

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 multiplied by the number of `max_rand_values_per_thread`, and then adds according for which contact a random number is currently requested. Due to memory efficiency it is not practical to compute 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 function 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 and then updated in each 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.

```
1 CHECKING STATISICAL QUALITY
2
3 Numbers generated: 100000000, in the interval between 0 and 1
4 checking number of values in the given intervals:
5
6 Bin 0 (Range 0 to 0.1): 1000288 10.00%
7 Bin 1 (Range 0.10 to 0.20): 1000191 10.00%
8 Bin 2 (Range 0.20 to 0.30): 999293 9.99%
9 Bin 3 (Range 0.30 to 0.40): 1002224 10.02%
10 Bin 4 (Range 0.40 to 0.50): 997971 9.98%
11 Bin 5 (Range 0.50 to 0.60): 998875 9.99%
12 Bin 6 (Range 0.60 to 0.70): 999221 9.99%
13 Bin 7 (Range 0.70 to 0.80): 1000107 10.00%
14 Bin 8 (Range 0.80 to 0.90): 1001703 10.02%
15 Bin 9 (Range 0.90 to 1.00): 1000127 10.00%
16
17 Mean: 0.500015
```

Port from CPU to GPU

To port the simulation 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 allocated 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 the corresponding transmission and recovery probabilities is done 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. During 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.

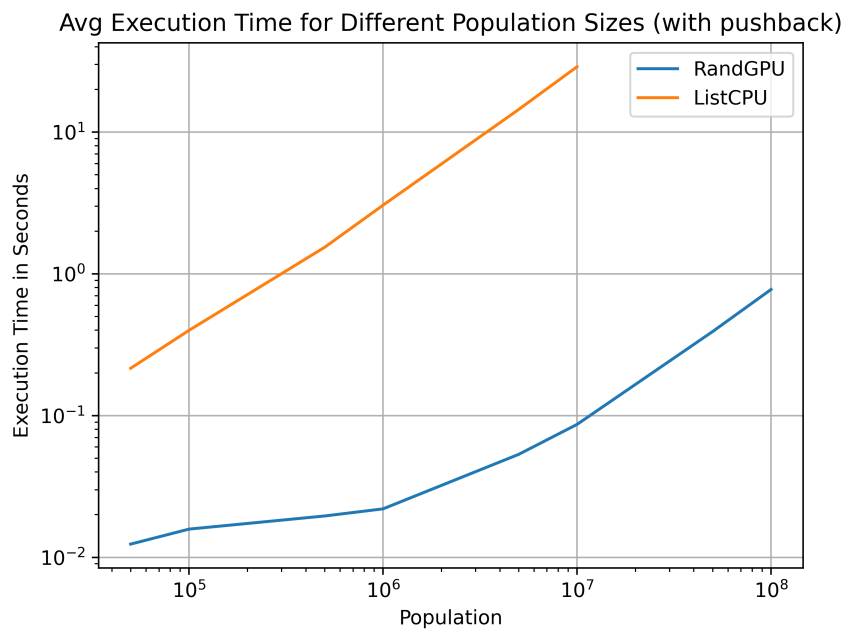


Figure 1: Execution times with pushback for different population sizes.

What is important 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 pushback is once disabled and once enabled.



Figure 2: Execution times with pushback for different population sizes(with and without 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.

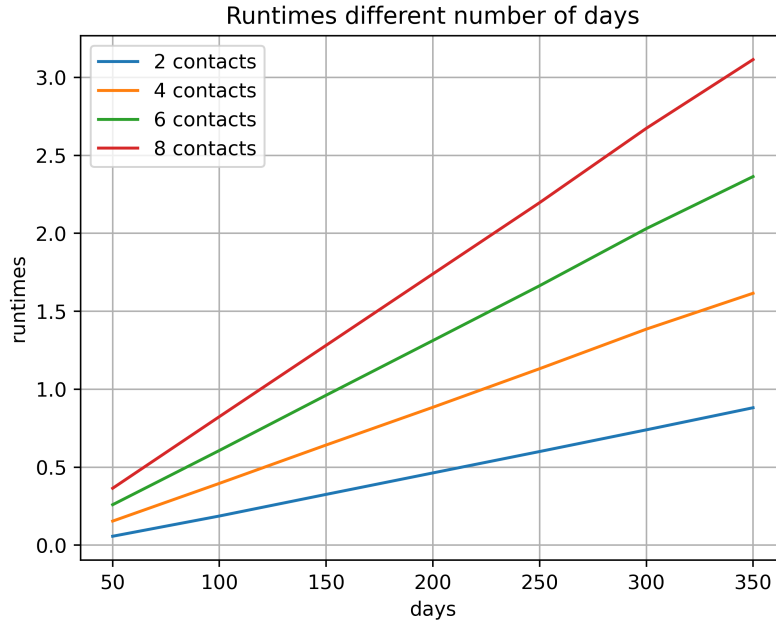


Figure 3: Execution times for different number of days for different number of contacts (no pushback).

To get an idea of the overall parameters that have the most impact on the runtime of the simulation the following parameters seem to be the most important:

- **Population Size:** This has an immediat impact on the performance in every scenario, because each cuda kernel loops over the whole population.
- **Contacts per Day:** This parameter is responsible for the size of the loop inside the population loop in the pass on fakenews kernel. The impact grows as more people become believers.
- **Days Simulated:** This is responsible for the length of the outer most loop in the simulation function, but is mostly not fully executed due to the pushback and its effect.

So a performance model for the worst case would be

$$O(\text{Population} \cdot \text{contacts} \cdot \text{days}) \quad (1)$$

This would be most accurate when there is no pushback and most of the people are already fakenews believers. In a normal execution with the pushback

parameters this limit will probably never be reached. But it gives a good idea of the worst case and the impacts the variables have on the performance of the simulation. The most influence definitely has the population size in a normal execution.