Evaluating MASS Library

CSS 600: Independent Study

Amala Gandhi

Professor Munehiro Fukuda

University of Washington, Bothell
## Contents

**Work Summary** .................................................................................................................. 3

**Detailed Description** ........................................................................................................... 3

**Network Motif** .................................................................................................................. 3

  - **Implementation/Design** .................................................................................................. 4
  - **Performance Analysis** .......................................................................................................... 5
  - **Running the program** .......................................................................................................... 6
  - **Source Code** .................................................................................................................... 6
  - **Output Screen Shot** ............................................................................................................ 13
  - **Future Enhancements** ....................................................................................................... 13

**Flute** ..................................................................................................................................... 14

  - **Implementation/Design** .................................................................................................. 14
  - **Performance Analysis** .......................................................................................................... 14
  - **Running the program** .......................................................................................................... 15
  - **Source Code** .................................................................................................................... 15
  - **Output Screen Shot** ............................................................................................................ 25
  - **Future Enhancements** ....................................................................................................... 25

**Brain Grid** .......................................................................................................................... 25

**References** .......................................................................................................................... 26
Work Summary
For three applications, Network Motif, Flute and Brain Grid, my work was to implement the computationally heavy part, sequentially and with MASS to evaluate the performance of the MASS library. I read many resources on these applications which helped me understand their implementation and their performance limitations.

After understanding the computationally heavy part of the Network Motif and Flute, I first implemented their sequential version with Java and then, implemented them with MASS library. The following section describes the details of my work and performance of the implementations.

Detailed Description
This section introduces the three applications, describes my implementation for these applications and performance results. Also, the source code and output screen shots are added for clarity.

Network Motif
Network motif can be defined as a frequent and unique subgraph pattern in a network. To search a network motif, all the possible subgraph patterns should be identified and counted. This approach is feasible in understanding the system of biological network including transcriptional regulation network (TRN), protein-protein interaction network (PPI) and metabolic network. With the fast growth of biological networks, parallel algorithms are developed and distributed computing has been tested in cloud environments.[1]

The computationally heavy part in searching a motif from the biological network is searching all the possible subgraph patterns. To implement this there are many algorithms and search methods. One of these is the Enumerated Sub Graph (ESU) algorithm. The basic idea of this algorithm is that—starting with a vertex v from the input graph—we add only those vertices to the V(Extension) set that have two properties: Their label must be larger than that of v and they may only be neighbored to the newly added vertex w but not to a vertex already in V(Subgraph), that is, they must be in the exclusive neighborhood of ‘w’ with respect to V(Subgraph). Some more insight into the structure of ESU can be gained by the following tree structure (Fig 1). [2]
**Implementation/Design**

**Sequential implementation** is done in two ways: First, is the simple implementation in which the graph is represented as the adjacency matrix and each row is searched for vertices that form an edge with that row index. For a given motif size, the vertices are added to the motif until the required size is reached. Second approach follows the ESU algorithm. ESU algorithm is much faster but the first approach closely resembles the implementation in MASS. For performance analysis the first approach is considered.

**Input files**, for sequential as well as MASS version, are text files that list the edges of the graph. The prefix of the file indicate the number of vertices in the graph.

**Implementation with MASS** involves using the following methods of the MASS library. Each edge holds an agent and that agent looks in its row to add vertices whose place value is true.

- **Places.callAll()**: To represent the initial values of graph. Each place holds a Boolean value “isEdge” to indicate if it is an edge in the graph or not.
- **Places.callSome()**: To update the place’s value to true if it is an edge.
- **Agents.callAll()**: After the agents are initialized, each place has an agent. The agents that do not have its place value as “true” are killed.
- **Agent.migrate()**: Each agent migrates to the next place and adds the place to motif, if that place has an edge.
- **Agent.kill()**: If the agent has reached the end of the matrix size, it is killed.
Performance Analysis
Although the above implementation works, there are certain repeated motifs in the output. The performance results are shown in the following tables. With the current implementation, the MASS version could not achieve a high performance because, there were too many agents created and their migration to every place in the row consumes more time. To improve the performance with MASS, the motif search should be implemented with parent and child logic using Agent.spawn(), where the agent can spawn a child to move to the row which has an edge with its place as shown in Figure 2. Moreover the Agent.spawn() type of implementation can resemble the ESU algorithm as described above.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 2: Performance of Network Motif with MASS

<table>
<thead>
<tr>
<th>Total Vertices</th>
<th>Processes</th>
<th>Threads</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>8</td>
<td>30</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>741</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>4</td>
<td>357</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>8</td>
<td>350</td>
</tr>
</tbody>
</table>
Running the program
The input files with different number of nodes should be present in the same directory. The input file as mentioned in the program will be searched for adding the edges. All files including the compile.sh and run.sh are available in the path, “dslab/MASS/java/appl/NetwokMotif”. Other than required parameters of MASS, the number of vertices should be given as the last parameter. This parameter “nVertices” should be entered as (total number of vertices + 1). Exact details can be seen in the output screen shot.

Source Code

MASS Version

Main.java

```java
import MASS.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.ArrayList;
import java.util.Date;
public class ParallelProteinNetworkMotif {
    public static void main( String[] args ) throws Exception {
        System.out.println("Total arguments = "+ args.length);
        if ( args.length != 5 )
        {
            System.out.println(" usage: java -cp MASS.jar:jsch-0.1.44.jar:. ParallelProteinNetworkMotif userid port nProcs nThrs size ");
            System.exit( -1 );
        }
        String[] massArgs = new String[4];
        massArgs[0] = args[0]; // user name
        massArgs[1] = "**********"; // password
        massArgs[2] = "machinefile.txt"; // machine file
        massArgs[3] = args[1]; // port
        int nProcesses = Integer.parseInt( args[2] );
        int nThreads = Integer.parseInt( args[3] );
        //should be number of vertices + 1 (file prefix +1), as there may be a node ’0’
        int nVertices = Integer.parseInt( args[4] );
        // start MASS
        MASS.init( massArgs, nProcesses, nThreads );
        //create the adjacency matrix
        Places adjMatrix = new Places( 1, "AdjMatrix", ( Object )null, nVertices, nVertices );
    }
```
// initialize the all matrix elements to false
adjMatrix.callAll( AdjMatrix.init_, null );

// find the edges
File file = new File( "986vertices.txt" );
try {
    Scanner scanner = new Scanner( file );
    String tok1_Vertex, tok2_Edge;
    while ( scanner.hasNextLine() ) {
        String line = scanner.nextLine();
        // Each line in file has V and E, separated by a tab.
        StringTokenizer tokenizer = new StringTokenizer( line, "\t" );
        tok1_Vertex = tokenizer.nextToken();
        tok2_Edge = tokenizer.nextToken();
        int V = Integer.parseInt( tok1_Vertex );
        int E = Integer.parseInt( tok2_Edge );
        int[] index = { V, E };
        // update the Place V, E to true
        adjMatrix.callSome( AdjMatrix.update_, ( Object )null,
                            index[0], index[1] );
        adjMatrix.callSome( AdjMatrix.update_, ( Object )null,
                            index[1], index[0] );
    }
} catch ( FileNotFoundException e ) {
    e.printStackTrace();
}

// Distribute agents over each row of adjMatrix
// The agents fill first row then second, and so on.
Agents agents = new Agents( 2, "MotifFinder", null, adjMatrix,
                            nVertices * nVertices );

// Start the time
Date startTime = new Date();
/* In case of initiating of nVertices * nVertices agents, kill the
agents who don’t have 1 in their place */
agents.callAll( MotifFinder.initialKill_ );
agents/manageAll();

// Loop until all the agents are killed
while ( agents.nAgents() > 0 ) {
    // examine if this place has an edge
    agents.callAll( MotifFinder.find_ );
    agents/manageAll(); // move to a next place
}

// End time
Date endTime = new Date();
//Total miliseconds in finding the motif
System.out.println( "\tTime (ms): " + ( endTime.getTime( ) -
startTime.getTime( ) ) );

//Finish MASS
MASS.finish();
}

Place.java
import MASS.*;
import java.util.Vector;
public class AdjMatrix extends Place {
    public boolean isEdge;
    public AdjMatrix()
    {
        super();
    }
    public AdjMatrix( Object args )
    {
        super();
    }
    //function id
    public static final int init_ = 0;
    public static final int update_ = 1;
    public static final int collect_ = 2;

    public Object callMethod( int funcId, Object args )
    {
        switch( funcId )
        {
            case init_: return init( args );
            case update_: return update ( args );
            case collect_ : return collect ( args );
        }
        return null;
    }
    //Initialize all the places to false
    public Object init( Object args )
    {
        //System.err.print( "init" );
        isEdge = false;
        return null;
    }

    // Update the place of index[0],index[1] to true, to indicate there is an edge
    public Object update( Object args )
    {
isEdge = true;
    return null;
}

//checkEdge is used by agent to know if there is an edge at this place
public boolean checkEdge()
{
    boolean thisEdge = isEdge;
    return thisEdge;
}

Agent.java
import MASS.*;
import java.util.Vector;
public class MotifFinder extends Agent
{
    int motifSize = 3; // the size of motifs to find
    int nEdges = 0; // # edges to traverse
    private Vector motif;
    public MotifFinder()
    {
        super();
    }

    public MotifFinder( Object args )
    {
        super();
        this.motif = new Vector( 3 ); // Initialize the motif
    }

    //function id
    public static final int find_ = 0;
    public static final int goToMyRow_ = 1;
    public static final int initialKill_ = 2;

    public Object callMethod( int funcId, Object args )
    {
        switch( funcId )
        {
            case find_: return find( args );
            case goToMyRow_: return goToMyRow( args );
            case initialKill_: return initialKill( args );
        }
        return null;
    }

    public Object initialKill ( Object args )
    {
        /* This place does not have edge, so, kill the agent on that place */
        if ( !(( AdjMatrix )place ).checkEdge() )
        {
        }
```java
kill();
/* Add this place to the motif */
motif.add ( place.index[0] );
return null;
}

// Not used currently
public Object goToMyRow( Object args)
{
    int x = place.index[0];
    int y = place.index[1];
migrate( y , x );
return null;
}

// Find the motifs
public Object find( Object args)
{
    // This place has an edge
    if ( ( ( AdjMatrix )place ).checkEdge() )
    {
        /* To eliminate duplicates, only if the
column > row then add column to motif */
motif.add( place.index[1] );

        /* If it reached motif size, remove the last
vertex in motif to create different combinations
with motif.size() - 1, like 1-2-3, 1-2-4, 1-2-5 */
if ( motif.size() == motifSize )
{
    for ( int i = 0 ; i < motif.size() ; i++ )
    {
        System.err.print( motif.get(i) + "-" );
    }
System.err.println();
motif.remove( 2 );
}

    // Now move to the right place
migrate( place.index[0] , place.index[1] + 1 );
    // If it’s the end border, kill the agent
if ( place.index[1] == place.size[1] - 1 )
{
    kill( );
}
}

/* Agent bumped into the rightmost column place
on the current row, so terminate it */
else if ( place.index[1] == place.size[1] - 1 )
{
    kill( );
}
```

/* Agent himself move to a next column place. */
else {
    migrate( place.index[0] , place.index[1] + 1 );
}
return null;

---

**Sequential Version**

/* Approach 1: This approach is used by MASS to find the network motif, currently this approach is considered to measure the performance */
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.StringTokenizer;
import java.util.ArrayList;
import java.util.Date;
public class NewProteinNetworkMotif {

    public static int filePrefix = 101; // total vertices + 1
    public static void main(String[] args) {
        int totalVertices = 1000; // greater than highest node
        //adjacency matrix
        boolean[][] adjMatrix = new boolean[totalVertices][totalVertices];

        //Initialize the adjacency matrix to false
        for (int i = 0; i < totalVertices; i++) {
            for (int j = 0; j < totalVertices; j++) {
                adjMatrix[i][j] = false;
            }
        }

        // Add the Edge to the Adjacency Matrix. Get V and E from input file.
        File file = new File("100vertices.txt");
        try {
            //Reading the file line by line by scanner
            Scanner scanner = new Scanner(file);
            String tok1_Vertex, tok2_Edge;
            while (scanner.hasNextLine())
            {
                String line = scanner.nextLine();
                // Each line in input file has V and E, separated by a space.
StringTokenizer tokenizer = new StringTokenizer(line, "\t");
tok1_vertex = tokenizer.nextToken();
System.out.println(tok1_vertex + "...");
tok2_edge = tokenizer.nextToken();
System.out.print(tok2_edge + ", ");

// Add edge by setting matrix element to "true"
int V = Integer.parseInt(tok1_vertex);
int E = Integer.parseInt(tok2_edge);
adjMatrix[V][E] = true;
adjMatrix[E][V] = true;
}
}
catch (FileNotFoundException e) {
    e.printStackTrace();
}

// Start Time
Date startTime = new Date();
// store the subgraph - possible motif
ArrayList<Integer> motif = new ArrayList();
ArrayList<Integer> tmpMotif = new ArrayList();
int motifSize = 3; // required motif size

// Loop for all the rows, Every row is the first node of motif
// so add the row to the motif
for (int row = 0; row < filePrefix; row++)
{
    motif.add(row);
tmpMotif = motif;

    // Add the column with represents the edge with the motif
    for (int col = 0; col < filePrefix; col++)
    {
        // check if there is an edge.
        if (adjMatrix[row][col])
        {
            motif.add(col);
            if (motif.size() == motifSize - 1)
            {
                for (int j = col + 1; j < filePrefix; j++)
                {
                    if (adjMatrix[row][j])
                    {
                        motif.add(j);
                        // Print the motif
                        for (int i = 0; i < motif.size(); i++)
                        {
                            System.out.print(motif.get(i) + "-" );
                        }
                        System.out.println();
                    }
                }
            }
        }
    }
}
motif.remove( motifSize - 1 );
}
}
motif.clear();
// Now find motifs from the next row
motif.add(row);
}
}
motif.clear();

// End Time
Date endTime = new Date();
System.out.println("\tTime (ms): " + (endTime.getTime() - startTime.getTime()));
}

Output Screen Shot

Figure 3 MASS Implementation output of Network Motifs for 9 vertices

Future Enhancements
Currently the user can input only a definite size of motif, which is a limitation, but the enhanced version may include finding the patterns with different sizes. Also, the current version is tested with only one computing node, but it may be implemented over different
computing nodes to improve performance. One the sub patterns with all the sizes are listed, pattern matching should be done to identify the overlapping, non-overlapping and different types of motif. Then each motif may be given a label and compared with different p and z score values [1]. In reality the biological network may have greater than 50000 nodes which should be tested to find the actual performance of the project.

**Flute**

Flute is a publicly available stochastic influenza epidemic simulation model. “It synthesizes populations, each corresponding to the year-2000 Census tract that is evenly subdivided into communities of 500-3000 individuals. One to seven individuals form a household where influenza is transmitted most often. Individuals are also categorized into preschool-age/school-age children and non-working/working adults. Workers are divided into non-migrant and migrant workers, the latter of whom travel across communities. After populating communities, a simulation repeats a day-based function calls: `day()`, `night()`, and `response()` each calculating daytime susceptibility and infection for each person, transmitting infection among family, and responding to epidemic (such as vaccination). After each call of these functions( ), the simulation also invokes `sync()` that exchanges migrant workers among communities. Since an unmitigated epidemic simulation with 10 million people requires 800MB memory and takes about 2 hours, FluTE has been already parallelized with OpenMP and MPICH to simulate the entire continental US with 280 million people, using 32 processors.” [3].

**Implementation/Design**

**Sequential Implementation** of FluTE, considers the computationally heavy loop as shown in the “Epimodel.cpp” file in the source code of the project [4]. For each person in the community, the program checks if the neighbors are infected. If yes, that person is infected. At day time, every neighbor within the community is checked since at day time the people of different communities interact with each other. At night time simulation the neighbors of the same community are checked for infection. As described above the `response()` method disinfects some people randomly and `sync()` method, exchanges some people between communities.

**Implementation with MASS** is done in similar manner. Places hold the total population and each place which represents a person, is infected. Following methods are used:

- Place.callSome(): To infect some people randomly
- Places.callAll(): Each time the Places Object is returned so that the neighbors can check for infection.

**Performance Analysis**

Currently the MASS version does not implement the `sync()` operation. But since I implemented the `sync()` operation in sequential version, the following tables show the results with and
without the sync() operation. When I ran the MASS version with 1 Process and 8 threads I could achieve the same performance as sequential. So, we can say that the performance can be improved by using two or more processes. Since the sequential version takes more time to execute when the sync() operation is performed, the MASS library can definitely increase the performance when implemented with sync(). Also, to improve the performance, agents can be populated over communities instead of considering places. [3].

Table 3: Flute Performance with MASS

<table>
<thead>
<tr>
<th>Communities</th>
<th>Population</th>
<th>Processes</th>
<th>Threads</th>
<th>Time (ms) (without exchange)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3000</td>
<td>1</td>
<td>1</td>
<td>481</td>
</tr>
<tr>
<td>7</td>
<td>3000</td>
<td>1</td>
<td>2</td>
<td>307</td>
</tr>
<tr>
<td>7</td>
<td>3000</td>
<td>1</td>
<td>4</td>
<td>267</td>
</tr>
</tbody>
</table>

Table 4: Sequential Flute Performance

<table>
<thead>
<tr>
<th>Communities</th>
<th>Population</th>
<th>Time (ms) (without exchange)</th>
<th>Time (ms) (with exchange) (100 migrant workers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>500</td>
<td>100</td>
<td>700</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>125</td>
<td>1123</td>
</tr>
<tr>
<td>7</td>
<td>3000</td>
<td>239</td>
<td>5190</td>
</tr>
</tbody>
</table>

Running the program
The application files including compile.sh and run.sh are available in the path,”dslab/MASS/java/appl/Epidemic Simulation”. Other than required parameters of MASS, the population (which is standard of 3000) should be given as the last parameter. Exact details can be seen in the output screen shot.

Source Code

**MASS Version**

**Main.java**

```java
import MASS.*;
import java.util.Date;
import java.util.Random;
import java.util.Vector;

public class FluteSimulation {
    public static void main( String[] args ) throws Exception {
        // verify arguments
        System.out.println("Total arguments = " + args.length);
```

if ( args.length != 5 ) {
    System.out.println("usage: java -cp MASS.jar:jsch-0.1.44.jar:.
    FluteSimulation userid port nProcs nThrs size");
    System.exit(-1);
}
String[] massArgs = new String[4];
massArgs[0] = args[0]; // user name
massArgs[1] = "**********"; // password
massArgs[2] = "machinefile.txt"; // machine file
massArgs[3] = args[1]; // port

int nProcesses = Integer.parseInt(args[2]);
int nThreads = Integer.parseInt(args[3]);
int PopulationPerCommunity = Integer.parseInt(args[4]);

// Inputs for the Simulation
int Communities = 7;
int InitialSeeds = 10;
int SimulationDays = 180;
Random rn = new Random();

// start MASS
MASS.init(massArgs, nProcesses, nThreads);

//create the adjacency matrix
Places infectedPeople = new Places(1, "InfectedPeople", (Object)null, Communities, PopulationPerCommunity);

//initialize the infected people to false
infectedPeople.callAll(InfectedPeople.init_, (Object[])null);

// Infect the initial seeds
for (int i = 0; i < InitialSeeds; i++) {
    int randomCommunity = rn.nextInt(Communities);
    int randomPerson = rn.nextInt(PopulationPerCommunity);
    int[] index = {randomCommunity, randomPerson};
    //update the Place to true to indicate its infected
    infectedPeople.callSome(InfectedPeople.initialInfection_, (Object)null, index[0], index[1]);
}

// Collect back the place to find the infected seeds
Object[] temp = new Object[Communities * PopulationPerCommunity];
Object[] TotalInfected = infectedPeople.callAll(InfectedPeople.collectInfected_, temp);

//Start the timer
Date startTime = new Date();

//Run the simulation for given number of days
for (int day = 1; day < SimulationDays + 1; day++) {
    //Infect People at Day time

TotalInfected = infectedPeople.callAll(InfectedPeople.collectInfected_, temp);
infectionPeople.callAll(InfectedPeople.day_,(Object)TotalInfected);

// Infect People at Night time
TotalInfected = infectedPeople.callAll(InfectedPeople.collectInfected_, temp);
infectionPeople.callAll(InfectedPeople.night_, (Object)TotalInfected);

// Find total infected people
TotalInfected = infectedPeople.callAll(InfectedPeople.collectInfected_, temp);

int totalInfections = 0;
for (int i = 0; i < TotalInfected.length; i++) {
    // System.out.print ( "t" + (int)(Object) TotalInfected[i] );
    if ((int)(Object) TotalInfected[i] == 1) {
        totalInfections++;
    }
}

// Vaccinate people - Response
int vaccinated = 0;
while (vaccinated < (int)(totalInfections/8)) {
    int randomCommunity = rn.nextInt(Communities);
    int randomPerson = rn.nextInt(PopulationPerCommunity);
    int[] index = {randomCommunity, randomPerson};
    infectedPeople.callSome(InfectedPeople.response_, (Object)null, index[0], index[1]);
    vaccinated++;
}

Date endTime = new Date();
System.out.println("\tTime (ms): " + (endTime.getTime() - startTime.getTime()));

// Finish MASS
MASS.finish();
import MASS.*;

public class InfectedPeople extends Place {
    int isInfected;
    int community = size[0];
    int populationPerCommunity = size[1];
    int data;

    // checking all four neighbors if infected: north, east, south, and west
    private final int north = 0, east = 1, south = 2, west = 3;
    int[] neighbour = new int[4];

    private int sizeX, sizeY;
    private int myX, myY;

    public InfectedPeople()
    {
        super();
    }

    public InfectedPeople( Object args )
    {
        super();
    }

    public static final int init_ = 0;
    public static final int initialInfection_ = 1;
    public static final int day_ = 2;
    public static final int night_ = 3;
    public static final int sync_ = 4;
    public static final int response_ = 5;
    //public static final int totalSeeds_ = 6;
    public static final int collectInfected_ = 7;

    public Object callMethod( int funcId, Object args )
    {
        switch( funcId )
        {
            case init_ : return init( args );
            case initialInfection_ : return initialInfection ( args );
            case collectInfected_ : return ( Object )collectInfected ( args );
            case day_ : return day ( args );
            case night_ : return night ( args );
            case sync_ : return sync ( args );
            case response_ : return response ( args );
            // case totalSeeds_ : return totalSeeds ( args );
                
        }
        return null;
    }

    /*Initialize all the places to false,
    to indicate that no one is infected */
    public Object init( Object args )
    {
        isInfected = 0;
        data = 0;
    }
sizeX = size[0]; sizeY = size[1]; // size is the base data members
myX = index[0]; myY = index[1]; // index is the base data members
return null;
}

/* Update the place to true, to indicate that these people are initially infected */
public Object initialInfection( Object args )
{
    isInfected = 1;
data = 1;
return null;
}

public Object day( Object args_TotalInfected )
{
    Object[] seedData = ( Object[] ) args_TotalInfected;
    int myIndex = myX * sizeY + myY;
    // Create the four neighbours
    int west = Math.max( myX - 1, 0 );
    int east = Math.min( myX + 1, sizeX - 1 );
    int south = Math.max( myY - 1, 0 );
    int north = Math.min( myY + 1, sizeY - 1 );
    neighbour[0] = west * sizeY + myY;
    neighbour[1] = east * sizeY + myY;
    neighbour[2] = myX * sizeY + south;
    neighbour[3] = myX * sizeY + north;
    // If I am not infected then check all 4 neighbour
    // even if they are in different communities
    if ( isInfected != 1 )
    {
        for ( int i = 0 ; i < 4 ; i++ )
        {
            // If neighbour is infected, then infect me
            if ( neighbour[i] == 1 )
            {
                isInfected = 1;
            }
        }
    }
return null;
}

public Object night( Object args )
{
    int west = Math.max( myX - 1, 0 );
    int east = Math.min( myX + 1, sizeX - 1 );
    neighbour[0] = west * sizeY + myY;
    neighbour[1] = east * sizeY + myY;
    // If I am not infected then check neighbour in my community
    // because at night i am with people in my community only
if ( isInfected != 1)
{
    for ( int i = 0 ; i < 2 ; i++ )
    {
        // If neighbour is infected, then infect me
        if ( neighbour[i] == 1 )
        {
            isInfected = 1;
            break;
        }
    }
    return null;
}

//Set isInfected = 1 to indicate that I am vaccinated
public Object response( Object args )
{
    boolean infectMe;
    if ( isInfected == 1 )
        infectMe = false;
    else
    {
        isInfected = 1;
        infectMe = true;
    }
    //return infectMe;
    return null;
}
/* Sequential Version of Epidemic Simulation "FluTE"*/

import java.util.ArrayList;
import java.util.Date;
import java.util.Random;

public class EpidemicInfluenzaSimulation {

    // Input Parameters
    public static int InitialSeeds = 10; // number of infected people at the beginning
    public static int NewSeeds = 10;
    public static int Communities = 7; // total communities
    public static int PopulationPerCommunity = 1000; // population per community
    public static int MigrantWorkers = 110; // migrant workers
    public static int SimulationDays = 180; // number of days
    public static double SpreadingRate = 0.06; // influenza spreading rate
    public static double ImmunityRate = 0.25; // Immunize people at this rate

    // InfectedPeople represents the whole Population
    public static boolean[][] InfectedPeople = new boolean[Communities][PopulationPerCommunity];

    public static void main(String[] args) {
        EpidemicInfluenzaSimulation fl = new EpidemicInfluenzaSimulation();
        Random rn = new Random();

        // Initialize all InfectedPerson to false, since no body is infected
        for (int i = 0; i < Communities; i++) {
            for (int j = 0; j < PopulationPerCommunity; j++) {
                InfectedPeople[i][j] = false;
            }
        }

        // Infect the initial seeds
        for (int i = 0; i < InitialSeeds; i++) {
            int randomCommunity = rn.nextInt(Communities);
            int randomPerson = rn.nextInt(PopulationPerCommunity);
            InfectedPeople[randomCommunity][randomPerson] = true;
            System.out.println("Infected Neighbour ", randomCommunity +"," + randomPerson);
        }

        // Number of infected people now changes
        NewSeeds = fl.totalSeeds();
        System.out.println("Number of People infected initially ", NewSeeds);

        // Start the timer
        Date startTime = new Date();
    }
}
// Run the simulation for given number of days, to infect people at given
date rate
for (int day = 1 ; day <= SimulationDays ; day++ )
{
  //Infect people at day time
  f1.day();
  //Infect people at night time
  f1.night();
  //Exchange the immigrants after one simulation cycle
  //f1.sync();
  //Vaccinate the people at given rate
  f1.response();
  //calculate the total seeds after each simulation cycle
  NewSeeds = f1.totalSeeds();
  System.out.println("Number of People infected after day " + day +
  "are " + NewSeeds );
}

Date endTime = new Date();
System.out.println("\tTime (ms): " + ( endTime.getTime( ) - startTime.getTime( ) ));

//Infect a person whose neighbour is infected
void InfectMyNeighbour( boolean isDay, int row , int col )
{
  boolean isNeighbourInfected = false;
  int rowStart  = Math.max( row - 1, 0   );
  int rowFinish = Math.min( row + 1, Communities - 1 );
  int colStart  = Math.max( col - 1, 0   );
  int colFinish = Math.min( col + 1, PopulationPerCommunity - 1 );
  /*Day time, so check all four neighbours because people may
interact with different communities */
  if ( isDay )
  {
    // Find the non-infected neighbour and infect him
    for ( int curRow = rowStart; curRow <= rowFinish; curRow++ )
    {
      for ( int curCol = colStart; curCol <= colFinish; curCol++ )
      {
        if ( !InfectedPeople[curRow][curCol])
        {
          InfectedPeople[curRow][curCol] = true;
          isNeighbourInfected = true;
          //System.out.println("Infected Neighbour " + curRow + "," +
          curCol);
          break;
        }
      }
    }
    //Only wanted to infect one neighbour, so break
    if ( isNeighbourInfected )
    {
      break;
    }
  }
}
/* Night time, so only check adjacent neighbours in same row, because all the people are in same community at night */
else {
    if ( !InfectedPeople[row][colStart] )
    {
        InfectedPeople[row][colStart] = true;
    }
    else if ( !InfectedPeople[row][colFinish] )
    {
        InfectedPeople[row][colFinish] = true;
    }
}

/* Total number of people infected after every simulation run*/
int totalSeeds()
{
    int count = 0;
    for ( int i = 0 ; i < Communities; i++ )
    {
        for ( int j = 0 ; j < PopulationPerCommunity ; j++)
        {
            if (InfectedPeople[i][j])
            {
                count++;
            }
        }
    }
    return count;
}

// DayTime Simulation
void day()
{
    // check every person in all communities
    for ( int i = 0 ; i < Communities ; i++)
    {
        for( int j = 0 ; j < PopulationPerCommunity; j++)
        {
            if (InfectedPeople[i][j])
            {
                /* check if my neighbour is infected
                if not, then infect him. day = true, night = false*/
                InfectMyNeighbour(true, i, j);
                j++; // jump over a neighbour
            }
        }
    }
}

// Night Time simulation
void night()
{ 
  for ( int i = 0 ; i < Communities ; i++ )
  {
    for( int j = 0 ; j < PopulationPerCommunity; j++ )
    {
      if (InfectedPeople[i][j])
      {
        /*check if my neighbour ( of my row ) is infected
         if not, then infect him. day = true, night = false*/
        InfectMyNeighbour(false , i , j);
        break;
      }
    }
  }
}

//Exchange migrant workers
void sync()
{
// temporary array to store the migrant workers
  boolean Exchange[][] = new boolean[Communities][MigrantWorkers];

  // Copy the first 100 elements of all Communities
  for ( int i = 0; i < Communities ; i++ )
  { 
    for ( int j = 0 ; j < MigrantWorkers ; j++ )
    { 
      Exchange[i][j] = InfectedPeople[i][j];
    }
  }

  // Exchange it in subsequent rows (1 to 2, 2 to 3 etc.)
  for ( int i = 0; i < Communities ; i++ )
  { 
    for ( int j = 0 ; j < MigrantWorkers ; j++ )
    { 
      if ( i == Communities - 1)
      { 
        InfectedPeople[i][j] = Exchange[0][j];
      }
      else
      { 
        InfectedPeople[i][j] = Exchange[i+1][j];
      }
    }
  }

// Give vaccine to people to immune them
void response()
{
  int totalPersonImmune = 0 ;
  Random rn = new Random();

  // Minimum of "NewSeeds/4" people should be vaccinated
  while (totalPersonImmune < (NewSeeds/8))
  { 
    int randomComunity = rn.nextInt(Communities);
    int randomPerson = rn.nextInt(PopulationPerCommunity);
  }
//If person is infected, vaccinate him
if (InfectedPeople[randomCommunity][randomPerson])
{
    InfectedPeople[randomCommunity][randomPerson] = false;
    totalPersonImmune++;
}

Future Enhancements
Since the Places Object returned from MASS could not work as expected, I could run it with multiple processes. But after performing the sync operation, the program should be checked with multiple nodes. There are many parameters which are hard coded in the program like Communities, Initial Seeds etc. In the original program there are further parameters which decide the infection rate, vaccination rate etc., which may be considered.

Brain Grid
I read many papers and source code to understand how brain grid works. I could understand the logic but I needed more time to find the heavy computationally loop since the source code is too large and needs lot of communication with team members to exactly understand it.

BrainGrid can be coded to use only MASS places to distribute a network of N ×N neurons on multiple GPUs. Each “neuron” place maintains synapses dynamically emanating from it with a vector of neighbors to be passed at run time to exchangeAll() that converges synapse input onto adjacent neurons. BrainGrid includes two communication-intensive code portions: (1) synaptic-strength computation must examines a possible connectivity overlap between all pairs of neurons and (2) each neuron may converge synapse inputs from any other neurons, both of
which requires a complete communication network among all neurons and makes multi-GPU computation difficult. In reality, an active neuron’s round-shaped connectivity has a finite radius that we should define as the range of BrainGrid’s ghost space and limit all data exchange within the range.

References