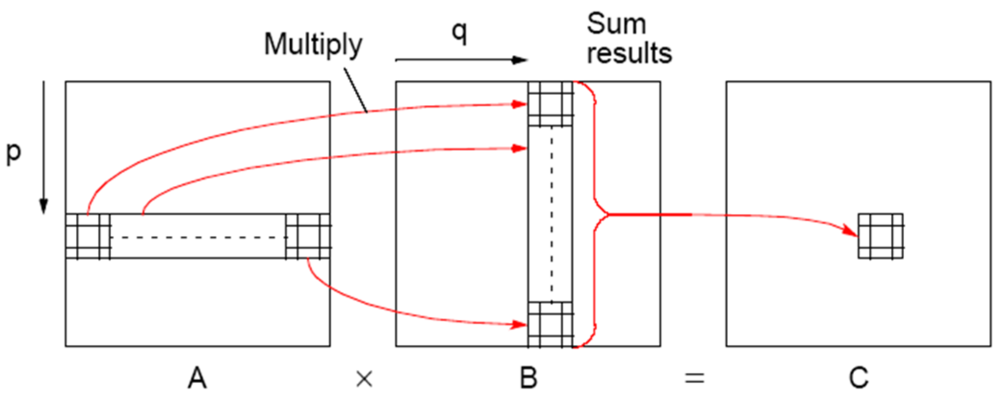


Figure 16 Block matrix multiplication

Each slave is given ***s*** rows and ***s*** columns to produce an ***s*** x ***s*** sub matrix answer. Choose *N* = 8 and *s* = 2. With *s* = 2, 16 slaves are needed.

Test your program with the following 8 x 8 matrices:

```
           Matrix A                               Matrix B
  1  2  3  4  5  6  7  8             64 63 62 61 60 58 58 57
  9 10 11 12 13 14 15 16             56 55 54 53 52 51 50 49
 17 18 19 20 21 22 23 24             48 47 46 45 44 43 42 41
 25 26 27 28 29 30 31 32             40 39 38 37 36 35 34 33
 33 34 35 36 37 38 39 40             32 31 30 29 28 27 26 25
 41 42 43 44 45 46 47 48             24 23 22 21 20 19 18 17
 49 50 51 52 53 54 55 56             16 15 14 13 12 11 10  9
 57 58 59 60 61 62 63 64              8  7  6  5  4  3  2  1
```

Make sure you show the numeric results that you get in your write up as a screenshot.

Graduate student (5% Extra credit for undergraduate students): Test you program with the two matrices given on the course home page under Assignment 1. Use 16 slaves. Compare your results with the posted results. You will need to write code to read the files.

SUBMISSION

Submit **ONE PDF** file on the UNC-C Moodle-2 by the due date as given on the course home page. Other formats not accepted. Your submission document should include *but is not limited to* the following:

1) Whether you are a graduate or undergraduate student;
2) Screenshot of the running Monte Carlo π program and the output.

3) How you corrected the flaw in the Monte Carlo π code, a listing your code *(not as screen shots)* with an explanation, and the execution results as a screen shot.
4) Screenshot of the running matrix addition program and the output.
5) Screenshot of the running matrix multiplication program and the output.
6) Code for the block matrix multiplication program and the output.

Your submission document should include insightful conclusions.

Every part and task specified will be allocated a score so make sure you ***clearly identify*** each part and task you did.

APPENDIX- JXTA P2P VERSION OF SEEDS FRAMEWORK

The following is provided if you want to try the JXTA P2P versions version of Seeds. There are two JXTA P2P versions of Seeds. The full network version of Seeds requires an Internet connection. The “NoNetwork” JXTA P2P version is a similar JXTA P2P implementation but runs on a single computer without an Internet connection. Obviously if you do not have an Internet connection, use the “NoNetwork” version.

RunMonteCarloPiModule.java. RunMonteCarloPiModule.java deploys the Seeds pattern and runs the workpool. The code for the network version of the framework is given in Figure A1:

```
package edu.uncc.grid.example.workpool;
import java.io.IOException;
import net.jxta.pipe.PipeID;
import edu.uncc.grid.pgaf.Anchor;
import edu.uncc.grid.pgaf.Operand;
import edu.uncc.grid.pgaf.Seeds;
import edu.uncc.grid.pgaf.p2p.Types;

public class RunMonteCarloPiModule {

    public static void main(String[] args) {
        try {
            MonteCarloPiModule pi = new MonteCarloPiModule();
            Seeds.start( args[0] , false);

            PipeID id = Seeds.startPattern(
                new Operand( (String[])null, new Anchor( args[1]  ,  Types.DataFlowRoll.SINK_SOURCE), pi ) );
            System.out.println(id.toString() );
            Seeds.waitOnPattern(id);
            System.out.println( "The result is: " + pi.getPi() ) ;

            Seeds.stop();

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure A1 RunMonteCarloPiModule.java – Network version

The code is quite similar to the multicore version except now Seeds is started and deployed on the list servers using the Seeds method start, which takes as its first argument the path to the seeds folder on the local computer. In the code given, the path is provided as the string argument of main (first command line argument, args[0]). The Seeds method startPattern starts the workpool pattern on the computers. It requires as a single argument an Operand object. Creating an Operand object requires three arguments. The first is a String list of argument that will be submitted to the remote hosts. The second is an Anchor object specifying the nodes that should have source and sink nodes (the master in this case) which in this provided as the string argument of main (second command line argument, args[1]). The third argument is an instance of MonteCarloPiModule previously created. The Seeds method waitOn Pattern waits for the pattern to complete, after which the results are obtained using the getPi method in MonteCarloPiModule.java. Seeds is stopped using the method stop.

As mentioned, to run this code, we will need to provide two command line arguments, the local path to the Seeds folder and the name of the local host. Both could have been hardcoded.

It is now necessary to specify the servers, even though in this case we will only use a single computer.

Specifying the computers to use

The **AvailableServers.txt** file found inside the **seeds** folder within the workspace folder needs to hold the name of the computers being used and other information can be included. For this session, we will only use a local computer and just need to provide its name of the computer. Lines starting with a # are commented out lines. Modify the one uncommented line:

```
<computerName> local - - - 1 10 GridTwo
```

replacing `<computerName>` (or whatever name is there) with the name of your computer and set the number of processors from 1 to however many processors you have (normally just one) and set the number of cores from 10 to the number of cores in each processor on your computer. The name of your computer can be found by typing hostname on the command line. **Do NOT use the computer name that you will see from "View system information" or similar, which can have additional characters added to the name. You MUST use the name returned by the hostname command.**

Executing the Monte Carlo π program.

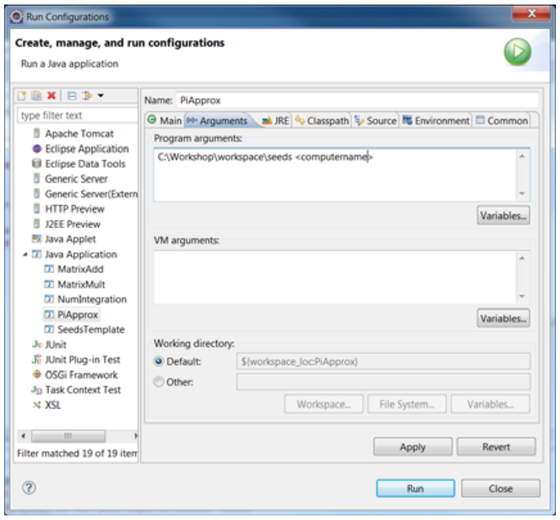
Steps 1-4 Open Eclipse. Create a new project. Add source code. Add build path to Seeds libraries Follow the same steps as previously to create a PiApprox project but for the JXTA P2P version of Seeds. Note the Internet version of Seeds has different libraries to the multicore version and should be given unique name, say Seeds for the network version or SeedsNoNetwork for the no network version.

Step 5 Command Line Arguments

For the full network version and "NoNetwork" version of Seeds, the bootstrap class is written to accept two arguments:

> 1st argument: Path to where AvailableServers.txt is located
> 2nd argument: Name of your computer

Enter the two arguments (Figure A2). Include double quotes to make a string if there are one or more spaces in the path. The name of your computer should be the same as you put in AvailableServers.txt.



Run Configurations

Create, manage, and run configurations

Run a Java application

Name: PiApprox

Program arguments:

```
C:\Workshop\workspace\seeds <computername>
```

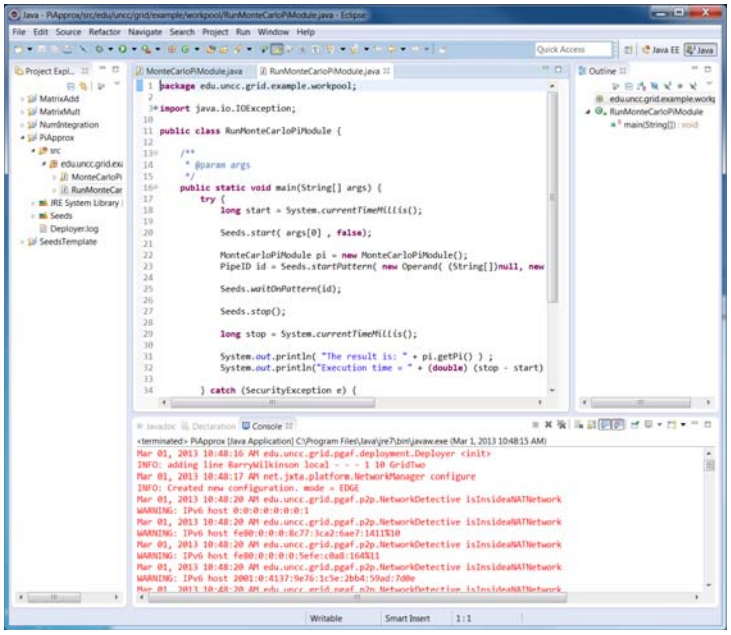
VM arguments:

Working directory:
[✓ Default:] ${workspace_loc:PiApprox}
Other:

Figure A2 Program command line arguments

Step 4 Run program

Click “Run” to run the project. The console output will begin with logging messages such as Figure A3.

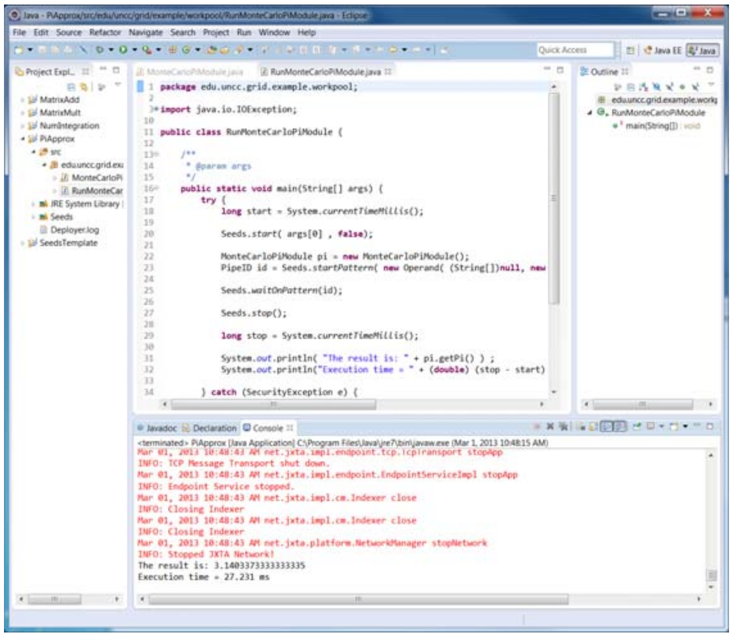


```
package edu.uncc.grid.example.workpool;

import java.io.IOException;

public class RunMonteCarloPiModule {

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            long start = System.currentTimeMillis();

            Seeds.start( args[0] , false);

            MonteCarloPiModule pi = new MonteCarloPiModule();
            PipeID id = Seeds.startPattern( new Operand( (String[])null, new

            Seeds.waitOnPattern(id);

            Seeds.stop();

            long stop = System.currentTimeMillis();

            System.out.println( "The result is: " + pi.getPi() ) ;
            System.out.println("Execution time = " + (double) (stop - start)

        } catch (SecurityException e) {
```

```
<terminated> PiApprox [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 1, 2013 10:48:15 AM)
Mar 01, 2013 10:48:16 AM edu.uncc.grid.pgaf.deployment.Deployer <init>
INFO: adding line BarryWilkinson local - - - 1 10 GridTwo
Mar 01, 2013 10:48:17 AM net.jxta.platform.NetworkManager configure
INFO: Created new configuration. mode = EDGE
Mar 01, 2013 10:48:20 AM edu.uncc.grid.pgaf.p2p.NetworkDetective isInsideaNATNetwork
WARNING: IPv6 host 0:0:0:0:0:0:0:1
Mar 01, 2013 10:48:20 AM edu.uncc.grid.pgaf.p2p.NetworkDetective isInsideaNATNetwork
WARNING: IPv6 host fe80:0:0:0:8c77:3ca2:6ae7:1411%10
Mar 01, 2013 10:48:20 AM edu.uncc.grid.pgaf.p2p.NetworkDetective isInsideaNATNetwork
WARNING: IPv6 host fe80:0:0:0:0:5efe:c0a8:164%11
Mar 01, 2013 10:48:20 AM edu.uncc.grid.pgaf.p2p.NetworkDetective isInsideaNATNetwork
WARNING: IPv6 host 2001:0:4137:9e76:1c5e:2bb4:59ad:7d0e
Mar 01, 2013 10:48:20 AM edu.uncc.grid.pgaf.p2p.NetworkDetective isInsideaNATNetwork
```

Figure A3 Sample PiApprox logging messages (Network versions)

The final result is in black at the end of the messages as shown in Figure A4:



```
package edu.uncc.grid.example.workpool;

import java.io.IOException;

public class RunMonteCarloPiModule {

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            long start = System.currentTimeMillis();

            Seeds.start( args[0] , false);

            MonteCarloPiModule pi = new MonteCarloPiModule();
            PipeID id = Seeds.startPattern( new Operand( (String[])null, new

            Seeds.waitOnPattern(id);

            Seeds.stop();

            long stop = System.currentTimeMillis();

            System.out.println( "The result is: " + pi.getPi() ) ;
            System.out.println("Execution time = " + (double) (stop - start)

        } catch (SecurityException e) {
```

```
<terminated> PiApprox [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 1, 2013 10:48:15 AM)
Mar 01, 2013 10:48:43 AM net.jxta.impl.endpoint.tcp.TcpTransport stopApp
INFO: TCP Message Transport shut down.
Mar 01, 2013 10:48:43 AM net.jxta.impl.endpoint.EndpointServiceImpl stopApp
INFO: Endpoint Service stopped.
Mar 01, 2013 10:48:43 AM net.jxta.impl.cm.Indexer close
INFO: Closing Indexer
Mar 01, 2013 10:48:43 AM net.jxta.impl.cm.Indexer close
INFO: Closing Indexer
Mar 01, 2013 10:48:43 AM net.jxta.platform.NetworkManager stopNetwork
INFO: Stopped JXTA Network!
The result is: 3.1403373333333335
Execution time = 27.231 ms
```

Figure A4 Sample PiApprox result (Network versions)