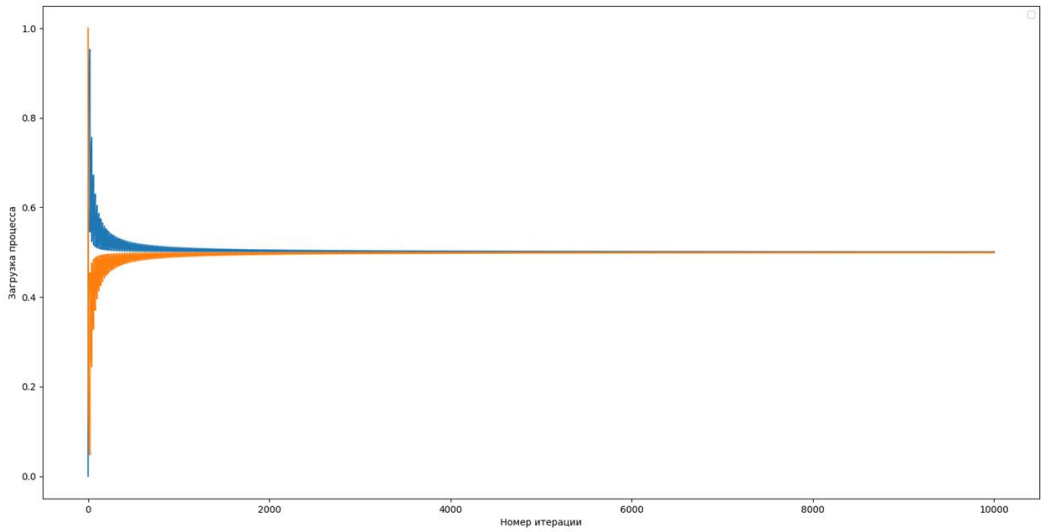
Отчет по 4-ому заданию «Системы Линденмайера»

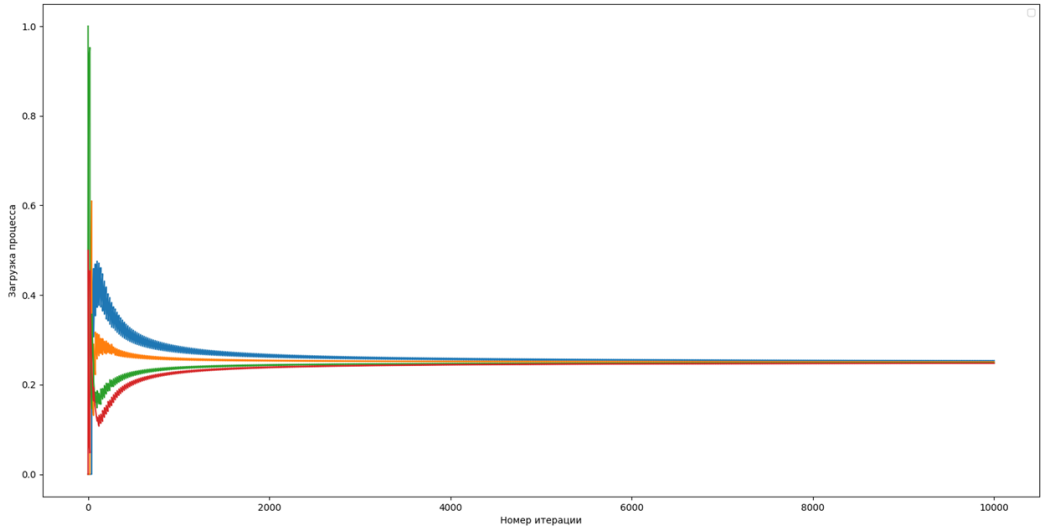
выполнил Женин Максим Николаевич студент 523 группы
Mail:maximham@mail.ru

Отчет запуска программы:
Ниже приведены графики, отражающие зависимость загрузки процессов от номера итерации t, где число процессов p ∈ {2, 4, 8, 16}, число итераций m = 10000(m решил сократить в два раза, так как с увеличением кол-ва итераций графики начинают сильно пересекаться друг с другом, что я посчитал не совсем информативным, также чтобы уменьшить время проведения экспериментов) , шаг обмена k = 20):
Графики для L1 = {a → ab, b → bc, ω = a;};

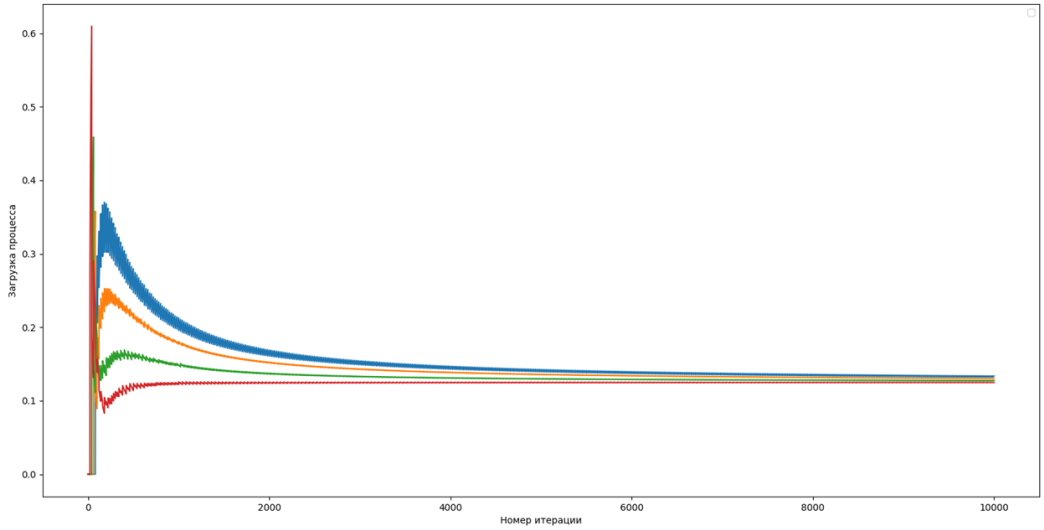
P = 2



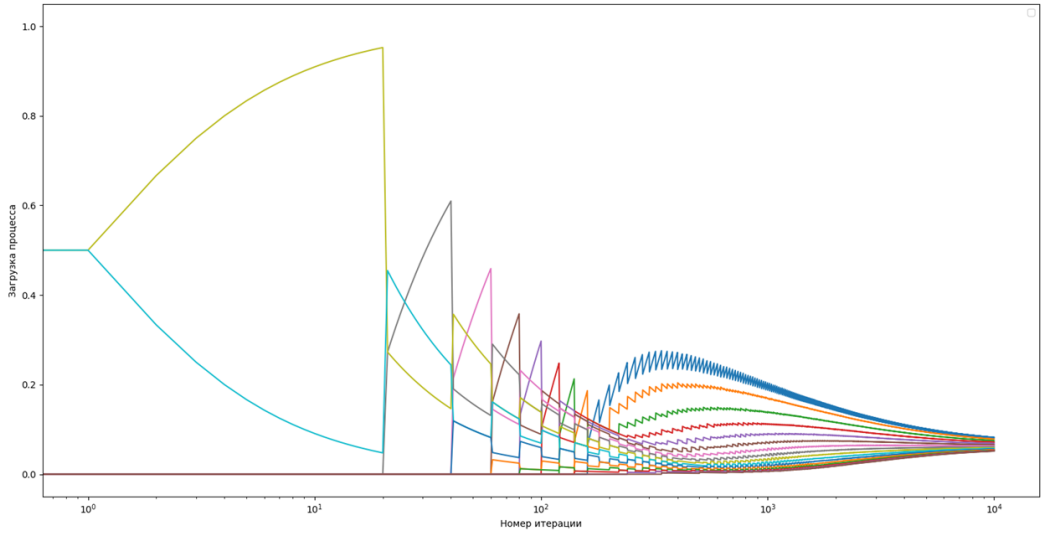
P = 4


Загрузка процесса
Номер итерации
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000

P = 8

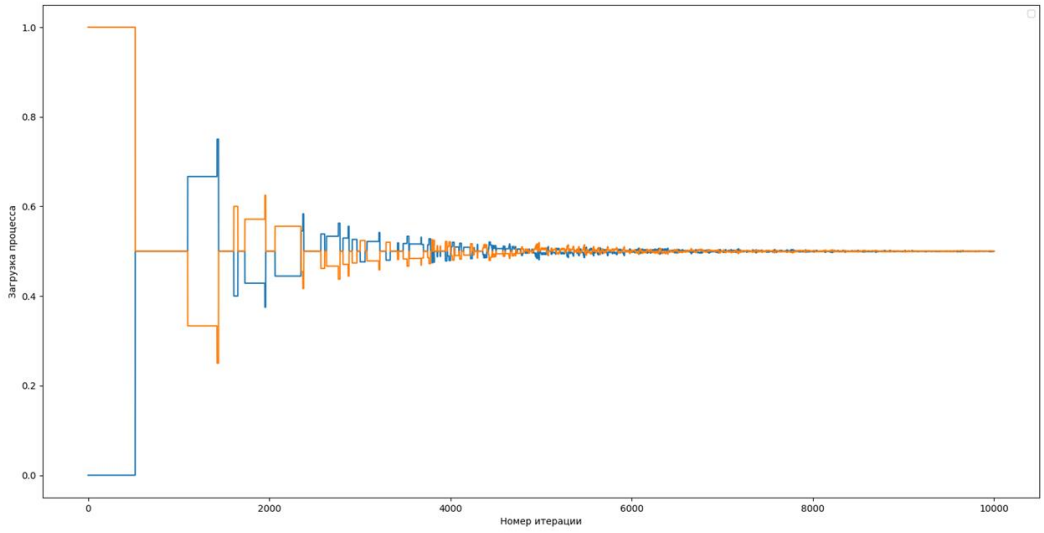

Загрузка процесса
Номер итерации
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

P = 16

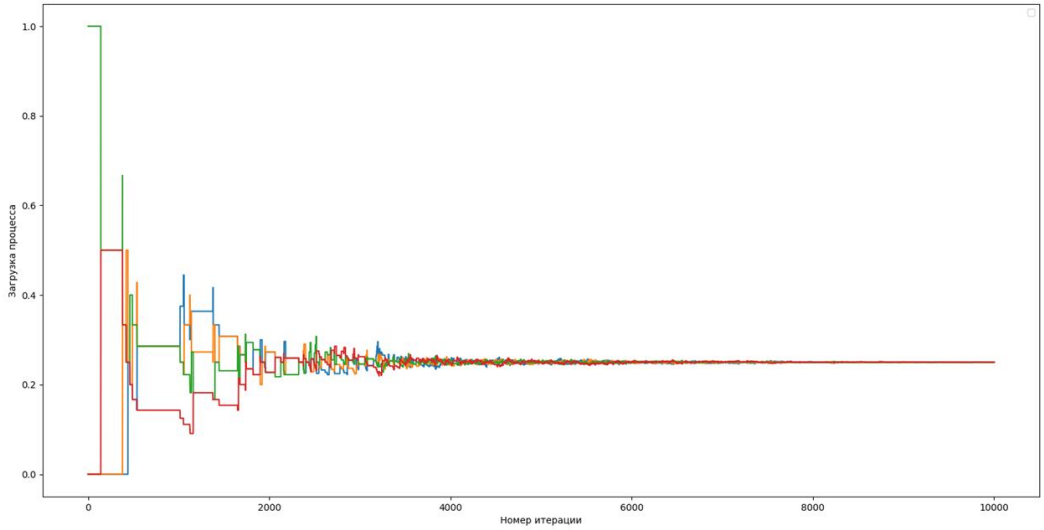


L2 система = {a → aa [0.001], ω = a;}

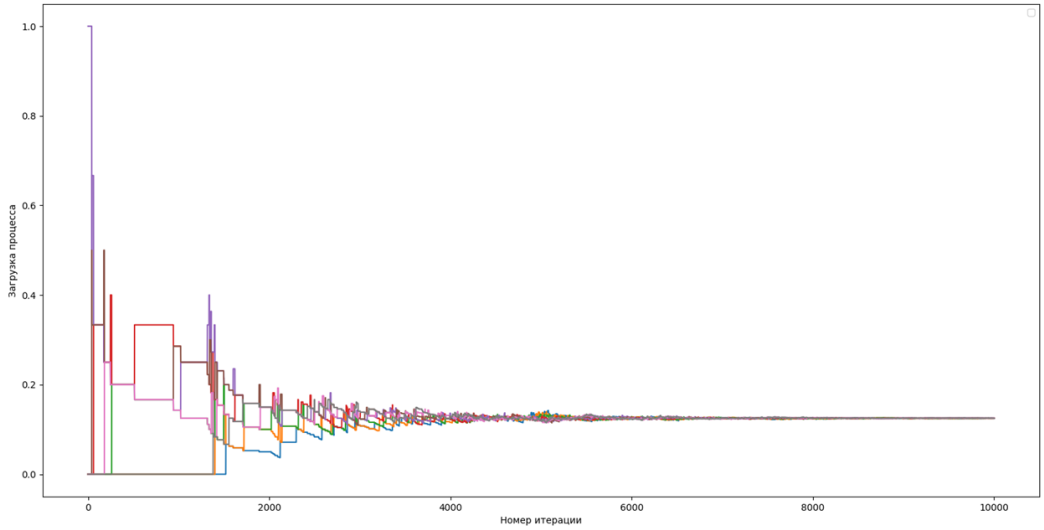
P = 2



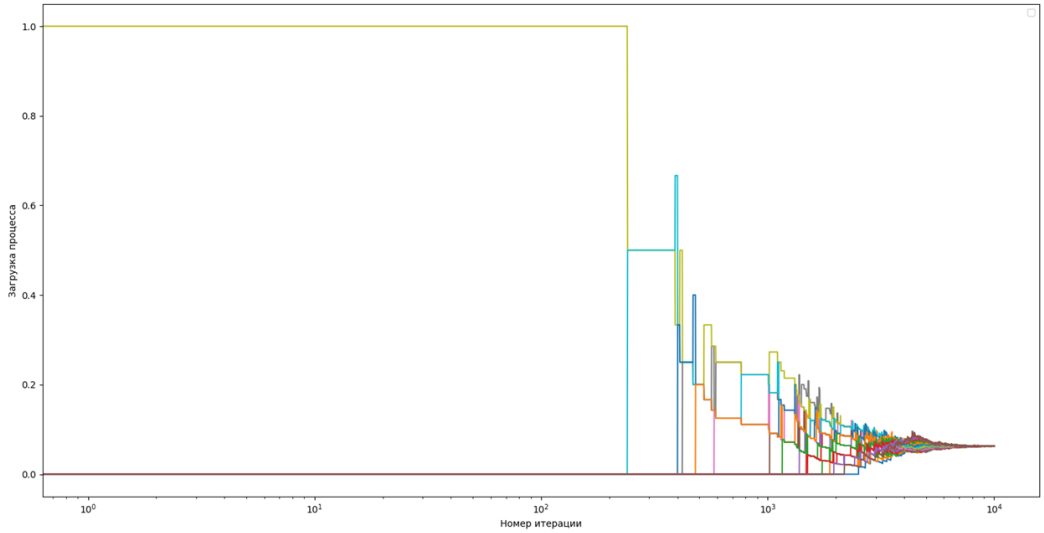
P = 4


1.0
0.8
0.6
0.4
0.2
0.0
Загрузка процесса
0
2000
4000
6000
8000
10000
Номер итерации

P = 8

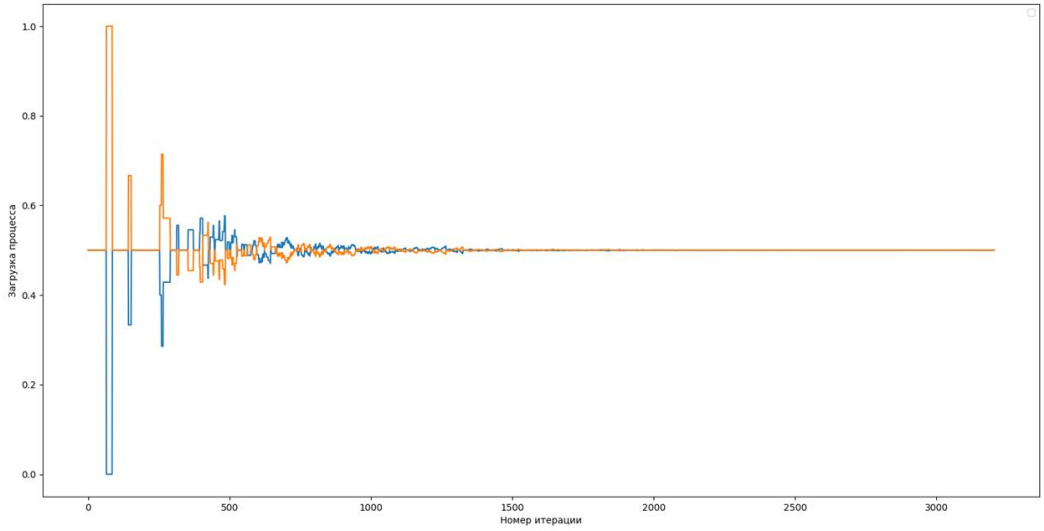

1.0
0.8
0.6
0.4
0.2
0.0
Загрузка процесса
0
2000
4000
6000
8000
10000
Номер итерации

P = 16

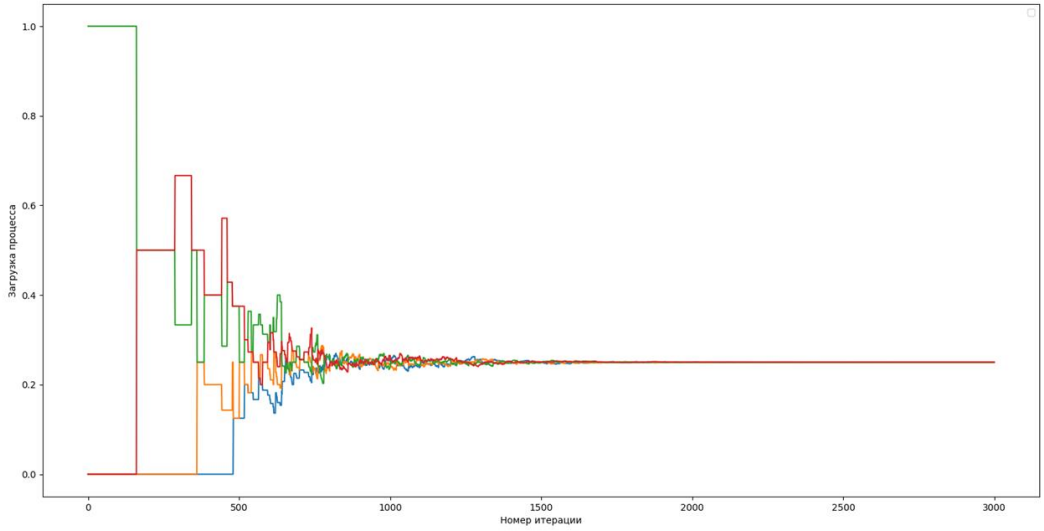


L3 система = {a → ab [0.01], b → a [0.01], ω = a;}
Тут кол-во итераций снизил до 3000 тысяч, так как иначе очень долго приходилось ждать.

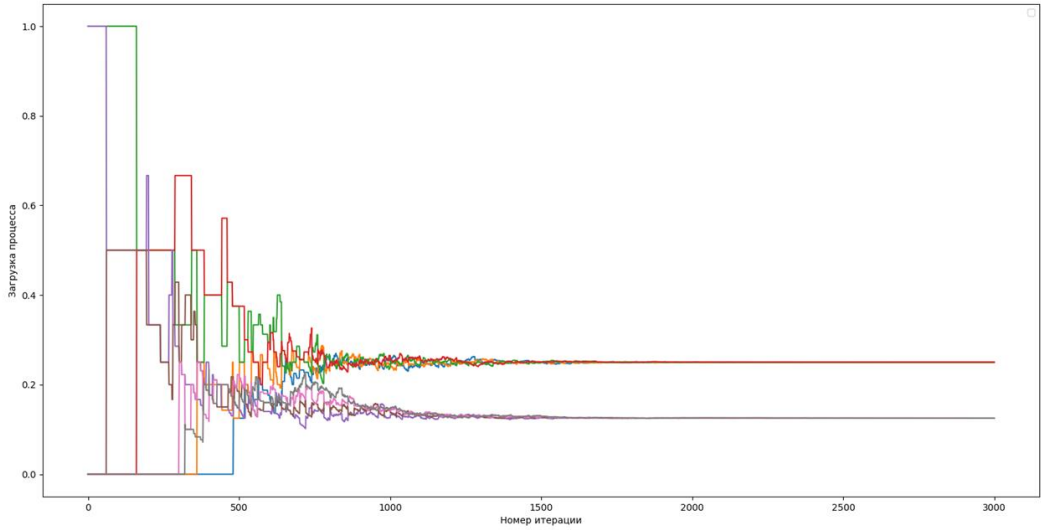
P = 2



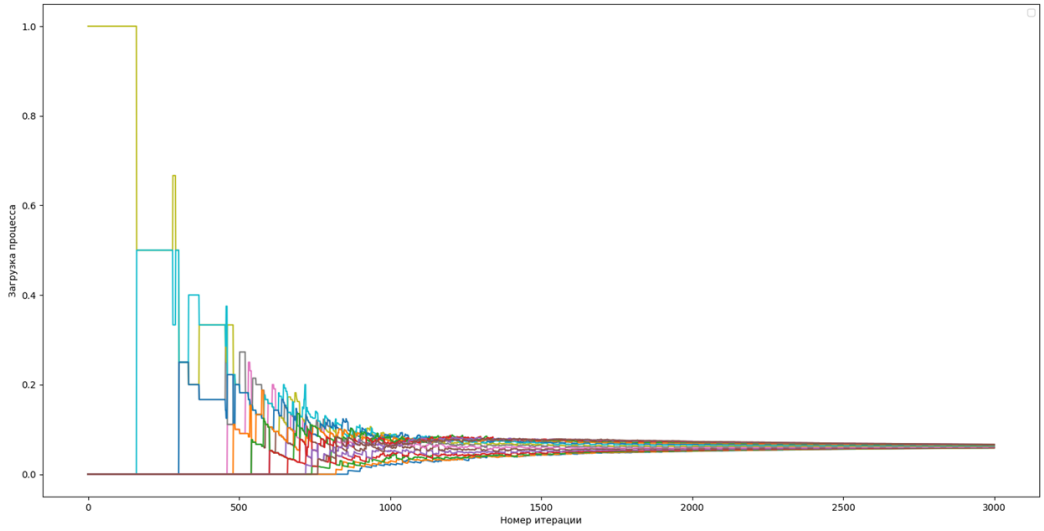
P = 4


Загрузка процесса
Номер итерации
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
2500
3000

P = 8


Загрузка процесса
Номер итерации
0.0
0.2
0.4
0.6
0.8
1.0
0
500
1000
1500
2000
2500
3000

P = 16



Код программы:

```
#include <fstream>
#include <string>
#include <map>
#include "mpi.h"
#include <random>
#include <vector>
#include <stdlib.h>

double randZeroToOne()
{
return rand() / (RAND_MAX + 1.);
}
using namespace std;

int Rank;
int Nproc;

string update_data(string &data, map<char, string>& R, double p1)
{
string buf = "";

for( unsigned int i=0; i<data.length(); i++ ){
double p = randZeroToOne();
if (p < p1) {
buf += R[data[i]];
} else {
buf += data[i];
}
}
```

```
return buf;
}

long long GetCount(long long l, long long l1) {
if (l > l1) {
return (l - l1) / 2;
} else {
return 0;
}
}

void AlignLoad(string &data) {
int prev = Rank == 0 ? MPI_PROC_NULL: Rank -1;
int next = Rank == Nproc - 1 ? MPI_PROC_NULL: Rank + 1;
long long l = data.length();
long long l_prev = l;
long long l_next = l;
MPI_Sendrecv(&l, 1, MPI_LONG_LONG, prev, 0, &l_next, 1, MPI_LONG_LONG, next, 0,
MPI_COMM_WORLD, 0);
MPI_Sendrecv(&l, 1, MPI_LONG_LONG, next, 0, &l_prev, 1, MPI_LONG_LONG, prev, 0,
MPI_COMM_WORLD, 0);
long long send_next = GetCount(l, l_next);
long long get_prev = GetCount(l_prev, l);
string tmp;
tmp.resize(get_prev);
MPI_Sendrecv(&data[0] + l - send_next, send_next, MPI_CHAR, next, 0, &tmp[0], get_prev, MPI_CHAR, prev,
0, MPI_COMM_WORLD, 0);
data = tmp + data.substr(0, l - send_next);
l = data.length();
l_next = l;
l_prev = l;
MPI_Sendrecv(&l, 1, MPI_LONG_LONG, prev, 0, &l_next, 1, MPI_LONG_LONG, next, 0,
MPI_COMM_WORLD, 0);
MPI_Sendrecv(&l, 1, MPI_LONG_LONG, next, 0, &l_prev, 1, MPI_LONG_LONG, prev, 0,
MPI_COMM_WORLD, 0);
long long send_prev = GetCount(l, l_prev);
long long get_next = GetCount(l_next, l);
tmp.resize(get_next);
MPI_Sendrecv(&data[0], send_prev, MPI_CHAR, prev, 0, &tmp[0], get_next, MPI_CHAR, next, 0,
MPI_COMM_WORLD, 0);
data = data.substr(send_prev, l - send_prev) + tmp;
}


void run_lsystem(int m, int k, int L)
{
string data;
if (Rank == Nproc / 2) {
data = "a";
} else {
data = "";
```

```
}
map<char, string> R;
double probability = 2.0;
if (L == 1){
R['a'] = "ab";
R['b'] = "bc";
}
else if(L == 2){
R['a'] = "aa";
probability = 0.001;
}
else {
R['a'] = "ab";
R['b'] = "a";
probability = 0.01;
}
for( int t=0; t<m; t++ ) {
data = update_data(data,R,probability);
if (t % k == 0) {
AlignLoad(data);
}
}
long long l = data.length();
if (Rank == 0) {
std::vector<long long > lengths(Nproc);
MPI_Gather(&l, 1, MPI_LONG_LONG, &lengths[0], 1, MPI_LONG_LONG, 0, MPI_COMM_WORLD);
long long maxl = 0;
for(auto i : lengths) {
maxl = max(maxl, i);
}
string tmp;
tmp.resize(maxl);
for(int i = 1 ; i < Nproc; ++i) {
MPI_Recv(&tmp[0], lengths[i], MPI_CHAR, i, 0, MPI_COMM_WORLD, 0);
data += tmp.substr(0, lengths[i]);
}
ofstream fout("output.txt");
fout << data << endl;
} else {
MPI_Gather(&l, 1, MPI_LONG_LONG, 0, 0, MPI_LONG_LONG, 0, MPI_COMM_WORLD);
MPI_Send(&data[0], l, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
}
}

int main(int argc, char** argv)
{
if (MPI_Init(&argc, &argv) != MPI_SUCCESS) {
fprintf(stderr, "failed to init MPI\n");
exit(1);
}
```

```
if (MPI_Comm_size(MPI_COMM_WORLD, &Nproc) != MPI_SUCCESS ||
MPI_Comm_rank(MPI_COMM_WORLD, &Rank) != MPI_SUCCESS) {
fprintf(stderr, "failed to get communicator Nproc or Rank\n");
exit(1);
}
int m = atoi(argv[1]); // число итераций алгоритма
int k = atoi(argv[2]); // шаг обмена
int Lsys = atoi(argv[3]);
run_lsystem(m,k,Lsys);
MPI_Finalize();
return 0;
}
```