Отчет по выполнению практического задания «Системы Линденмайера по курсу Естественные модели параллельных вычислений студента 523 группы Ухина Сергея Алексеевича.

ser191097@gmail.com

Графики зависимостей загруженности процессов от номера итерации.

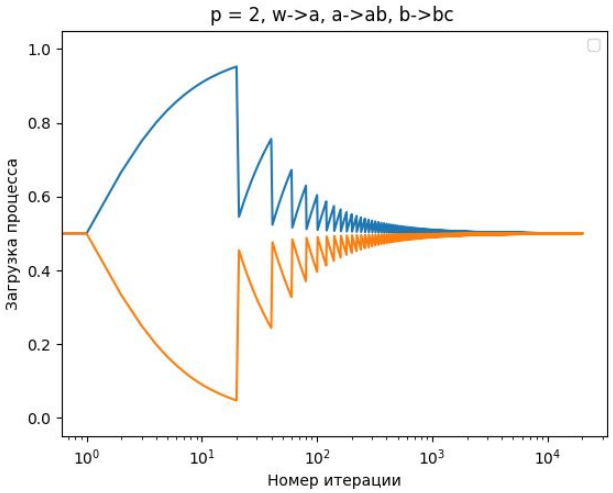

Рис. 1: Первая система, p = 2

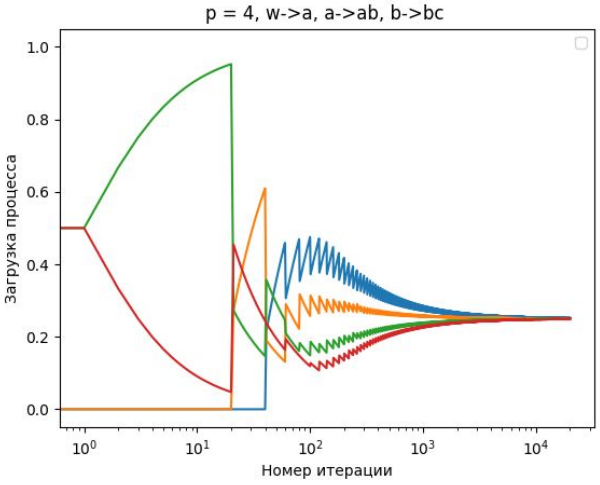

Рис. 2: Первая система, p = 4

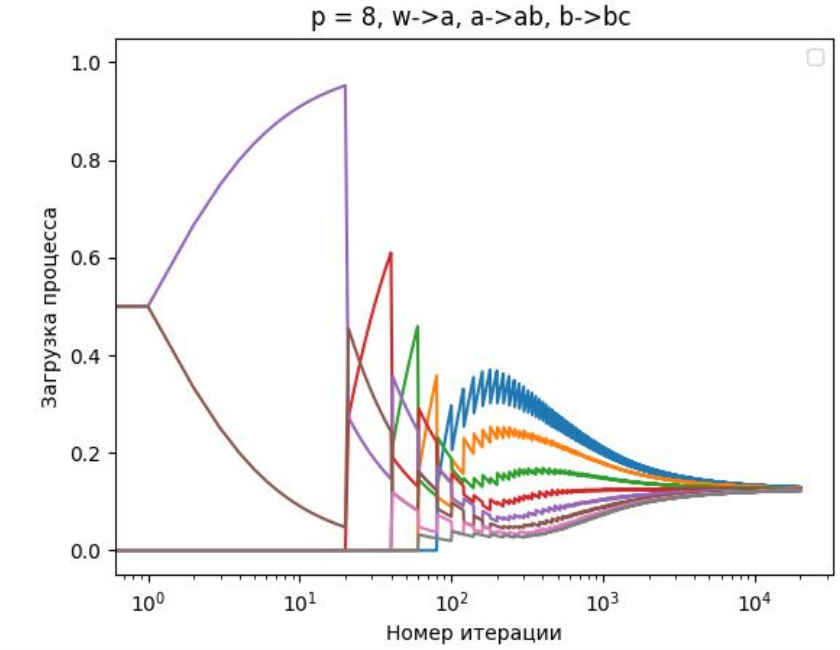

Рис. 3: Первая система, p = 8

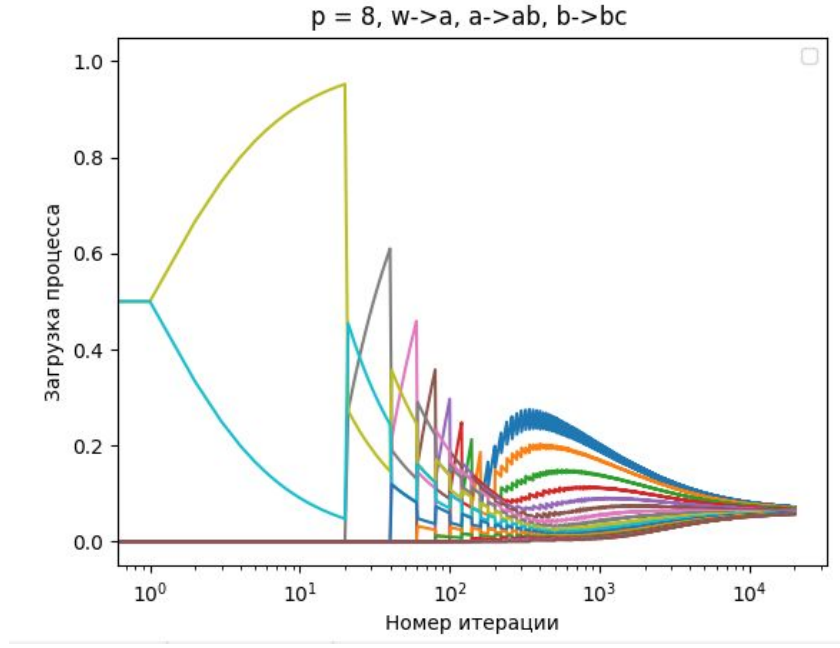

Рис. 4: Первая система, p = 16

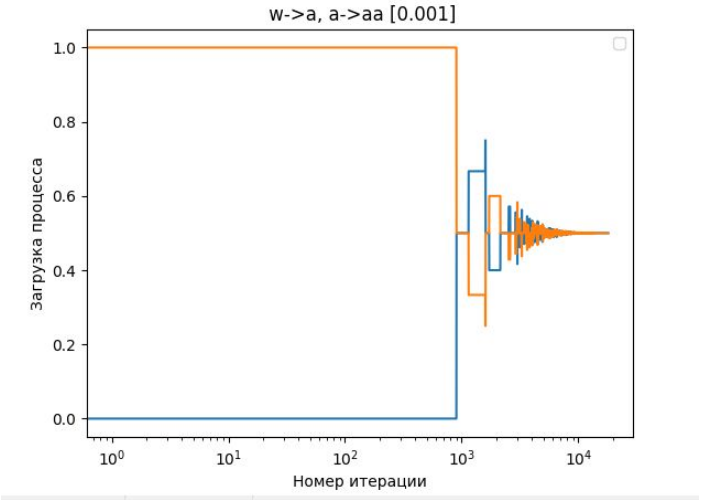

Рис. 5: Вторая система, p = 2

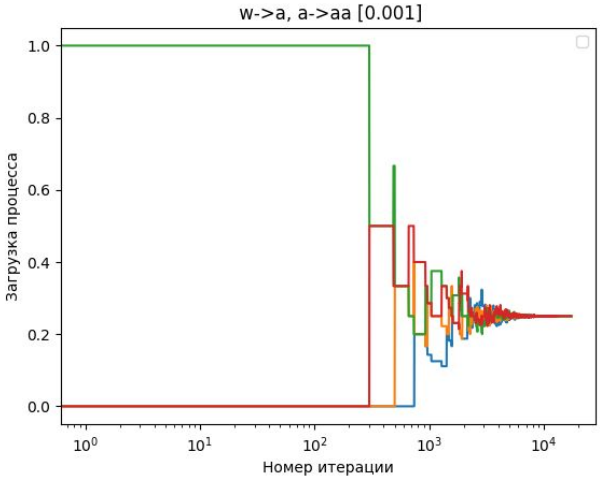

Рис. 6: Вторая система, p = 4

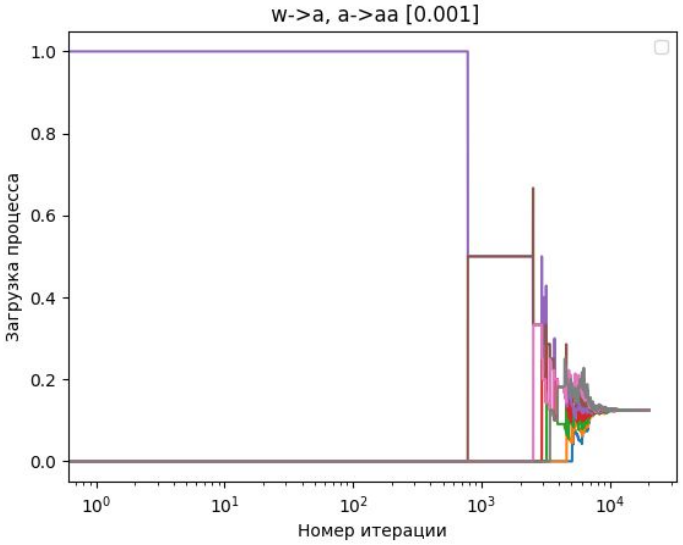

Рис. 7: Вторая система, p = 8

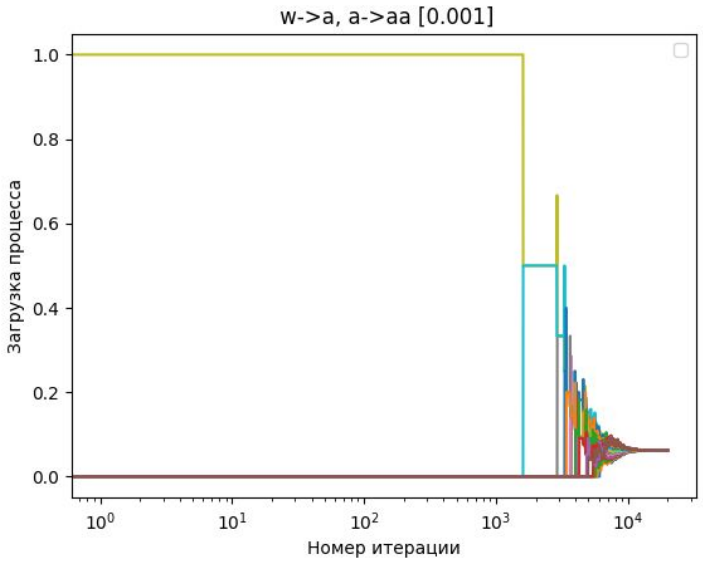

Рис. 8: Вторая система, p = 16

Для третьей системы количество итераций равно 3000, так как очень быстро растет строка.

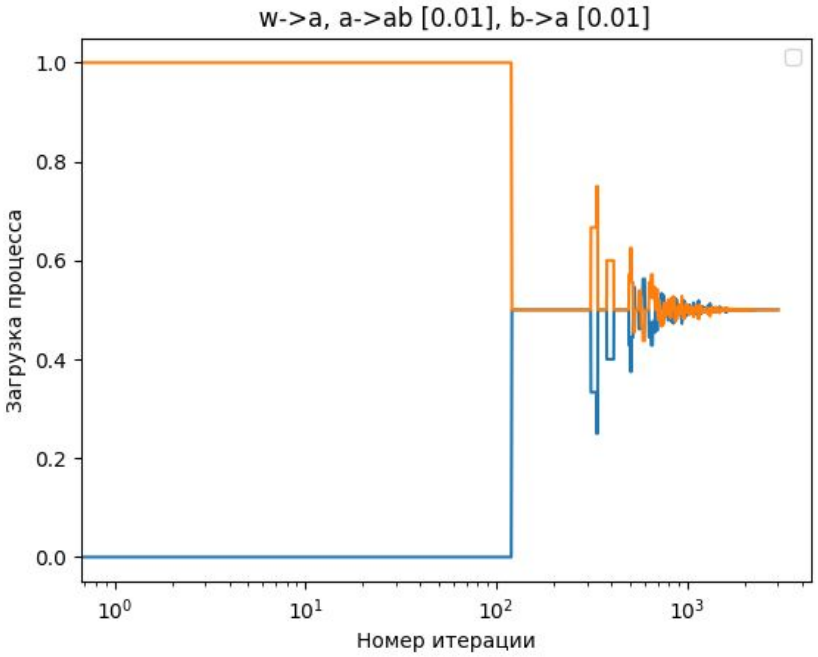

Рис. 9: Третья система, p = 2

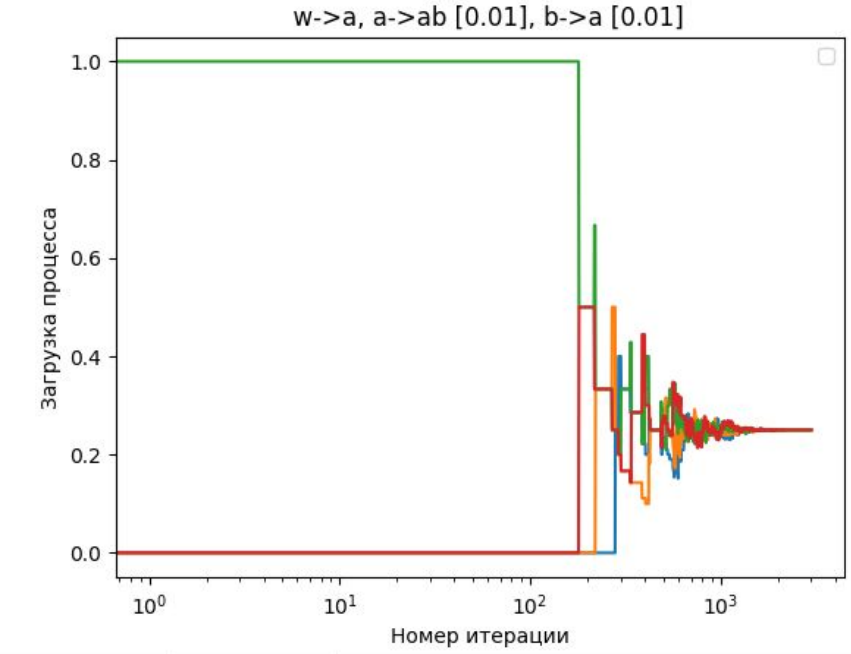

Рис. 10: Третья система, p = 4

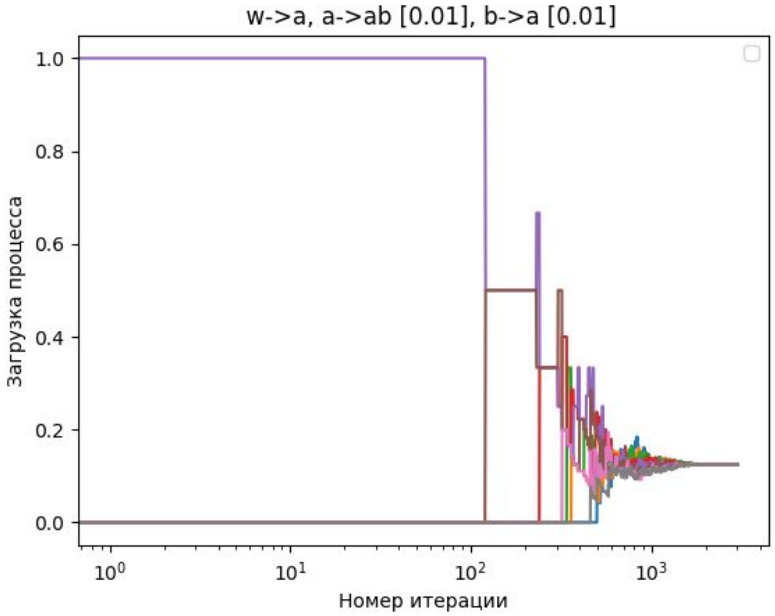

Рис. 11: Третья система, p = 8

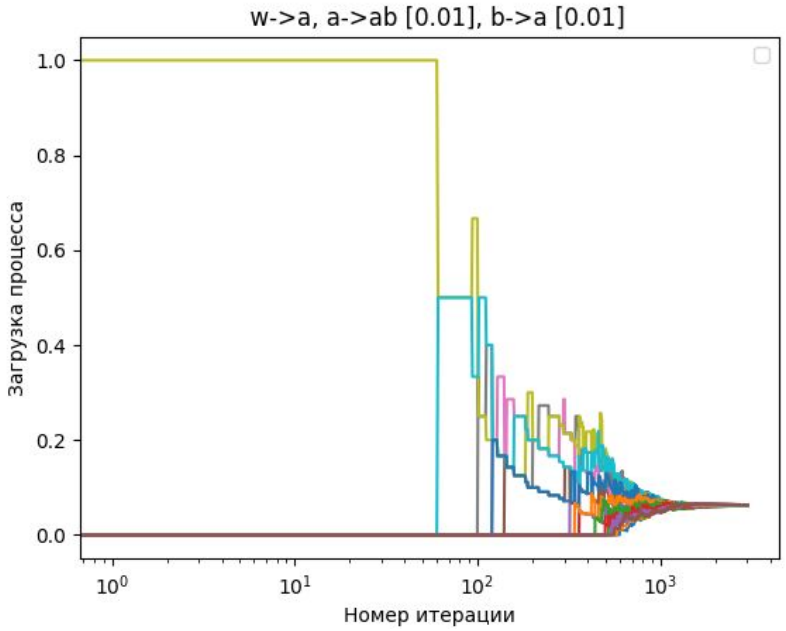

Рис. 12: Третья система, p = 16

Листинг 1: Код программы

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

using namespace std;



double genDouble ( const double a , const double b ) {
   static random_device rd ;
   static mt19937 gen (rd()) ;
   uniform_real_distribution <> dis (a,b) ;
   return dis(gen) ;
}

string update_data(string &data, map<char, string>& R, double prob = 10)
{
   string buf = "";

   for( unsigned int i=0; i<data.length(); i++ ){
      double p = genDouble(0, 1);
      if (p < prob) {
         buf += R[data[i]];
      } else {
```

```
            buf += data[i];
        }
    }

    return buf;
}

long long  GetCount(long long l, long long l2) {
    if (l > l2) {
        return (l - l2) / 2;
    } else {
        return 0;
    }
}

void AlignLoad(string &data, int rank, int size) {
    int prev = rank == 0 ? MPI_PROC_NULL: rank -1;
    int next = rank == size - 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 PrintStat(int it, string &data, int rank, int size) {
   long long l = data.length();
   long long suml = 0;
   MPI_Allreduce(&l, &suml, 1, MPI_LONG_LONG, MPI_SUM, MPI_COMM_WORLD);
   ofstream fout( to_string(size) + "stat" + to_string(rank) + ".txt", ios::app);
   fout.setf(ios::fixed);
   fout << (double) l / suml << endl;
}

void run_lsystem(int m, int k, int rank, int size)
{
   string data;
   if (rank == size / 2) {
      data = "a";
   } else {
      data = "";
   }
   // система правил
   map<char, string> R;
   R['a'] = "ab";
   R['b'] = "a";
   // основной цикл
   for( int t=0; t<m; t++ ) {
      data = update_data(data,R, 0.01);
      PrintStat(t, data, rank, size);
      if (t % k == 0) {
         AlignLoad(data, rank, size);
      }
   }
   // сохранение результатов расчета
   long long l = data.length();
   if (rank == 0) {
      std::vector<long long > lengths(size);
      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 < size; ++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)
{
    MPI_Init(&argc, &argv);
    int rank;
    int size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank) ;
    MPI_Comm_size(MPI_COMM_WORLD, &size) ;
    int m = atoi(argv[1]); // число итераций алгоритма
    int k = atoi(argv[2]); // шаг обмена
    run_lsystem(m,k, rank, size);
    MPI_Finalize();
    return 0;
}
```