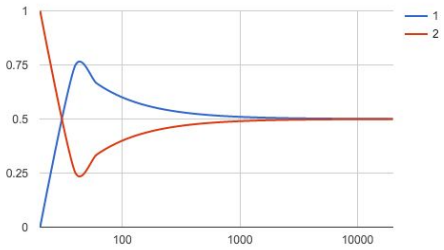
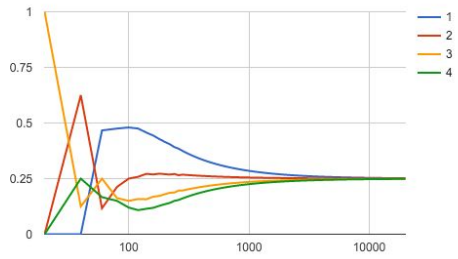
Задание #4. Системы Линденмайера

Горемыкин Александр Олегович, ifresh.wp@gmail.com

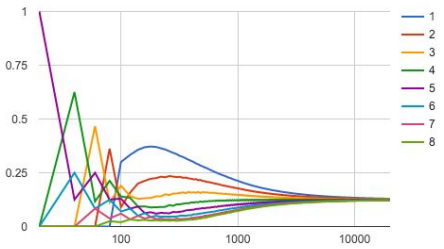
L1, 2 процесса



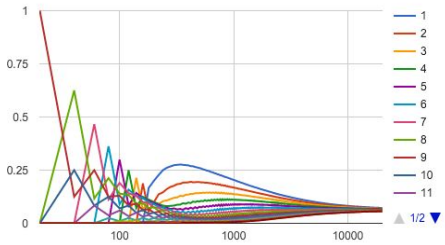
L1, 4 процесса



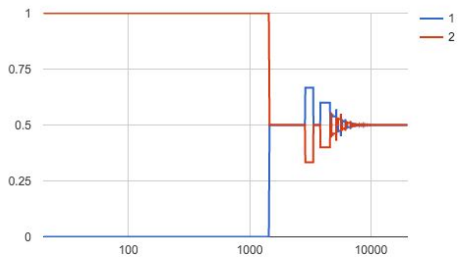
L1, 8 процессов



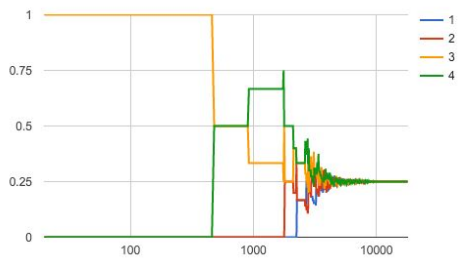
L2, 16 процессов



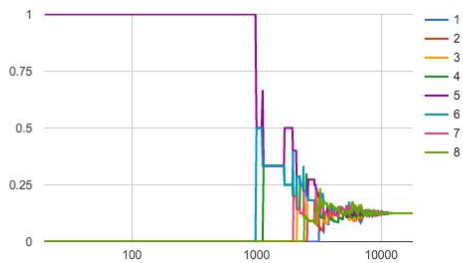
L2, 2 процесса



L2, 4 процесса



L2, 8 процессов



L2, 16 процессов

---

github: https://github.com/sanllier/Practice_Ershov/tree/master/03

```
#include <fstream>
#include <ctime>
#include <vector>

#include "helpers.h"
#include "parparser.h"

//-------------------------------------------------------------------------------

inline int getCount(int a, int b) { return a > b ? (a - b) / 2 : 0; }

void alignLoad(string& state, int prev, int next) {
    int lenNext = state.length();
    int len = state.length();
    MPI_Status status;

    MPICHECK(MPI_Sendrecv(&len, 1, MPI_INT, prev, 0, &lenNext, 1, MPI_INT,
        next, 0, MPI_COMM_WORLD, &status));
    int c = getCount(len, lenNext);

    int recvSize;
    MPICHECK(MPI_Probe(prev, 0xF, MPI_COMM_WORLD, &status));
    MPICHECK(MPI_Get_count(&status, MPI_CHAR, &recvSize));
    string temp(recvSize, 0);
    MPICHECK(MPI_Sendrecv(&state[len - c], c, MPI_CHAR, next, 0xF, &temp[0], recvSize,
MPI_CHAR,
        prev, 0xF, MPI_COMM_WORLD, &status));
    state = temp.append(state.substr(0, len - c));

    // ---------------------------------------------

    int lenPrev = state.length();
    len = state.length();

    MPICHECK(MPI_Sendrecv(&len, 1, MPI_INT, next, 0, &lenPrev, 1, MPI_INT,
        prev, 0, MPI_COMM_WORLD, &status));
    c = getCount(len, lenPrev);

    MPICHECK(MPI_Probe(next, 0xFF, MPI_COMM_WORLD, &status));
    MPICHECK(MPI_Get_count(&status, MPI_CHAR, &recvSize));
    temp.resize(recvSize);
    MPICHECK(MPI_Sendrecv(&state[0], c, MPI_CHAR, prev, 0xFF, &temp[0], recvSize,
MPI_CHAR,
        next, 0xFF, MPI_COMM_WORLD, &status));

    state = state.substr(c, len - c).append(temp);
}

void printStat(const string& state, int commSize, ostream* oStr) {
    MPI_Status status;

    if (oStr != 0) {
        long long sum = state.length();
        vector<int> lens(commSize, 0);
        lens[0] = state.length();

        for (int i = 1; i < commSize; ++i) {
            MPICHECK(MPI_Recv(&lens[i], 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status));
            sum += lens[i];
```

```
        }

        for (int i = 0; i < commSize; ++i) {
            (*oStr) << float(lens[i]) / float(sum) << ", ";
        }
    } else {
        const int localLen = state.length();
        MPICHECK(MPI_Send(&localLen, 1, MPI_INT, MASTER, 0, MPI_COMM_WORLD));
    }
}

void updateStateLOne(string& state) {
    string temp = "";

    for (int i = 0; i < state.length(); ++i) {
        if (state[i] == 'a') temp.append("ab");
        else if (state[i] == 'b') temp.append("bc");
    }

    state = temp;
}

void updateStateLTwo(string& state) {
    string temp = "";

    for (int i = 0; i < state.length(); ++i) {
        float r = float(rand()) / float(RAND_MAX);
        if (state[i] == 'a') temp.append(r <= 0.001 ? "aa" : "a");
    }

    state = temp;
}

void updateStateLThree(string& state) {
    string temp = "";

    for (int i = 0; i < state.length(); ++i) {
        float r = float(rand()) / float(RAND_MAX);
        if (state[i] == 'a') temp.append(r <= 0.01 ? "ab" : "a");
        if (state[i] == 'b') temp.append(r <= 0.01 ? "a" : "b");
    }

    state = temp;
}

void updateState(string& state, int test) {
    if (test == 1) updateStateLOne(state);
    else if (test == 2) updateStateLTwo(state);
    else updateStateLThree(state);
}

//------------------------------------------------------------------------------

int main(int argc, char** argv) {

       parparser parser(argc, argv);

    const int iterationsThreshold = parser.get("m").asInt();
    const int balanceStep = parser.get("k").asInt();
    const int test = parser.get("t").asInt();
    const string outFile = parser.get("o").asString();
    const string statFile = parser.get("s").asString();

    srand(time(0));
```

```
        //---------------------------------------------------------------------------

        MPICHECK(MPI_Init(&argc, &argv));
        //---------------------------------------------------------------------------

        int commSize = 0;
        int rank = 0;
        MPICHECK(MPI_Comm_size(MPI_COMM_WORLD, &commSize));
        MPICHECK(MPI_Comm_rank(MPI_COMM_WORLD, &rank));

    const int prevProcRank = rank > 0 ? rank - 1 : MPI_PROC_NULL;
    const int nextProcRank = rank < commSize - 1 ? rank + 1 : MPI_PROC_NULL;

    string state = "";
    if (rank == commSize / 2) { state = "a"; }

    ofstream *oStr = 0;
    if (rank == MASTER) {
        oStr = new ofstream(statFile.empty() ? "stat.txt" : statFile, ofstream::out);
    }

    //---------------------------------------------------------------------------

    const double startTime = MPI_Wtime();

    for (int i = 1; i <= iterationsThreshold; ++i) {
        if (i % 100 == 0 && rank == MASTER) {
            cout << i << "/" << iterationsThreshold << "\n";
        }

        updateState(state, test);

        if (i % balanceStep == 0) {
            if (rank == MASTER) (*oStr) << i << ", ";
            printStat(state, commSize, oStr);
            if (rank == MASTER) (*oStr) << "\n";

            alignLoad(state, prevProcRank, nextProcRank);
        }
    }

    const double endTime = MPI_Wtime();

    if (rank == MASTER) {
        oStr->close();
        delete oStr;
    }

    MPI_Status status;

    if (rank == MASTER) {
        ofstream oStr(outFile.empty() ? "output.txt" : outFile, ofstream::out);
        oStr << state;

        for (int i = 1; i < commSize; ++i) {
            int recvSize;
            MPICHECK(MPI_Recv(&recvSize, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &status));

            string temp(recvSize, 0);
            MPICHECK(MPI_Recv(&temp[0], recvSize, MPI_CHAR, i, 0, MPI_COMM_WORLD,
&status));
            oStr << temp;
        }

        oStr << "\n";
```

```
        oStr.close();
    } else {
        const int sendSize = state.length();
        MPICHECK(MPI_Send(&sendSize, 1, MPI_INT, MASTER, 0, MPI_COMM_WORLD));
        MPICHECK(MPI_Send(&state[0], sendSize, MPI_CHAR, MASTER, 0, MPI_COMM_WORLD));
    }

    //---------------------------------------------------------------------------

    MPICHECK(MPI_Finalize());
    return 0;
}
```