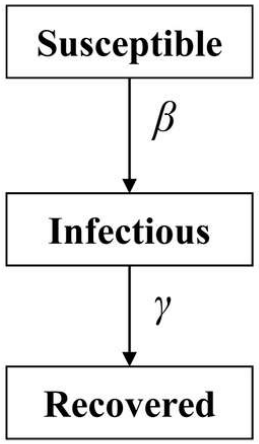
Моделирование распространения инфекционного заболевания в замкнутой популяции с использованием GPU

Выполнил Гаврилов Владислав

SIR - модель

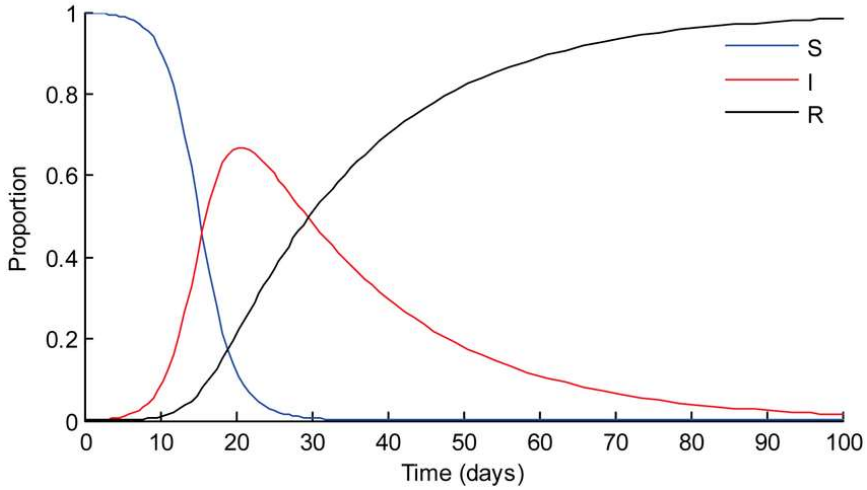
Идея заключается в разделении популяции на три группы: здоровые (S - Susceptible), зараженные (I - Infectious) и переболевшие (R - Recovered). Динамику системы описывает система дифференциальных уравнений.



$$\frac{dS}{dt} = -\beta SI$$

$$\frac{dI}{dt} = \beta SI - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$



Постановка задачи моделирования

Задается начальная система:

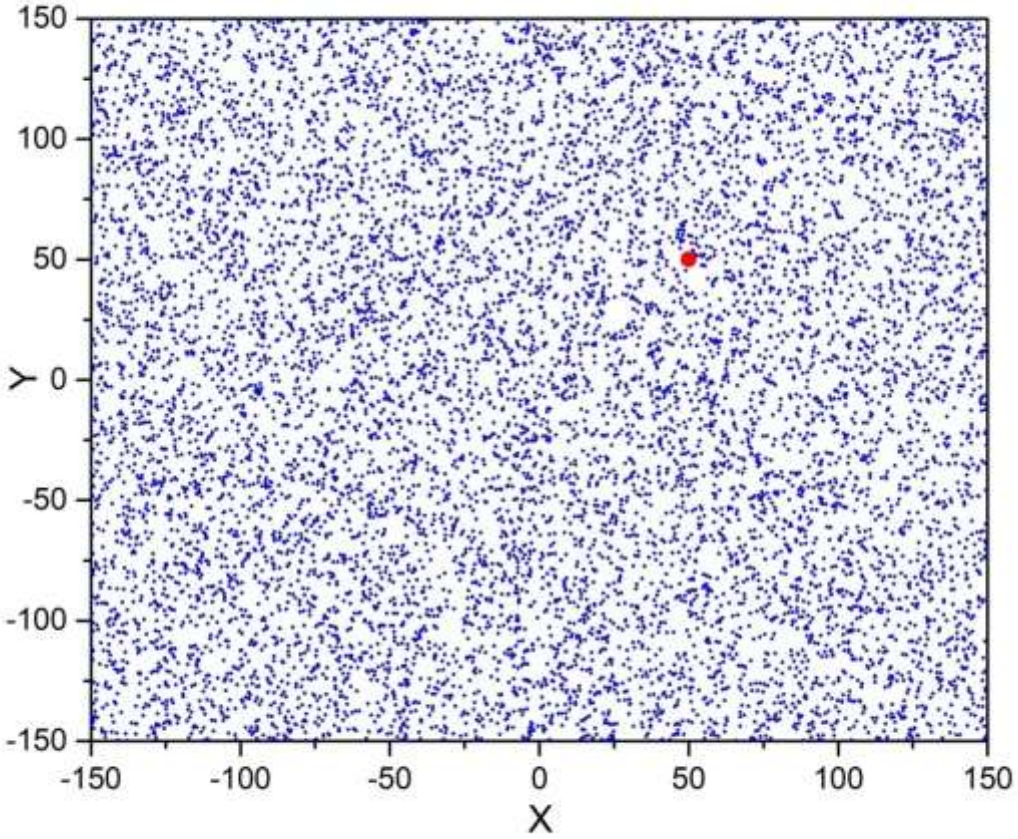
1) Коробка RX на RY (место обитания популяции)
2) N частиц (особей)
3) NOI изначально отмеченных точек (зараженных особей)

Задаются следующие параметры:

1) Радиус инфицирования INF_R
2) Время моделирования T
3) Время выздоровления T_REC
4) Максимальное перемещение частицы по оси MAX_R

Задача:

1) Получить временную зависимость количества зараженных
2) Узнать время окончания пандемии



Изначальное положение 10240 частиц в системе 300 X 300

Движение частиц

Движение осуществляется случайным образом: на каждой итерации координаты каждой частицы рассчитываются прибавлением к координатам на прошлой итерации случайного числа типа float из промежутка [-MAX_R ; MAX_R]. При попадании частицы за пределы коробки на некоторой итерации, координаты для неё пересчитываются заново:

```
x[nt * N + i] = x[(nt-1) * N + i] + 2 * MAX_R * ((float)rand() / RAND_MAX) - MAX_R;
y[nt * N + i] = y[(nt-1) * N + i] + 2 * MAX_R * ((float)rand() / RAND_MAX) - MAX_R;
while ((fabsf(x[nt * N + i]) > RX/2) || (fabsf(y[nt * N + i]) > RY/2))
{
x[nt * N + i] = x[(nt - 1) * N + i] + 2 * MAX_R * ((float)rand() / RAND_MAX) - MAX_R;
y[nt * N + i] = y[(nt - 1) * N + i] + 2 * MAX_R * ((float)rand() / RAND_MAX) - MAX_R;
}
```

Процесс инфицирования

На каждой итерации для каждой частицы определяется, находится ли она в радиусе инфицирования зараженных частиц. Если условие выполнено, то ее статус s помечается единицей. Условие проверяется функцией status:

```
__global__ void status_GPU(float *x, float *y, int *s, int nt)
{float xx, yy, rr;
int i = threadIdx.x + blockIdx.x * blockDim.x;
for (int j = 0; j < N; j++)
{if ((j != i) && (s[i] == 0) && (s[j] == 1))
{xx = x[nt * N + i] - x[nt * N + j];
yy = y[nt * N + i] - y[nt * N + j];
rr = sqrtf(xx * xx + yy * yy);
if (rr < INF_R)
{s[i] = 1;}}}}
```

Процесс выздоровления

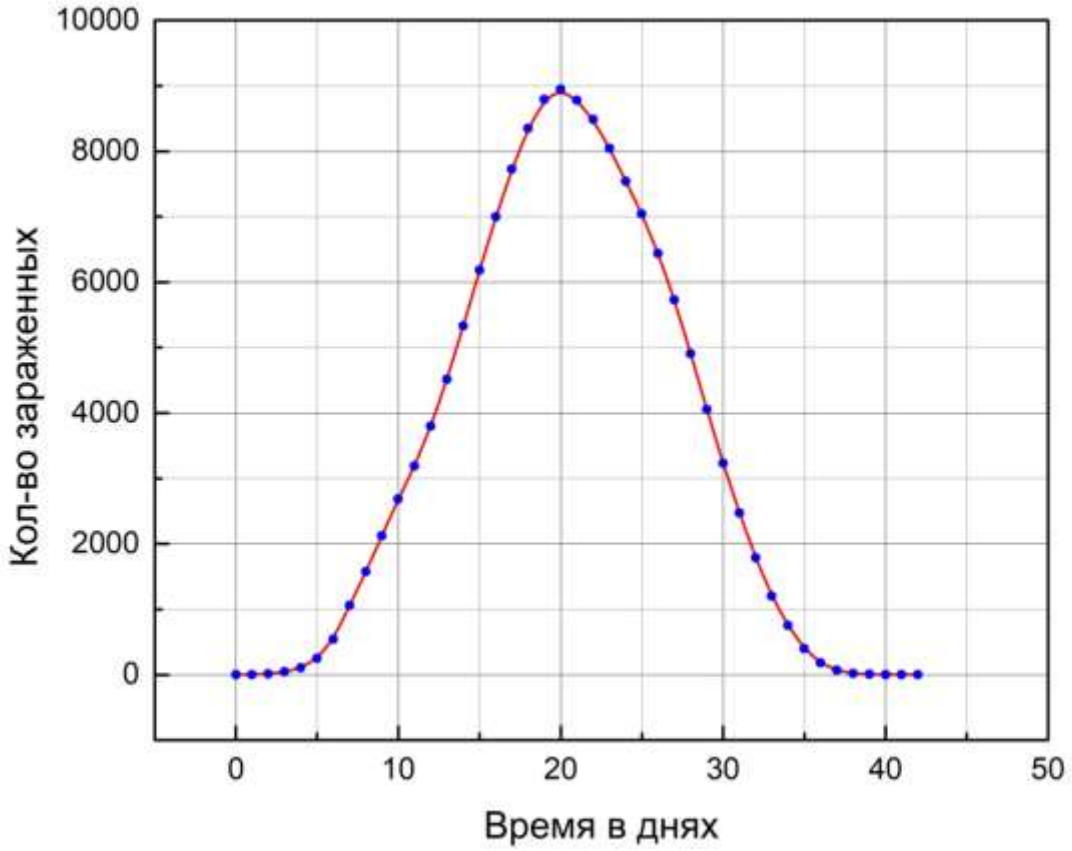
Если на данной итерации статус частицы равен единице (то есть особь инфицирована) , то переменная t выздоровления увеличивается на единицу. Если переменная t = T_REC, то статус частицы меняется на 2 (особь выздоравливает). Процесс выздоровления симулируется функцией control:

```
__global__ void control_GPU(int *s, int *t, int nt)
{
int i = threadIdx.x + blockIdx.x * blockDim.x;
if (t[i] == T_REC)
{
s[i] = 2;
}
if (s[i] == 1)
{
t[i] = t[i] + 1;
}
}
```

Результаты 1

Параметры моделирования:
RX = 300, RY = 300 (размеры города)
T = 60
T_REC = 14 (время выздоровления)
INF_R = 2 (радиус заражения)
N = 10240 (кол-во особей)
MAX_R = 25 (перемещение)

Результаты:
Продолжительность пандемии = 43 дня
(GPU+CPU)_time = 4926 ms
CPU_time = 27843 ms

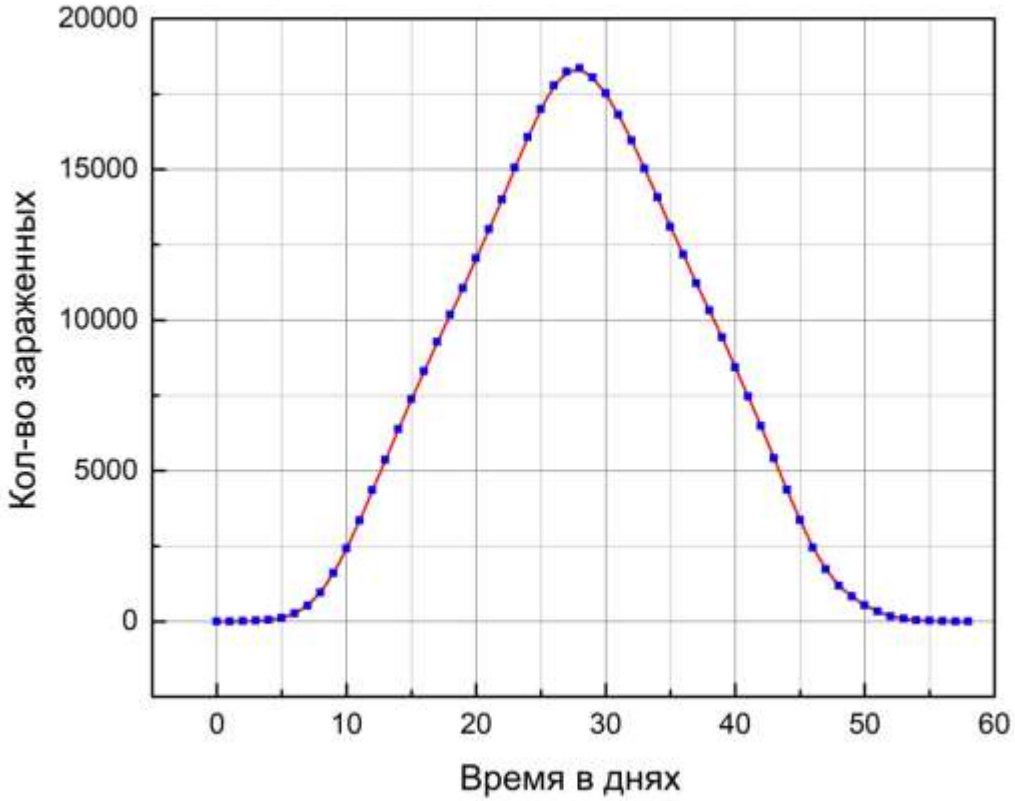


Временная зависимость кол-ва заражённых

Результаты 2

Параметры моделирования:
RX = 700, RY = 700 (размеры города)
T = 150
T_REC = 20 (время выздоровления)
INF_R = 3 (радиус заражения)
N = 20480 (кол-во особей)
MAX_R = 35 (перемещение)

Результаты:
Продолжительность пандемии = 59 дней
(GPU+CPU)_time = 43827 ms
CPU_time = 249265 ms



Временная зависимость кол-ва зараженных

Выводы

1) Проверена адекватность придуманной модели и ее соответствие теоретической SIR – модели.
2) Написаны реализации данной модели для CPU и для гибрида GPU+CPU.
3) Получено ускорение выполнения алгоритма в 5-6 раз с помощью использования гибридного вычисления GPU+CPU. Стоит однако отметить, что корректно оценить выгоду использования GPU в данном случае сложно, так как процессы вероятностные и даже при равных начальных условиях алгоритмом может быть выполнено разное кол-во операций.