ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

**Отчет по лабораторной работе №3**

**по дисциплине: «Реализация алгоритма для систем с распределенной памятью»**

студента очного отделения

3 курса 12001801 группы

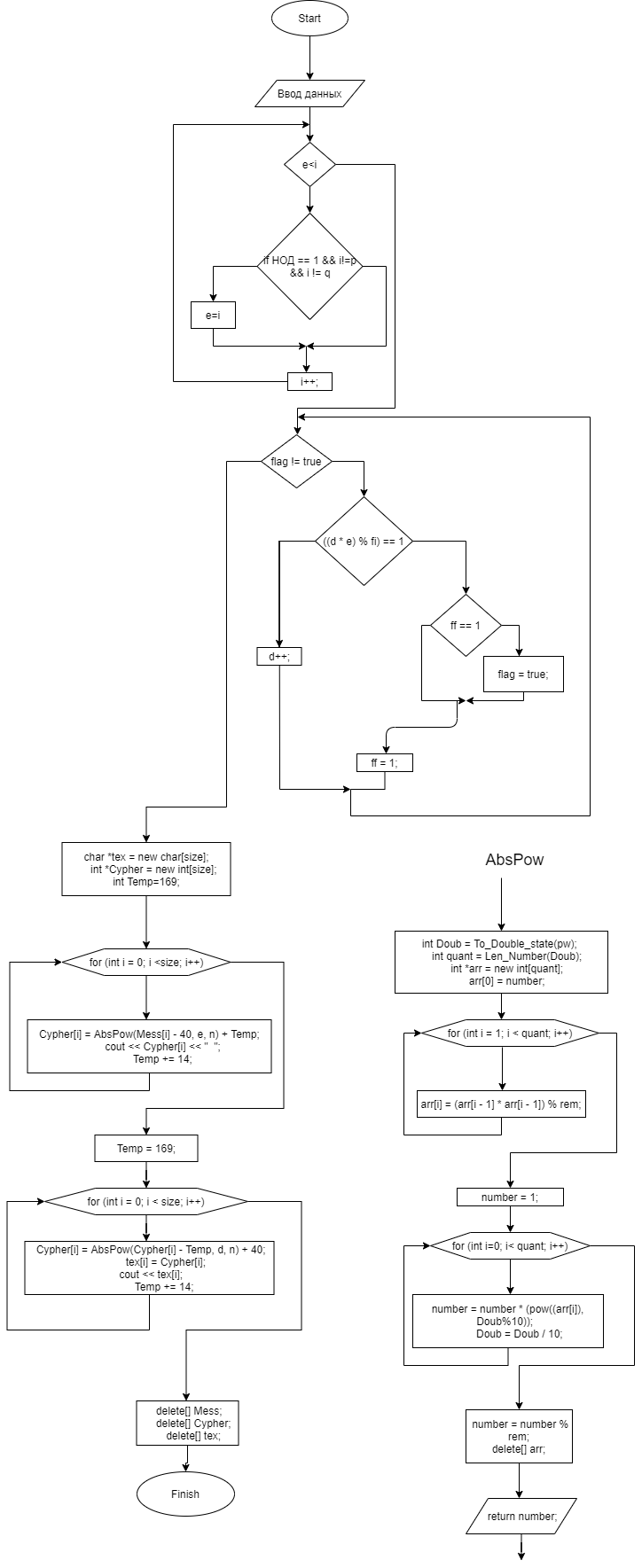
Капустина Виктора Сергеевича

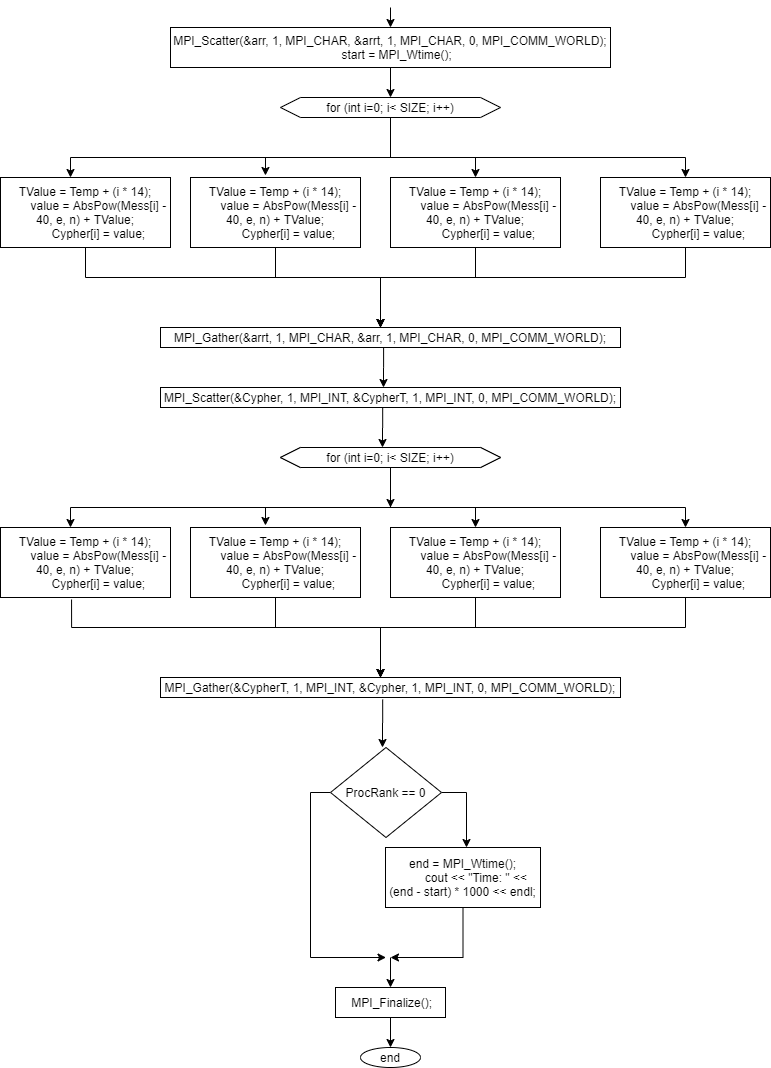
Проверил(а):

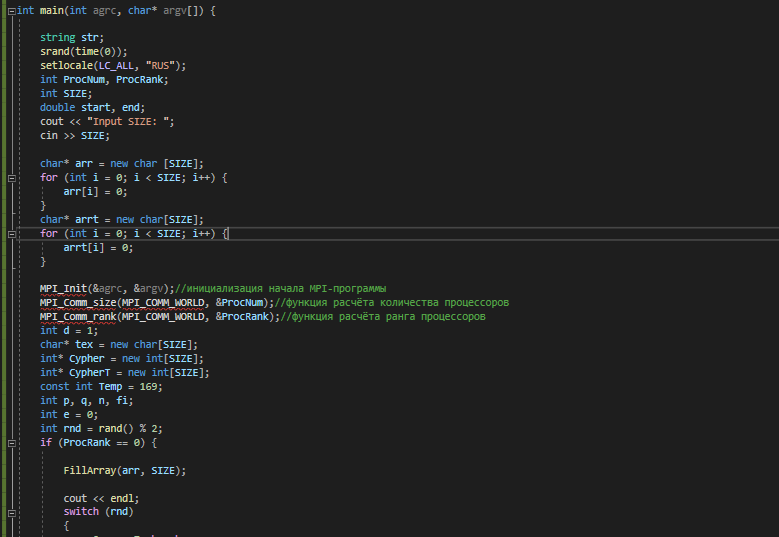
Петров Денис Васильевич

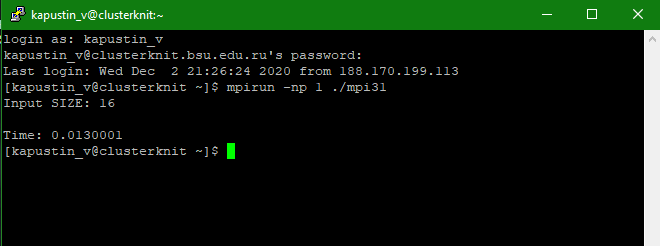
Белгород 2020

**16 Вариант  
Блок схема**





  
Рис. 1. Процесс разработки программного обеспечения

  
Рис. 2. Работа приложения

## Таблица 1. Зависимость времени вычислений от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Время расчета на 1 ядре, сек. | Время расчета на 2 ядрах, сек. | Время расчета на 3 ядрах, сек. | Время расчета на 4 ядрах, сек. | Время расчета на 5 ядрах, сек. | Время расчета на 6 ядрах, сек. |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 0,013 | 0,101 | 0,268 | 0,601 | 0,361 | 0,44 |
| **160** | 0,042 | 0,076 | 0,081 | 0,082 | 0,076 | 0,099 |
| **1600** | 4,001 | 2,156 | 2,816 | 4,449 | 3,247 | 2,4 |
| **16000** | 222,623 | 131,645 | 96,517 | 74,782 | 83,39 | 59,681 |
| **160000** | 2363,162 | 1878,501 | 834,1918 | 661,1423 | 586,9057 | 522,2603 |

## Таблица 1.1 Зависимость времени вычислений от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Время расчета на 7 ядрах, сек. | Время расчета на 8 ядрах, сек. | Время расчета на 9 ядрах, сек. | Время расчета на 10 ядрах, сек. | Время расчета на 11 ядрах, сек. | Время расчета на 12 ядрах, сек. |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 0.0013781 | 0.0015013 | 0.001402 | 0.0018002 | 0.002013 | 0.0019007 |
| **160** | 0.00161 | 0.001749 | 0.00200 | 0.001969 | 0.0020748 | 0.002402 |
| **1600** | 0.04879 | 0.043913 | 0.04046 | 0.040994 | 0.0356269 | 0.0335501 |
| **16000** | 3.00767 | 2.64789 | 2.47325 | 2.30505 | 2.186121 | 2.0244736 |
| **160000** | 28.6018 | 26.0009 | 23.2767 | 21.9137 | 20.1236 | 19.1762 |

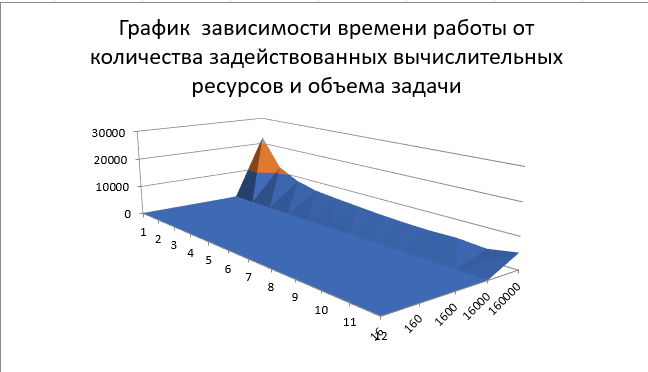
  
Рис. 3 Графика зависимости времени расчетов от количества задействованных вычислительных ресурсов и объема задачи

Таблица 2. Зависимость ускорения от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Коэффициент ускорения K на 1 ядре | Коэффициент ускорения K на 2 ядрах | Коэффициент ускорения K на 3 ядрах | Коэффициент ускорения K на 4 ядрах | Коэффициент ускорения K на 5 ядрах | Коэффициент ускорения K на 6 ядрах |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 1 | 0,16239316 | 0,06620209 | 0,03362832 | 0,05307263 | 0,04222222 |
| **160** | 1 | 0,54666667 | 0,52564103 | 0,56164384 | 0,46067416 | 0,40196078 |
| **1600** | 1 | 1,64032587 | 1,43156772 | 0,8658353 | 1,23262932 | 1,68352843 |
| **16000** | 1 | 1,81173952 | 2,35880726 | 3,00325832 | 2,76158418 | 3,81591298 |
| **160000** | 1 | 1,86927239 | 2,68501315 | 3,38234106 | 3,80804708 | 4,26499774 |

Таблица 2.1 Зависимость ускорения от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Коэффициент ускорения K на 7 ядрах | Коэффициент ускорения K на 8 ядрах | Коэффициент ускорения K на 9 ядрах | Коэффициент ускорения K на 10 ядрах | Коэффициент ускорения K на 11 ядрах | Коэффициент ускорения K на 12 ядрах |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 0,03853955 | 0,09178744 | 0,11949686 | 0,04871795 | 0,03442029 | 0,02695035 |
| **160** | 0,34166667 | 0,48809524 | 0,63076923 | 0,87234043 | 0,44086022 | 0,63076923 |
| **1600** | 0,91814865 | 2,46601347 | 1,33123967 | 2,66865474 | 1,02703392 | 1,31687377 |
| **16000** | 3,85187378 | 4,52569327 | 2,45736459 | 3,64106706 | 3,99006274 | 3,96511486 |
| **160000** | 4,8140335 | 5,27730035 | 5,48336051 | 5,26322419 | 6,22039873 | 4,28962767 |

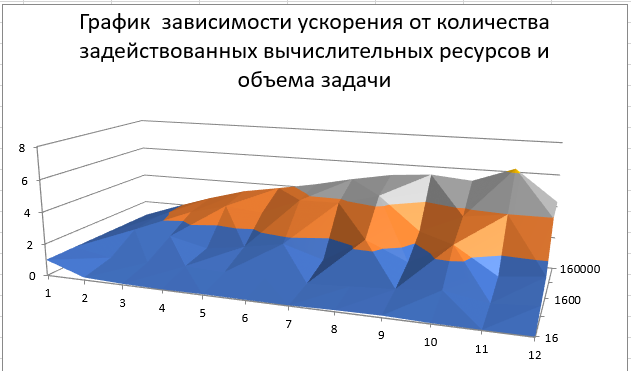


Рис. 4 График зависимости ускорения от количества задействованных вычислительных ресурсов и объема задачи

Таблица 3. Зависимость эффективности от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Коэффициент ускорения K на 1 ядре | Коэффициент ускорения K на 2 ядрах | Коэффициент ускорения K на 3 ядрах | Коэффициент ускорения K на 4 ядрах | Коэффициент ускорения K на 5 ядрах | Коэффициент ускорения K на 6 ядрах |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 1 | 0,0811965812 | 0,0220673635 | 0,084070796 | 0,0106145251 | 0,7037037 |
| **160** | 1 | 0,273333333 | 0,175213675 | 0,140410959 | 0,0921348315 | 0,0669934641 |
| **1600** | 1 | 0,820162933 | 0,47718924 | 0,216458826 | 0,246525865 | 0,280588071 |
| **16000** | 1 | 0,905869761 | 0,786269086 | 0,750814579 | 0,552316836 | 0,635985496 |
| **160000** | 1 | 0,934636193 | 0,895004384 | 0,84,5585266 | 0,761609416 | 0,710832957 |

Таблица 3.1 Зависимость эффективности от объема исходных данных и количества задействованных вычислительных ядрах

| **Объем задачи** | Коэффициент ускорения K на 7 ядрах | Коэффициент ускорения K на 8 ядрах | Коэффициент ускорения K на 9 ядрах | Коэффициент ускорения K на 10 ядрах | Коэффициент ускорения K на 11 ядрах | Коэффициент ускорения K на 12 ядрах |
| --- | --- | --- | --- | --- | --- | --- |
| **16** | 0,55056505 | 0,1147343 | 0,132774284 | 0,48717949 | 0,31291173 | 0,22458629 |
| **160** | 0,488095238 | 0,610119048 | 0,700854701 | 0,672340426 | 0,400782014 | 0,5316410 |
| **1600** | 0,131164094 | 0,308251684 | 0,17915519 | 0,256865474 | 0,9336672 | 0,11739481 |
| **16000** | 0,550267683 | 0,565711659 | 0,27304051 | 0,364106706 | 0,362732977 | 0,330426238 |
| **160000** | 0,687719072 | 0,659662544 | 0,609262279 | 0,526322419 | 0,565490794 | 0,357468973 |

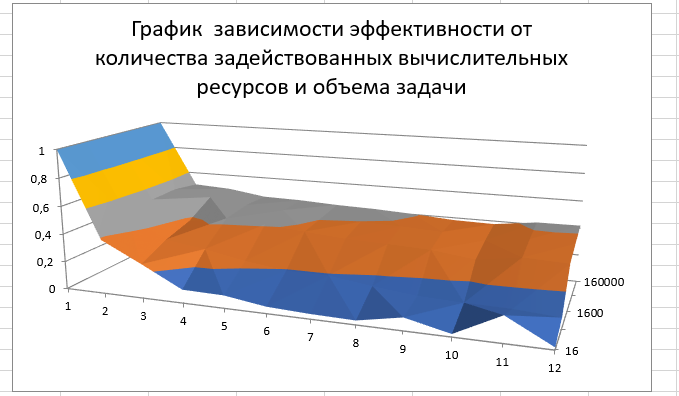


Рис. 5 График зависимости эффективности от количества задействованных вычислительных ресурсов и объема задачи

**Листинг программы**

#include <iostream>

#include <cstdlib>

#include <cctype>

#include <string>

#include <bitset>

#include <cmath>

#include <time.h>

#include <ctime>

#include <string>

#include <iomanip>

#include <mpi.h>

using namespace std;

int gcd(int x, int y)

{

return y ? gcd(y, x % y) : x;

}

void FillArray(char\* arr, int SIZE) {

for (int i = 0; i < SIZE; i++) {

arr[i] = 65 + rand() % 57;

}

}

void PrintArray(int\* arr, int SIZE) {

for (int i = 0; i < SIZE; i++) {

cout << arr[i] << "\t";

cout << endl;

}

}

int main(int agrc, char\* argv[]) {

string str;

srand(time(0));

setlocale(LC\_ALL, "RUS");

int ProcNum, ProcRank;

int SIZE;

double start, end,time;

char\* arr = new char [SIZE];

for (int i = 0; i < SIZE; i++) {

arr[i] = 0;

}

char\* arrt = new char[SIZE];

for (int i = 0; i < SIZE; i++) {

arrt[i] = 0;

}

MPI\_Init(&agrc, &argv);//инициализация начала MPI-программы

MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);//функция расчёта количества процессоров

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);//функция расчёта ранга процессоров

cout << "Input SIZE: ";

cin >> SIZE;

char\* tex = new char[SIZE];

int\* Cypher = new int[SIZE];

int\* CypherT = new int[SIZE];

int d = 1;

const int Temp = 169;

int p, q, n, fi;

int e = 0;

int rnd = rand() % 2;

if (ProcRank == 0) {

FillArray(arr, SIZE);

cout << endl;

switch (rnd)

{

case 0: p = 7; break;

case 1: p = 13; break;

}

q = 17;

n = abs(p \* q);

fi = (p - 1) \* (q - 1);

int y = 2;

while (e < 1) {

if ((gcd(y, fi) == 1) && (y != p) && (y != q))

{

e = y;

}

y++;

}

bool flag = false;

int ff = 0;

while (flag != true)

{

if (((d \* e) % fi) == 1)

{

if (ff == 1)

{

flag = true;

}

ff = 1;

}

else d++;

}

}

MPI\_Scatter(&arr, 1, MPI\_CHAR, &arrt, 1, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

start = MPI\_Wtime();

for (int i = 0; i < SIZE; i++)

{

int TValue = 169 + (i \* 14);

// abs pow

int number = arrt[i] - 40;

int pw = e;

int rem = n;

//Double state

int num1 = pw;

int h = 0, g = 1;

while (num1)

{

h += (num1 % 2) \* g;

num1 = num1 / 2;

g = g \* 10;

}

int Doub = h;

//

// Len number

int num = Doub;

int t = 1;

while ((num / 10) != 0)

{

t++;

num = num / 10;

}

int quant = t;

//

int\* arr1 = new int[quant];

arr1[0] = number;

for (int j = 1; j < quant; j++)

{

arr1[j] = (arr1[j - 1] \* arr1[j - 1]) % rem;

}

number = 1;

for (int k = 0; k < quant; k++)

{

number = number \* (pow((arr1[k]), Doub % 10));

Doub = Doub / 10;

}

number = number % rem;

delete[] arr1;

arrt[i] = number + TValue;

}

MPI\_Gather(&arrt, 1, MPI\_CHAR, &arr, 1, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

if (ProcRank == 0) {

for (int i = 0; i < SIZE; i++)

{

Cypher[i] = arr[i];

}

}

MPI\_Scatter(&Cypher, 1, MPI\_INT, &CypherT, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

for (int i = 0; i < SIZE; i++)

{

int TValue = 169 + (i \* 14);

// abs pow

int number = CypherT[i] - TValue;

int pw = d;

int rem = n;

//Double state

int num1 = pw;

int h = 0, g = 1;

while (num1)

{

h += (num1 % 2) \* g;

num1 = num1 / 2;

g = g \* 10;

}

int Doub = h;

//

// Len number

int num = Doub;

int t = 1;

while ((num / 10) != 0)

{

t++;

num = num / 10;

}

int quant = t;

//

int\* arr1 = new int[quant];

arr1[0] = number;

for (int j = 1; j < quant; j++)

{

arr1[j] = (arr1[j - 1] \* arr1[j - 1]) % rem;

}

number = 1;

for (int k = 0; k < quant; k++)

{

number = number \* (pow((arr1[k]), Doub % 10));

Doub = Doub / 10;

}

number = number % rem;

delete[] arr1;

CypherT[i] = number + 40;

}

MPI\_Gather(&CypherT, 1, MPI\_INT, &Cypher, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (ProcRank == 0) {

end = MPI\_Wtime();

time = (end - start);

cout << "Time: " << time << endl;

}

MPI\_Finalize();

}