Exercise 10

Random Number List (CPU)

In this task we should compute the random numbers on the CPU and copy an array to the GPU which then the threads can draw the needed random numbers from. This was achieved, by computing the maximum number of contacts inside the contacts_per_day array. With this information an array was created cuda_rand_values_threads which then contains the maximum possible amount of random numbers that could be needed for one day. Each thread indexes the array via its thread ID multiplyed by the number of max_rand_values_per_thread, and then adds according for which contact a random number is currently requested. Due to memoryy efficiency it is not practical to copmpute one large array containing all the random numbers for all days of the simulation. The reason behind this, is that most of these numbers are never needed due to the pushback that is implemented.

Random Number Generation on GPU

In this task we should create our own random number generator on the GPU which is then used to compute the needed random numbers. To achieve this three additional cuda funciton were created:

- **cuda_taus:** This function performs one Tausworthe step on the state reference and updates the state.
- **cuda_LCG:** This function performs a linear congruential generator step on the state reference and updates this state.
- **cuda_gen_rand_num:** This function combines one Tausworthe step with one LCG step to generate better random numbers. Then the result is scaled and returned.

The states used are arrays where each thread has its own state for the Tausworthe and LCG steps. These arrays are initialized with random numbers on the CPU und then updated in ech step of the random number generation process. In my implementation one Tausworthe and one LCG steps are combined, but with the given functions this could easily be expanded to use multiple steps. For this simulation after verifying the desired properties of the random numbers generated it suffices to only use these two steps and cut down on computation cost.

Quality of the Random Numbers

To verify the quality of the generated random numbers an extra program was created, using the same method to compute random numbers on the GPU and afterwards analyzed. The mean was checked and the distribution of the numbers in 0.1 intervals between zero and one, with the following results.

```
CHECKING STATISICAL QUALITY

Numbers generated: 10000000, in the interval betweene 0 and 1
checking number of values in the given intervals:

Bin 0 (Range 0 to 0.1): 1000288 10.00%
Bin 1 (Range 0.10 to 0.20): 1000191 10.00%
Bin 2 (Range 0.20 to 0.30): 999293 9.99%
Bin 3 (Range 0.30 to 0.40): 1002224 10.02%
Bin 4 (Range 0.40 to 0.50): 997971 9.98%
Bin 5 (Range 0.50 to 0.60): 998875 9.99%
Bin 6 (Range 0.60 to 0.70): 999221 9.99%
Bin 7 (Range 0.70 to 0.80): 1000107 10.00%
Bin 8 (Range 0.80 to 0.90): 1001703 10.02%
Bin 9 (Range 0.90 to 1.00): 1000127 10.00%

Mean: 0.500015
```

Port from CPU to GPU

To port the simualtion from the CPU to the GPU most of the parts in the simulation function were realized with cuda kernels. The specific implementation differs slightly depending on if the random numbers are generated on the CPU or GPU. The general approach is, that first all the necessary arrays are allcoated on the CPU and GPU. After that a cuda kernel is used to generate the initial data array. Then the main loop over the simulation days starts. In this loop a kernel is used to compute the daily number of fake news believers, which then reports the result back to the CPU. The checking if pushback is true, as well as computing th ecorresponding transmissiona and recovery probabilities on the CPU (for these calculation it makes no sense to do this on the GPU because it is only once computed for the day). After that the generated values are used in the final cuda kernel where the fakenews are passed on according to the reference implementation. Duriung the loop over the days all the data remains on the GPU only the believer count of the current day is reported back to the CPU.

Performance Model

To assess the performance of the Implementations first the different execution times for the GPU implementation of the usage of the random number list from the CPU compared to the direct computation on the GPU. In Figure 1 it can be seen, that the computation directly on the GPU has huge advantage in terms of performance compared to the usage of a precomputed list.

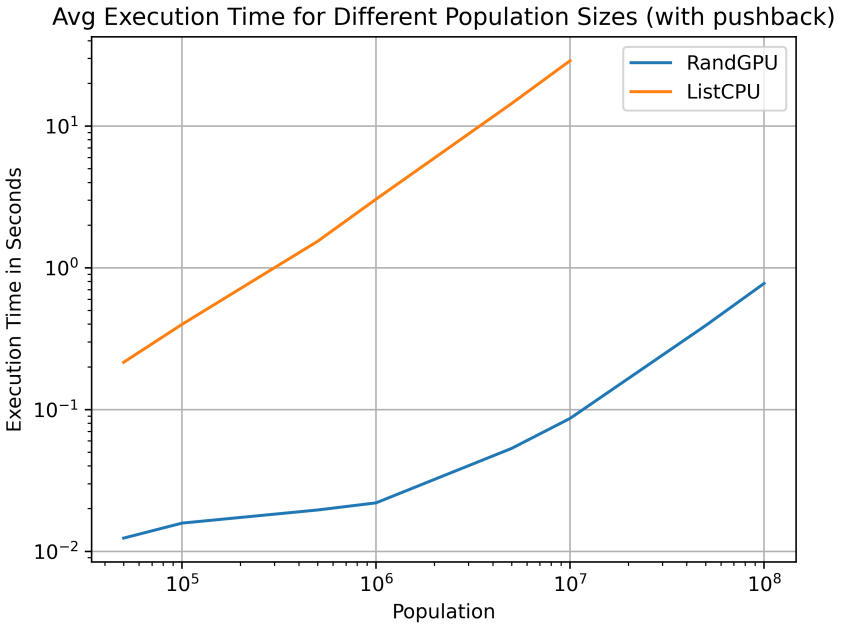

Figure 1: Execution times with pushback for different ppopulation sizes.

What i simportant to note here is, that due to the pushback the variation of the days that are simulated, does not really make a difference due to the fact that the pushback threshold will be reached always at a similar point (provided, that all the other parameters stay the same) which then kills the 'epidemic' at similar points in time. To visualize this the following figure shows the execution times for different population sizes for the program that computes the random numbers on the GPU. The only difference being that pushsback is once disabled and once enabled.

Figure 2: Execution times with pushback for different ppopulation sizes(with and withou pushback).

Another aspect that influences performance is the number of contacts each person has on each day. In the following plot the random number generator on the gPU is used to show the impact of the number of contacts of each person each day. To have a better understanding of the scaling the days were varied but pushback was disabled to get the full picture of the simulated days.

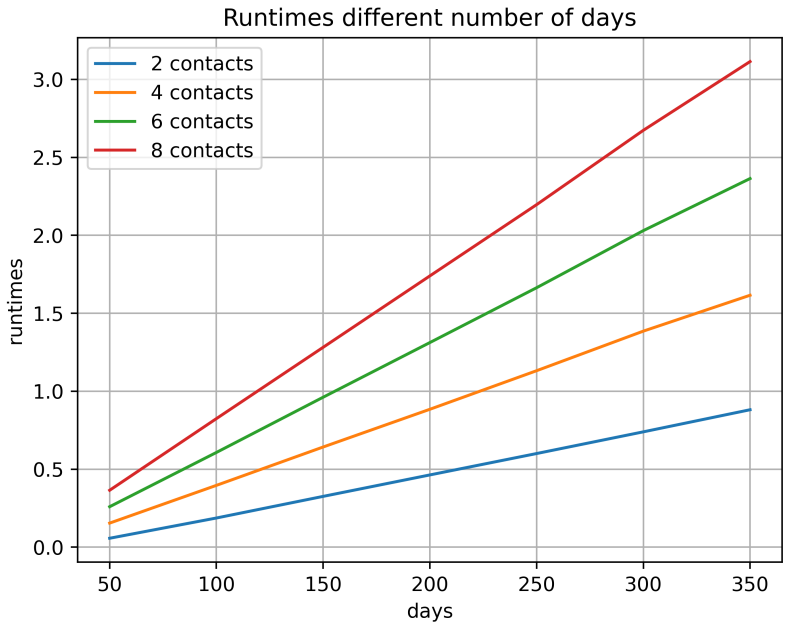

Figure 3: Execution times for different number of days for different number of contacts.