CS5500 Parallel Programming
Project Report: Pandemic Simulation

Benji Stewart, someguynamedben7@gmail.com
Connor Osborne, connorosborne@aggiemail.usu.edu

Description

We designed a system to model how a disease would travel through a population as a pandemic. We designed a city where people would travel to and from their homes to other destinations in the city. While at home or traveling around the town they people have a chance to get infected if they're in the presence of another person who is infected. This chance is increased relative to how many infected people there are inside the building they are in. After a certain period (which we decided was going to be 3 days) the people become immune to the disease. We decided not to specify whether people die from the disease or not since immunity to becoming sick or passing on the disease is irrelevant to whether that person is still alive or not. Therefore we simply have immune people who are unable to become sick or get others sick. For simplicity each building can only hold a certain number of people including the homes, and each person will leave their home for the day and travel to 3 other buildings before returning to their home. For simplicity we made each process a building or a home depending on that particular process's rank.

Effort Overview

To split up writing the code we decided to have each of us handle writing code that would either be for the even or odd processes. This worked out since we made each process either a regular building or a home, and each had different tasks it had to perform. This ended up making the code roughly the same amount. We both worked on debugging when didn't work, but Connor was able to put an extra day's work in debugging and was able to get the project working correctly. Connor wrote up the 2-page write up of the project himself as well. I (Benji) am writing up this project report myself. Connor has also decided to do the slides himself because he's awesome. We ended up spending around three days working on this project, but I'm not sure how many hours it took us.

Code

As was stated before we decided to have each process be its own building. Even processes were assigned as the homes, and odd processes were setup as the regular building such as businesses or something. Each person was stored as a vector which was used to store different states such as their health and where their home was or how many days they have been infected as we can see here.

```
// homeId, Status: 0 = isNormal 1 = isSick 2 =isImmune, stopCount, daysInfected
std::vector<int> person1 = {rank,0,0,0};
```

For the homes we had them first send each person out to a building. Then for the regular buildings we gave each a queue of people which could not exceed the maximum allowed inside each building as a time. From the queue of people the buildings would either send the people to other random buildings or send them home if the person had already visited 3 buildings.

```
if(tempPerson[2] >= 3){
    if(tempPerson[1] == 1){
        infectedCount--;
    }

    //send home
    MPI_Isend(&tempPerson[0], 4, MPI_INT, tempPerson[0], 2, MCW, &request);
}else{
    if(tempPerson[1] == 1){
        infectedCount--;
    }
    //send to another building
    tempPerson[2] += 1;
    MPI_Isend(&tempPerson[0], 4, MPI_INT, getOddRank(rank, size), 1, MCW, &request);
}
```

Both the buildings and the homes update the status of each person if they get infected, but we had only the homes update each person's status to being immune after the set amount of days. So once the homes get back all their people then the day is over and the numbers of immune, infected and non-infected people are tallied up, and the results are saved to a file.

```
fout << "Day " << numberOfDays << ": " << std::endl;
fout << "Number of non-infected people: " << totalNormal << std::endl;
fout << "Number of infected people: " << totalInfected << std::endl;
fout << "Number of immune people: " << totalImmune << std::endl;
fout << std::endl;
fout << std::endl;
```

The program then repeats for the number of days in the simulation and then exits. To calculate how likely it was that someone got infected when they were in a building with infected people we wrote a function to determine the chance that the person would become infected. Each infected person in the room increased the chances by 25%. The function would return true or false depending on if that person became infected or not.

```
bool willBeInfected(int infectedCount){
    int chance = 25 * infectedCount;

    if(rand() % 100 < chance){
        return true;
    }else{
        return false;
    }
}
```

Tests

The tests we ran were based on different variables in the simulation such as: the number of people, the number of sick people, the number of days to run the simulation, and the infection rate. Changing the number of days simply let us change the window of time we were able to observe the changes to the population. We found we had to increase the infection rate or the population would reach heard immunity too quickly. As it stands the population was able to reach heard immunity pretty quickly anyways, so more testing is likely necessary. The way our program was written the number of people was tied to the number of homes in the simulation, so increasing the number of buildings would increase the population size. This created the situation where the amount of time until heard immunity was reached was roughly the same no matter the population size because ratios were preserved.

Additional Remarks

This was a fun project especially because we had to think about how to not only get the program to work with both of us adding code at the same time and to different parts of the same program, but also that we had to think about how to coordinate the message passing to each other's section of code. Another interesting thing was to see that tweaking each individual variable for the tests could change how the virus spread. It reminded me a but of a neural network where you have to tweak hyper-parameters. That might be another non-mpi project to try is to train a neural network to learn to tweak the different parameters of the simulation. Another thing I though was interesting to see was what would happen to the population since no preventative measures were taken though, which is something I think I would want to try and implement in the future just to see how it would differ. I would also probably consider refactoring the code in the future to make parts of the logic, such as the population size, more in dependant of other factors. All in all I enjoyed this project and I'm glad we decided to do it.