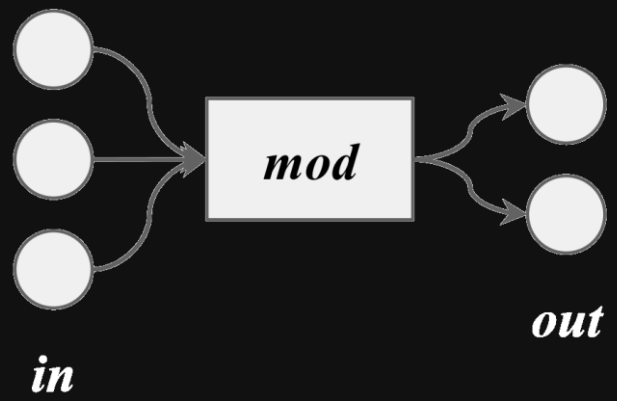
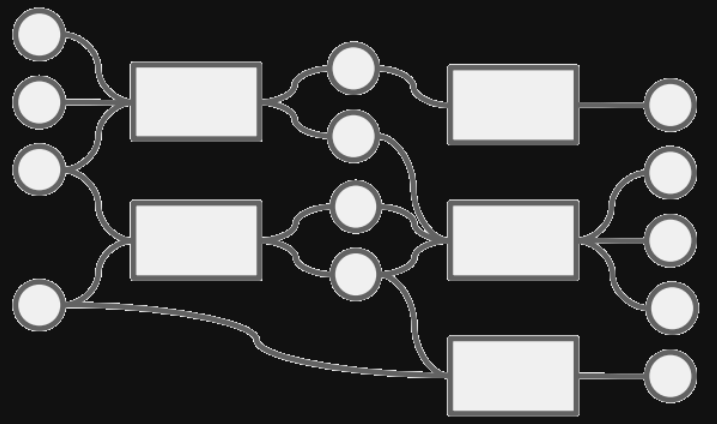
Компилятор системы LuNA

Кондратов Антон
НГУ ФИТ
Перепёлкин Владислав Александрович

Фрагментированный алгоритм

Язык **LuNA** – описывает рекурсивно-перечислимое множество триплетов **<in, mod, out>**



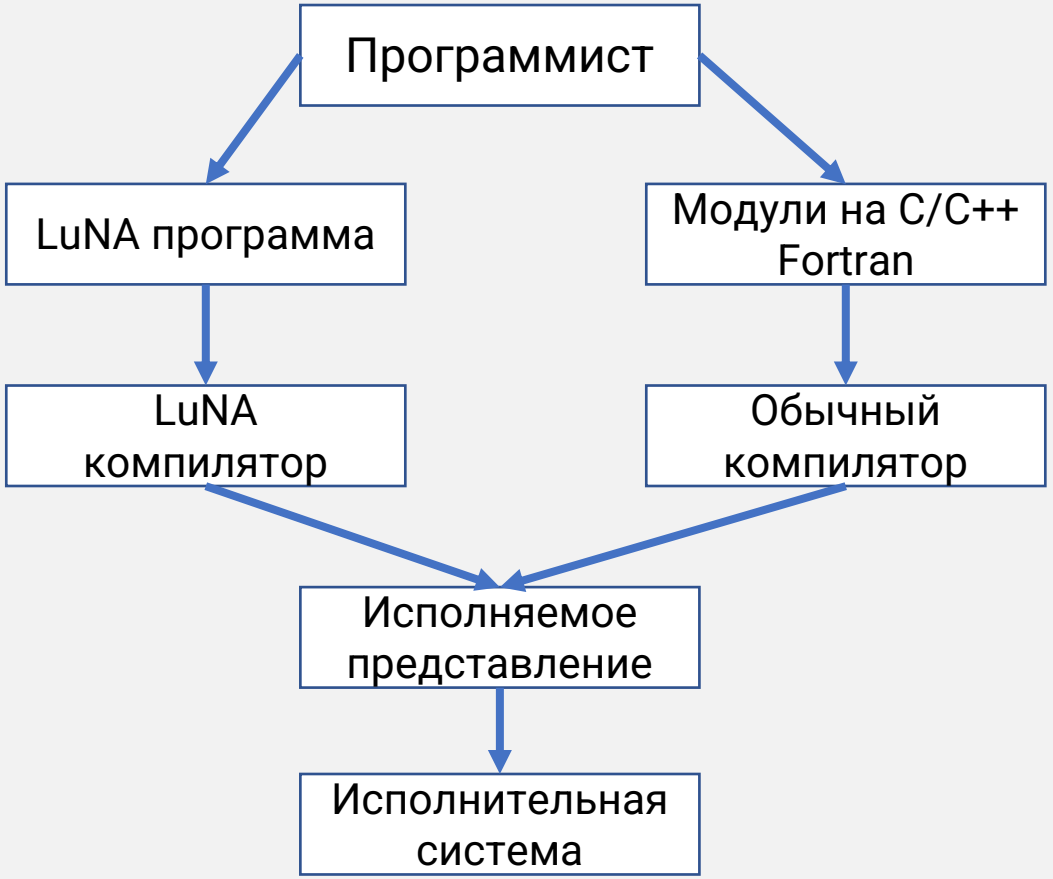


Цель — автоматизировать конструирование (синтез) эффективных параллельных программ, реализующих заданный численный алгоритм для класса суперкомпьютеров

Задачи

- написать компилятор, создающий MPI программу
- определить «правила» для компилятора
- проигрыш ручному написанию кода ≈15%

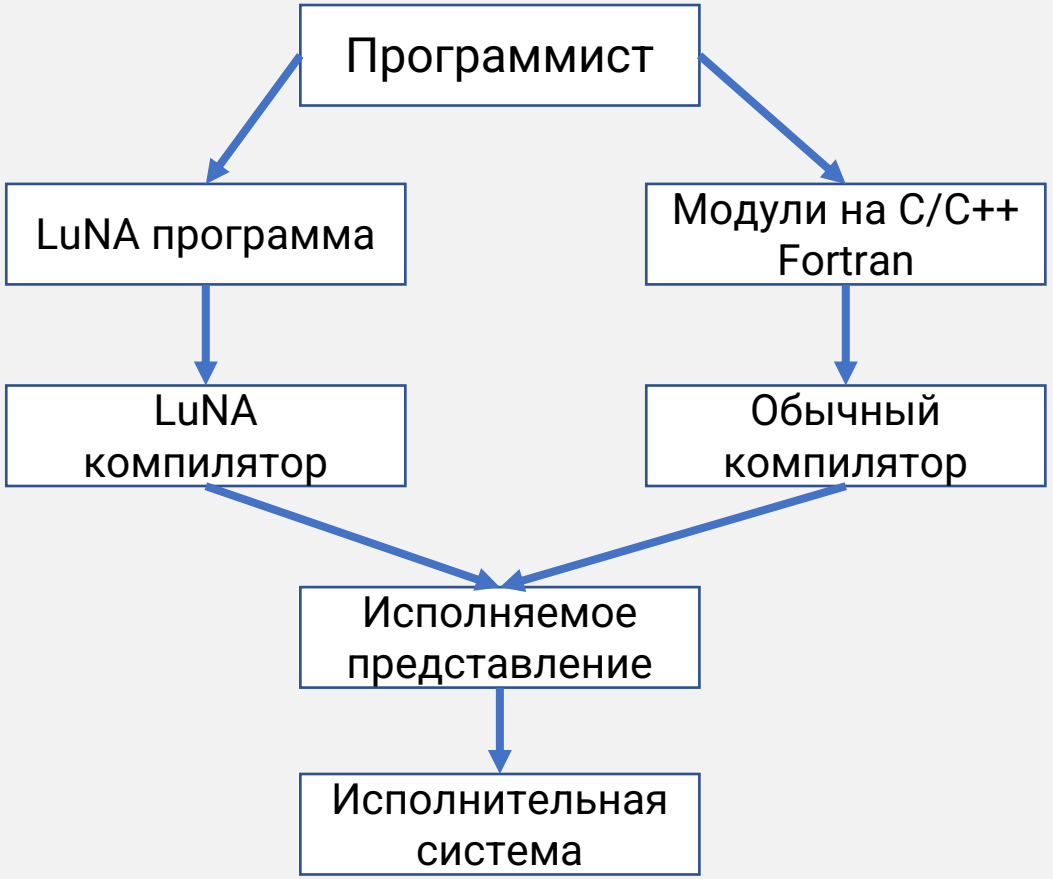
LuNA v6



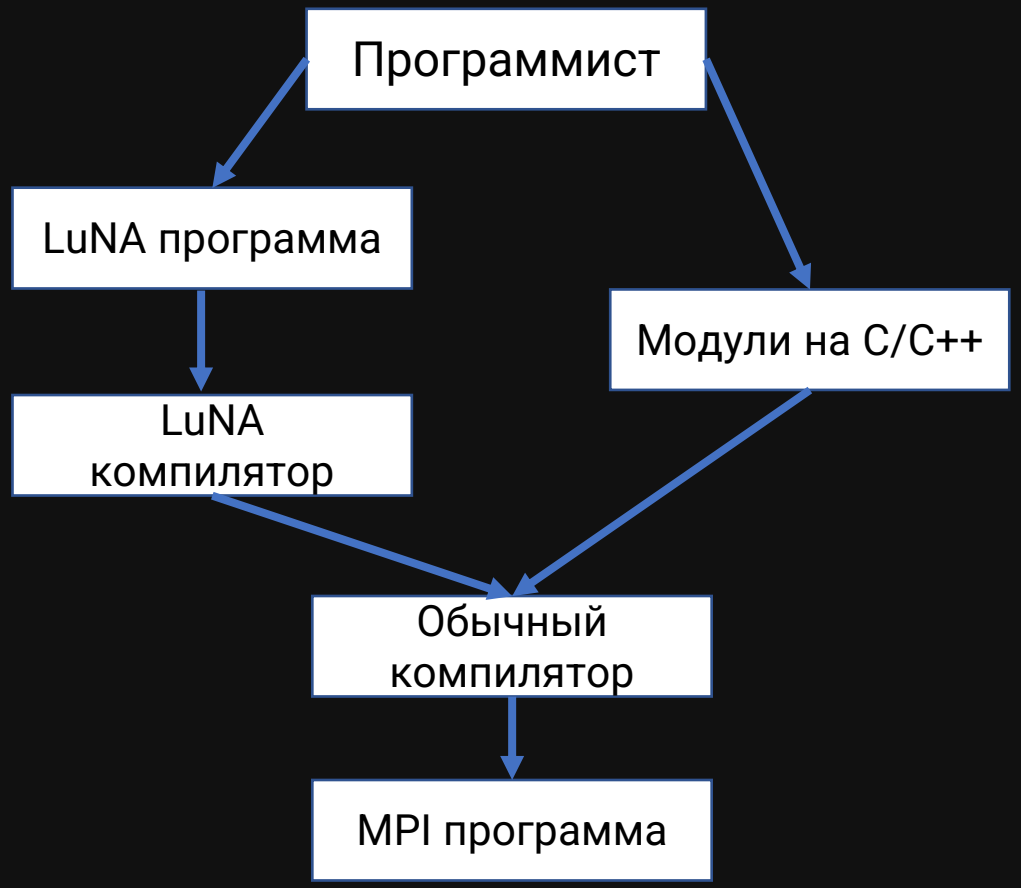
LuNA vX

?

LuNA v6



LuNA vX



Исключается исполнительная система

LuNA и C++

```
void c_print(){
    cout<<"Hello world";
}

import c_print() as pr
int;

sub main()
{
    print();
}
```

MPI (C++)

```
#include <iostream>
#include <mpi.h>
#include<LuNA>
void c_print(){
    std::cout<<"Hello world";
}

int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commSize);

    c_print();

    MPI_Finalize();
    return 0;
}
```

LuNA и C++

```
import c_init(int, name) as init;
import c_iprint(int) as iprint;

sub main()
{
    df x;
    init(7, x);
    iprint(x);
}
```

MPI (C++)

```
void c_print(int n) {
    std::cout << n;
}
void c_init(int val, DF& df) {
    df.setValue(val);
}

int main(int argc, char** argv) {
    int rank, commSize;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commSize);
    MPI_Status status;
    int num;
    if (!rank) {
        DF x;
        c_init(7, x);
        assert(x.get_type() == TYPE_INT);
        num = x.getValue<int>();
        MPI_Send(&num, 1, MPI_INT, commSize - 1, 0, MPI_COMM_WORLD);
    }
    if (rank == commSize - 1) {
        MPI_Recv(&num, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        c_print(num);
    }
    MPI_Finalize();
    return 0;
}
```

LuNA и C++

```
import c_init(int, name) as init;
import c_iprint(int) as iprint;

sub main()
{
    df x;
    init(7, x[4]);
    iprint(x[2+2]);
    init(8, x[7]);
    iprint(x[x[2+2]]);
}
```

MPI (C++)

```
int main(int argc, char** argv) {
    int rank, commSize;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &commSize);
    map<string, DF> dfSet = { {"x_4", DF()}, {"x_7", DF()}
};

        if (!rank) {
            c_init(7, dfSet["x_4"]);
            c_print(dfSet["x_4"].getValue<int>());

          int k = dfSet["x_4"].getValue<int>();
          string d = "x_" + k;

            c_init(7, dfSet[d]);
            c_print(dfSet[d].getValue<int>());
        }
    MPI_Finalize();
    return 0;
}
```

LuNA и C++

```
#define FG_SIZE 40
#define FG_COUNT 10
sub calc_mat(name A, name B, name C, int i, int j, int N){
    df Ctmp, Csum;
    for k=0..N-1{
        cf f[i][j][k]: mult_mat(A[i][k], B[k][j], Ctmp[k]);
    }
    if N>1
        sum_mat(Ctmp[0], Ctmp[1], Csum[1]);
    if N==1
        copy_mat(Ctmp[0], Csum[0]);
    for k=2..N-1
        cf sum[k]: sum_mat(Ctmp[k], Csum[k-1], Csum[k]);
    copy_mat(Csum[N-1], C);
}
sub main(){
    df A, B, C, N, M;
    init($FG_COUNT, N);
    init($FG_SIZE, M);
    for i=0..N-1
        for j=0..N-1
        {
            cf initA[i][j]: init_mat(0, i*M, j*M, M, M, A[i][j]);
            cf initB[i][j]: init_mat(1, i*M, j*M, M, M, B[i][j]);
            cf calc[i][j]: calc_mat(A, B, C[i][j], i, j, N);
        }
}
```

MPI (C++)

```
int main(int argc, char** argv) {
    char fileO[10] = "in.txt";
    FILE * f_in;
    int i, j, C1, C2, rows, cols
        int rank, size;
    int * matrix, * vector, * result, * matrix1, * rez;
    int * counts, * countsrez;
    int * disp, * disprez;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (!rank) {
        if ((f_in  = fopen(fileO, "r")) == NULL){
            printf ("\n Cannot open %s file.\n", fileO); return 0;
        }
        fscanf(f_in, "%d", &rows);
        fscanf(f_in, "%d", &cols);
        matrix = new int(cols * rows);
        vector = new int(cols);
        result = new int(rows);
        for (i = 0; i < cols * rows; i++)
            fscanf(f_in, "%d", matrix + i);
        for (i = 0; i < cols; i++)
            fscanf(f_in, "%d", vector + i);
    }
    MPI_Bcast(&rows, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Bcast(&cols, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (rank) vector = new int(cols);
    MPI_Bcast(vector, cols, MPI_INT, 0, MPI_COMM_WORLD);
    counts = new int(size); disp = new int(size);
    countsrez = new int(size); disprez = new int(size);
    matrix1 = new int(cols * rows); rez = new int(rows);
C1 = rows / size; C2 = rows % size;
    for (i = 0; i < size - C2; i++) {
        counts[i] = C1 * cols;
        countsrez[i] = C1;
        disprez[i] = i * C1;
        disp[i] = i * C1 * cols;
    }
    counts[size - C2] = (C1 + 1) * cols; countsrez[size - C2] = C1 + 1;
}
```

●●●

Заключение

- Сбор подходов к ручному переводу ФА* в MPI
- Ручная компиляция ряда задач: сумма двух векторов, скалярное произведение векторов, произведение квадратных матриц и другие более простые алгоритмы

ФА – фрагментированный алгоритм

Заключение и дальнейшие планы

- Сбор подходов к ручному переводу ФА* в MPI
- Ручная компиляция ряда задач: сумма двух векторов, скалярное произведение векторов, произведение квадратных матриц и другие более простые алгоритмы

- Расширение ряда задач ручной компиляции, выделение общих подходов в разных задачах
- Определение подходов для автоматизации перевода ФА* в MPI
- Создание прототипа компилятора

ФА – фрагментированный алгоритм

Благодарю за внимание