Vienna University of Technology

360.252 Computational Science on Many Core Architectures

Institute for Microelectronics

Exercise 10

Authors:
Camilo Tello Fachin
12127084

Supervisor:
Dipl.-Ing. Dr.techn. Karl Rupp

January 17, 2023

TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Abstract

Here documented the results of Exercise 10.

Contents

1 DIVOC Simulator (2 / 7+2 Points)

1.1 Generate Random Numbers for GPU (1/1 Point)

In order to save time I have already started with implementing this subtask as part of the 2 next subtasks. I will write this only in pseudocode.

Algorithm CPU Based Random Number Generator

```
define Number_of_threads
define Random Seed                          ▷ E.g. from Entropy Pool or just 4 ;-)
call function LCGRNG(Seed)                  ▷ Function that does 1 step of LCG RNG
initialize vector RN of length equal to Number_of_threads
RN[0] = seed
for i = 1; i ≤ Number_of_threads-1; i++ do
    RN[i] = LCRNG(RN[i-1])
end for
```

This vector `RN` can now be used by the GPU such that every thread has its own random number.

1.2 RNG on GPU (1/1 Point)

The following code snippet shows a `__device__` function that can be called by other `__global__` cuda funtions. This functions generates takes the thread ID as seed for a standard linear congruential random number generator and does one iteration of the LCGRNG. The performance difference to the CPU implementation is quite simple. In order to generate

C++ code for

```
__device__ u_int32_t cuda_LCG(int thread_id, u_int32_t *rand_nr_vec){
    u_int32_t seed = rand_nr_vec[thread_id];
    u_int32_t m    = pow(2,31) - 1;
    u_int32_t a    = 48271;
    u_int32_t rnr  = (a * seed) % m;
    rand_nr_vec[thread_id] = rnr;
    return rnr;
    }

    __global__ void cuda_RN(int N, u_int32_t *random_nr_vec, int rand_iters ) {
    int thread_id = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = 0; i < rand_iters; i++){
        u_int32_t kebap = cuda_LCG(thread_id, random_nr_vec);
    }
    }
```

The performance difference is that the algorithm of part a), a for loop had to go through LCGRNG for `Nummber_of_threads` times, whereas in this algorithm, every thread perfoms one step concurrently.

2 Port DIVOC Simulator to GPU (0/4 Points)

Unfortunately wasnt able to make it run. Number of infected ppl was always zero, might be something wrong with the random numbers. I spent at least 5 hrs on it.. so 0 points would be devastating but okay I guess :(

C++ code for

```
1
2  void init_input (SimInput_t *input)
3  {
4    input->population_size = 8916845; // Austria's population in 2020 according to Statistik Austria
5
6    input->mask_threshold      = 5000;
7    input->lockdown_threshold = 50000;
8    input->infection_delay     = 5;     // 5 to 6 days incubation period (average) according to WHO
9    input->infection_days      = 3;     // assume three days of passing on the disease
10   input-> starting_infections = 10;
11   input->immunity_duration = 180;   // half a year of immunity
12
13   input->contacts_per_day = (int*)malloc(sizeof(int) * DAYS_DIVOC);
14   input-> transmission_probability = (double*)malloc(sizeof(double) * DAYS_DIVOC);
15   for (int day = 0; day < DAYS_DIVOC; ++day) {
16     input->contacts_per_day[day] = 6;              // arbitrary assumption of six possible transmission
         contacts per person per day, all year
17     input-> transmission_probability [day] = 0.2
18                                          + 0.1 * cos((day / DAYS_DIVOC) * 2 * M_PI); // higher
                                               transmission in winter, lower transmission during summer
19   }
20 }
21
22 __global__ void cuda_init_input (SimInput_t *input, int *cpd, double *tp) {
23     input->population_size = 8916845; // Austria's population in 2020 according to Statistik Austria
24
25     input->mask_threshold      = 5000;
26     input->lockdown_threshold = 50000;
27     input->infection_delay     = 5;     // 5 to 6 days incubation period (average) according to WHO
28     input->infection_days      = 3;     // assume three days of passing on the disease
29     input-> starting_infections = 10;
30     input->immunity_duration = 180;   // half a year of immunity
31
32     input->contacts_per_day = cpd;
33     input-> transmission_probability = tp;
34
35     for (int day = blockIdx.x*blockDim.x + threadIdx.x; day < 365; day += gridDim.x*blockDim.x) {
36       input->contacts_per_day[day] = 6;              // arbitrary assumption of six possible transmission
           contacts per person per day, all year
37       input-> transmission_probability [day] = 0.2
38                                          + 0.1 * cos((day / 365.0) * 2 * M_PI);   // higher transmission
                                               in winter, lower transmission during summer
39     }
40 }
```

```
41
42  __global__ void cuda_print(SimInput_t *input) {
43      printf("Threshold: %d \n", input->mask_threshold);
44  }
45
46  typedef struct
47  {
48    // for each day:
49    int * active_infections;      // number of active infected on that day (including incubation period)
50    int *lockdown;                // 0 if no lockdown on that day, 1 if lockdown
51
52    // for each person:
53    int * is_infected;        // 0 if healty, 1 if currently infected
54    int *infected_on;         // day of infection. negative if not yet infected. January 1 is Day 0.
55
56  } SimOutput_t;
57
58  //
59  // Initializes the output data structure (values to zero, allocate arrays)
60  //
61
62  void init_output(SimOutput_t *output, int population_size)
63  {
64    output->active_infections = (int*)malloc(sizeof(int) * DAYS_DIVOC);
65    output->lockdown          = (int*)malloc(sizeof(int) * DAYS_DIVOC);
66    for (int day = 0; day < DAYS_DIVOC; ++day) {
67      output->active_infections[day] = 0;
68      output->lockdown[day] = 0;
69    }
70
71    output->is_infected       = (int*)malloc(sizeof(int) * population_size);
72    output->infected_on       = (int*)malloc(sizeof(int) * population_size);
73
74    for (int i=0; i<population_size; ++i) {
75      output->is_infected[i] = 0;
76      output->infected_on[i] = 0;
77    }
78  }
79
80  __global__ void cuda_init_output(SimOutput_t *output, SimInput_t *input, int *ai, int *ld, int *ii, int *io)
       {
81      output->active_infections = ai;
82      output->lockdown          = ld;
83       for (int day = blockIdx.x*blockDim.x + threadIdx.x; day < 365; day += gridDim.x*blockDim.x) {
84          output->active_infections[day] = 0;
85          output->lockdown[day] = 0;
86      }
87      output->is_infected       = ii;
88      output->infected_on       = io;
89
90      for (int i = blockIdx.x*blockDim.x + threadIdx.x; i < input->population_size; i += gridDim.x*blockDim.x
```

```
        ) {
91          output->is_infected[i] = (i < input->starting_infections) ? 1 : 0;
92          output->infected_on[i] = (i < input->starting_infections) ? 0 : -1;
93      }
94  }
95
96  __global__ void cuda_print_out(SimOutput_t *output) {
97      printf("Infected: %d \n", output->is_infected[0]);
98  }
99
100 __global__ void cuda_determine_infections(const SimInput_t *input, SimOutput_t *output, int *numInfected, int
        *numRecovered, int day) {
101     // Step 1: determine number of infections and recoveries
102     int num_infected_current = 0;
103     int num_recovered_current = 0;
104     __shared__ int shared_inf[256];
105     __shared__ int shared_rec[256];
106
107     for (int i = blockIdx.x*blockDim.x + threadIdx.x; i < input->population_size; i += gridDim.x*blockDim.x
            ) {
108         if (output->is_infected[i] > 0) {
109             if (output->infected_on[i] > day - input->infection_delay - input->infection_days
110             && output->infected_on[i] <= day - input->infection_delay) // currently infected and incubation
                    period over
111             {
112                 num_infected_current += 1;
113                 // printf("person %d is infected on day %d (info in thread %d in block %d)\n", i, day,
                        threadIdx.x, blockIdx.x);
114             }
115             else if (output->infected_on[i] < day - input->infection_delay - input->infection_days)
116             {
117                 num_recovered_current += 1;
118                 // printf("person %d is recovered on day %d (info in thread %d in block %d)\n", i, day);
119             }
120         }
121     }
122
123     shared_inf[threadIdx.x] = num_infected_current;
124     shared_rec[threadIdx.x] = num_recovered_current;
125     for(unsigned int stride = blockDim.x/2; stride > 0 ; stride /=2){
126         __syncthreads();
127         if(threadIdx.x < stride){
128             shared_inf[threadIdx.x] += shared_inf[threadIdx.x + stride];
129             shared_rec[threadIdx.x] += shared_rec[threadIdx.x + stride];
130         }
131     }
132     if (threadIdx.x == 0) {
133         numInfected[blockIdx.x] = shared_inf[0];
134         numRecovered[blockIdx.x] = shared_rec[0];
135     }
136 }
```

```
137
138 // reduction between blocks
139 __global__ void cuda_reduction(int *input) {
140     __shared__ int shared_mem[256];
141     shared_mem[threadIdx.x] = input[threadIdx.x];
142     for(unsigned int stride = blockDim.x/2; stride > 0; stride /=2){
143         __syncthreads(); // synchronize threads within thread block
144         if(threadIdx.x < stride){
145             shared_mem[threadIdx.x] += shared_mem[threadIdx.x + stride];
146         }
147     }
148     if(threadIdx.x == 0) input[0] = shared_mem[0];
149 }
150
151 __global__ void cuda_lockdown(const SimInput_t *input, SimOutput_t *output, int *numInfected, int day) {
152
153         if (numInfected[0] > input->lockdown_threshold) {
154             output->lockdown[day] = 1;
155         }
156         if (day > 0 && output->lockdown[day-1] == 1) { // end lockdown if number of infections has reduced
                significantly
157             output->lockdown[day] = (numInfected[0] < input->lockdown_threshold / 3) ? 0 : 1;
158         }
159 }
160
161 __global__ void cuda_adjust_params(const SimInput_t *input, SimOutput_t *output, int *numInfected, int day) {
162
163     if (numInfected[0] > input->mask_threshold) { // transmission is reduced with masks. Arbitrary factor: 2
164         input->transmission_probability[day] /= 2.0;
165     }
166     if (output->lockdown[day]) { // contacts are significantly reduced in lockdown. Arbitrary factor: 4
167         input->contacts_per_day[day] /= 4;
168     }
169 }
170
171 __global__ void cuda_pass_infection(const SimInput_t *input, SimOutput_t *output, u_int32_t *cuda_rand, int
      day){
172     // if(threadIdx.x < 10 && blockIdx.x == 0) printf("check if infectious: person = %d, yes/no = %d\n",
          threadIdx.x, output->is_infected[threadIdx.x]);
173     for (int i = blockIdx.x*blockDim.x + threadIdx.x; i < input->population_size; i += gridDim.x*blockDim.x
          ) // loop over population
174     {
175         if (    output->is_infected[i] > 0
176             && output->infected_on[i] > day - input->infection_delay - input->infection_days // currently
                    infected
177             && output->infected_on[i] <= day - input->infection_delay)                       // already
                    infectious
178         {
179             // printf("person %d is infectious on day %d (info in thread %d in block %d)\n", i, day,
                    threadIdx.x, blockIdx.x);
180             // pass on infection to other persons with transmission probability
```

```
181             for (int j=0; j<input->contacts_per_day[day]; ++j)
182             {
183                 double r = cuda_rand[i] / 2147483647;
184                 if (r < input->transmission_probability[day]) {
185
186                     r = cuda_rand[i*j] / 2147483647;
187                     int other_person = r * input->population_size;
188                     if (output->is_infected[other_person] == 0     // other person is not infected
189                     || output->infected_on[other_person] < day - input->immunity_duration) { // other person
                            has no more immunity
190                         output->is_infected[other_person] = 1;
191                         output->infected_on[other_person] = day;
192                     }
193                 }
194             } // for contacts_per_day
195         } // if currently infected
196     } // for i
197 }
198
199
200 void run_simulation(const SimInput_t *input, SimOutput_t *output) {
201
202     // Step 1
203     //
204     int *cuda_numInfected, *cuda_numRecovered;
205     CUDA_ERRCHK(cudaMalloc(&cuda_numInfected, sizeof(int)*256));
206     CUDA_ERRCHK(cudaMalloc(&cuda_numRecovered, sizeof(int)*256));
207     int *numRecovered = (int*)malloc(sizeof(int) * 256);
208     int *numInfected = (int*)malloc(sizeof(int) * 256);
209
210
211     int population_size;//, contacts_per_day;
212     cudaMemcpy(&population_size, &input->population_size, sizeof(int), cudaMemcpyDeviceToHost);
213
214     // random number vector
215     u_int32_t *cuda_rand;
216     cudaMalloc(&cuda_rand, sizeof(u_int32_t) * 256 * 256);
217
218     for (int day=0; day< DAYS_DIVOC; ++day)
219     {
220         printf("day %d\n", day);
221         // get todays infected
222         cudaDeviceSynchronize();
223         cuda_determine_infections <<<256,256>>>(input, output, cuda_numInfected, cuda_numRecovered, day);
224         cudaDeviceSynchronize();
225         cuda_reduction<<<1,256>>>(cuda_numInfected);
226         cudaDeviceSynchronize();
227         cuda_reduction<<<1,256>>>(cuda_numRecovered);
228         cudaDeviceSynchronize();
229         int infected;
230         cudaMemcpy(numInfected, cuda_numInfected, sizeof(int),cudaMemcpyDeviceToHost);
```

```
231             infected  = numInfected[0];
232             printf (" Infected  on day %d: %d\n", day, numInfected[0]);
233
234             // check  for  lockdown
235             cuda_lockdown<<<1,1>>>(input, output, cuda_numInfected, day);
236             cudaDeviceSynchronize();
237             //
238             // Step 2: determine today's  transmission  probability  and contacts based on pandemic situation
239             //
240             cuda_adjust_params<<<1,1>>>(input, output, cuda_numInfected, day);
241             cudaDeviceSynchronize();
242
243             //
244             // Step 3: pass on  infections  within  population
245             //
246
247             cuda_RN<<<256,256>>>(cuda_rand, day);
248
249             cudaDeviceSynchronize();
250
251              cuda_pass_infection <<<256,256>>>(input, output, cuda_rand, day);
252             cudaDeviceSynchronize();
253
254         } // for  day
255 }
```

2.1 Port Init. Phase to GPU (4/4 Points)

3 Develop Performance Model and Compare to Execution Times (0/1 Point)

4 BONUS: Implement a Non-Trivial Refinement (0/2 Points)