###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ИЗМЕРЕНИЕ СТЕПЕНИ АССОЦИАТИВНОСТИ КЭШ-ПАМЯТИ»

студента 2 курса, группы 19212

**Хомченко Станислава Евгеньевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Ажбаков Артём Альбертович

# **ЗАДАНИЕ**

1. Написать программу на языке C или C++, которая реализует итерационный алгоритм решения системы линейных алгебраических уравнений вида Ax=b в соответствии с выбранным вариантом. Здесь A – матрица размером N×N, x и b – векторы длины N. Тип элементов – double.
2. Программу распараллелить с помощью MPI с разрезанием матрицы A по строкам на близкие по размеру, возможно не одинаковые, части. Соседние строки матрицы должны располагаться в одном или в соседних MPI-процессах. Реализовать два варианта программы:

* Вариант 1: векторы x и b дублируются в каждом MPI-процессе,
* Вариант 2: векторы x и b разрезаются между MPI-процессами аналогично матрице A. Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

1. Замерить время работы двух вариантов программы при использовании различного числа процессорных ядер: 1,2, 4, 8, 16. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ε подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
2. Выполнить профилирование двух вариантов программы с помощью MPE при использовании 16-и ядер.
3. На основании полученных результатов сделать вывод о целесообразности использования одного или второго варианта программы

# **ОПИСАНИЕ РАБОТЫ**

В качестве алгоритма для реализации был выбран метод простой итерации:

# 1. Написание программы на языке Си(последовательная).

2. Написание двух программ при помощи MPI(параллельные).

3. Замер времени, нахождение ускорения и эффективности.

4. Профилирование

5. Вывод о проделанной работе.

# **Приложение 1.** *Листинг программы (векторы x и b дублируются в каждом MPI-процессе)*

#include <iostream>  
#include <cmath>  
#include <ctime>  
#include <mpi.h>  
  
#define N 4096   
#define t 10e-6  
#define eps 10e-9  
  
void init(int \*\*perProcess, int \*startLine, double \*\*matrix, double \*\*b, double \*\*x, int size, int rank) {  
 \*perProcess = new int[size]();  
 for(int i = 0, tmp = size - (N % size); i < size; ++i) {  
 (\*perProcess)[i] = i < tmp ? (N / size) : (N / size + 1);  
 if(i < rank) {  
 \*startLine += (\*perProcess)[i];  
 }  
 }  
  
 \*matrix = new double[(\*perProcess)[rank] \* N];  
 for(int i = 0; i < (\*perProcess)[rank]; ++i) {  
 for(int j = 0; j < N; ++j) {  
 (\*matrix)[i \* N + j] = ((\*startLine) + i) == j ? 2 : 1;  
 }  
 }  
  
 \*b = new double[(\*perProcess)[rank]];  
 for(int i = 0; i < (\*perProcess)[rank]; ++i) {  
 (\*b)[i] = N + 1;  
 }  
  
 \*x = new double[N]();  
}  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 int rank = 0, size = 0;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
 int startLine = 0;  
 int \*perProcess = 0;  
 double \*matrix = 0, \*b = 0, \*x = 0;  
 init(&perProcess, &startLine, &matrix, &b, &x, size, rank);  
  
 double normB = 0, startTime = 0;  
 if(rank == 0) {  
 startTime = MPI\_Wtime();  
  
 for(int i = 0; i < N; ++i) {  
 normB += b[i] \* b[i];  
 }  
 normB = sqrt(normB);  
 }  
  
 double \*processX = new double[perProcess[rank]];  
 int keepCalc = 1;  
 while(keepCalc) {  
 double processAnswer = 0;  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 double sum = 0;  
 for(int j = 0; j < N; ++j) {  
 sum += matrix[i \* N + j] \* x[j];  
 }  
 sum -= b[i];   
 processX[i] = x[i + startLine] - t \* sum;  
 processAnswer += sum \* sum;  
 }  
  
 if(rank != 0) {  
 MPI\_Send(processX, perProcess[rank], MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);  
 MPI\_Send(&processAnswer, 1, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);  
 } else {  
 for(int i = startLine, c = startLine + perProcess[rank]; i < c; ++i) {  
 x[i] = processX[i - startLine];  
 }  
  
 double sum = processAnswer;  
 for(int i = 1, currentLine = perProcess[rank]; i < size; ++i) {  
 MPI\_Status status;  
 MPI\_Recv(&x[currentLine], perProcess[i], MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, &status);  
 currentLine += perProcess[i];  
  
 double tmp;  
 MPI\_Recv(&tmp, 1, MPI\_DOUBLE, i, 1, MPI\_COMM\_WORLD, &status);  
 sum += tmp;  
 }  
 sum = sqrt(sum);  
  
  
 keepCalc = sum / normB >= eps;  
 }  
 MPI\_Bcast(&keepCalc, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
 MPI\_Bcast(x, N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);  
 }  
  
 if(rank == 0) {  
 double endTime = MPI\_Wtime();  
 std::cout << "Size: " << size << ", time: " << (endTime - startTime) << std::endl;  
  
 bool correctAnswer = true;  
 for(int i = 0; i < N; ++i) {  
 if(fabs(fabs(x[i]) - 1) >= eps) {  
 correctAnswer = false;  
 break;  
 }  
 }

if(correctAnswer)  
 std::cout << "Accepted." << std::endl;  
 else  
 std::cout << "WA." << std::endl;  
 }  
  
 delete[] processX;  
 delete[] x;  
 delete[] b;  
 delete[] matrix;  
 delete[] perProcess;  
 MPI\_Finalize();  
 return 0;  
}

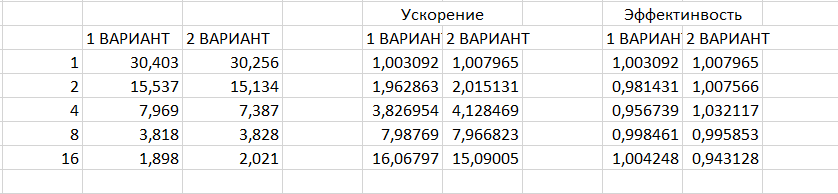
# **Приложение 2.** *Листинг программы(векторы x и b разрезаются между MPI-процессами аналогично матрице A)*

#include <iostream>  
#include <cmath>  
#include <ctime>  
#include <mpi.h>  
#include <cstdio>  
  
#define N 4096  
#define t 10e-6  
#define eps 10e-9  
  
void init(int \*\*perProcess, int \*startLine, double \*\*matrix, double \*\*b, double \*\*x, int size, int rank) {  
 \*perProcess = new int[size]();  
 for(int i = 0, tmp = size - (N % size); i < size; ++i) {  
 (\*perProcess)[i] = i < tmp ? (N / size) : (N / size + 1);  
 if (i < rank) {  
 \*startLine += (\*perProcess)[i];  
 }  
 }  
  
 \*matrix = new double[(\*perProcess)[rank] \* N];  
 for(int i = 0; i < (\*perProcess)[rank]; ++i) {  
 for(int j = 0; j < N; ++j) {  
 (\*matrix)[i \* N + j] = ((\*startLine) + i) == j ? 2 : 1;  
 }  
 }  
  
 \*b = new double[(\*perProcess)[rank]];  
 for(int i = 0; i < (\*perProcess)[rank]; ++i) {  
 (\*b)[i] = N + 1;  
 }  
  
 \*x = new double[N / size + 1]();  
}  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 int rank = 0, size = 0;  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
 int startLine = 0;  
 int \*perProcess = 0;  
 double \*matrix = 0, \*b = 0, \*x = 0;  
 init(&perProcess, &startLine, &matrix, &b, &x, size, rank);  
  
 double startTime = 0, normB = 0;  
 if(rank != 0) {  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 normB += b[i] \* b[i];  
 }  
 MPI\_Send(&normB, 1, MPI\_DOUBLE, 0, 4, MPI\_COMM\_WORLD);  
 } else {  
 startTime = MPI\_Wtime();  
  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 normB += b[i] \* b[i];  
 }  
  
 for(int i = 1; i < size; ++i) {  
 double tmp;  
 MPI\_Status status;  
 MPI\_Recv(&tmp, 1, MPI\_DOUBLE, i, 4, MPI\_COMM\_WORLD, &status);  
 normB += tmp;  
 }  
  
 normB = sqrt(normB);  
 }  
  
 double \*tmpSum = new double[perProcess[rank]];  
 double \*tmpX = new double[N / size + 1]();  
 int keepCalc = 1;  
 while(keepCalc) {  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 tmpSum[i] = 0;  
 }  
  
 for(int i = 0, currentCrds = startLine; i < size; ++i) {  
  
 for(int j = 0; j < perProcess[rank]; ++j) {  
 for(int k = currentCrds, c = currentCrds + perProcess[i]; k < c; ++k) {  
 tmpSum[j] += matrix[j \* N + k] \* x[k - currentCrds];  
 }  
 }  
  
 MPI\_Status status;  
 MPI\_Sendrecv(x, N / size + 1, MPI\_DOUBLE, (rank - 1 + size) % size, 0,  
 tmpX, N / size + 1, MPI\_DOUBLE, (rank + 1) % size, 0, MPI\_COMM\_WORLD, &status);  
  
 std::swap(x, tmpX);  
 currentCrds = (currentCrds + perProcess[i]) % N;  
 }  
  
 double processAnswer = 0;  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 tmpSum[i] -= b[i];  
 x[i] = x[i] - tmpSum[i] \* t;  
 processAnswer += tmpSum[i] \* tmpSum[i];  
 }  
  
 if(rank != 0) {  
 MPI\_Send(&processAnswer, 1, MPI\_DOUBLE, 0, 2, MPI\_COMM\_WORLD);  
 } else {  
 double sum = processAnswer;  
 for(int i = 1; i < size; ++i) {  
 MPI\_Status status;  
 double tmp;  
 MPI\_Recv(&tmp, 1, MPI\_DOUBLE, i, 2, MPI\_COMM\_WORLD, &status);  
 sum += tmp;  
 }  
 sum = sqrt(sum);  
  
 keepCalc = sum / normB >= eps;  
 }  
  
 MPI\_Bcast(&keepCalc, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);  
 }  
  
 if(rank != 0) {  
 MPI\_Send(x, perProcess[rank], MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD);  
 } else {  
 double \*fullX = new double[N];  
 for(int i = 0; i < perProcess[rank]; ++i) {  
 fullX[i] = x[i];  
 }  
  
 for(int i = 1, currentLine = perProcess[rank]; i < size; ++i) {  
 MPI\_Status status;  
 MPI\_Recv(&fullX[currentLine], perProcess[i], MPI\_DOUBLE, i, 1, MPI\_COMM\_WORLD, &status);  
 currentLine += perProcess[i];  
 }  
  
 double endTime = MPI\_Wtime();  
 std::cout << "Size: " << size << ", time: " << (endTime - startTime) << std::endl;  
  
 bool correctAnswer = true;  
 for(int i = 0; i < N; ++i) {  
 if(fabs(fabs(fullX[i]) - 1) >= eps) {  
 correctAnswer = false;  
 break;  
 }  
 }  
  
 if(correctAnswer)  
 std::cout << "Accepted." << std::endl;  
 else  
 std::cout << "WA." << std::endl;  
  
 delete[] fullX;  
 }  
  
 delete[] tmpX;  
 delete[] x;  
 delete[] b;  
 delete[] matrix;  
 delete[] perProcess;  
 MPI\_Finalize();  
 return 0;  
}

# **Листинг 4.** *Графики программ*

# Время исполнения программы на Си на 1 ядре составляет 30,497 sec.

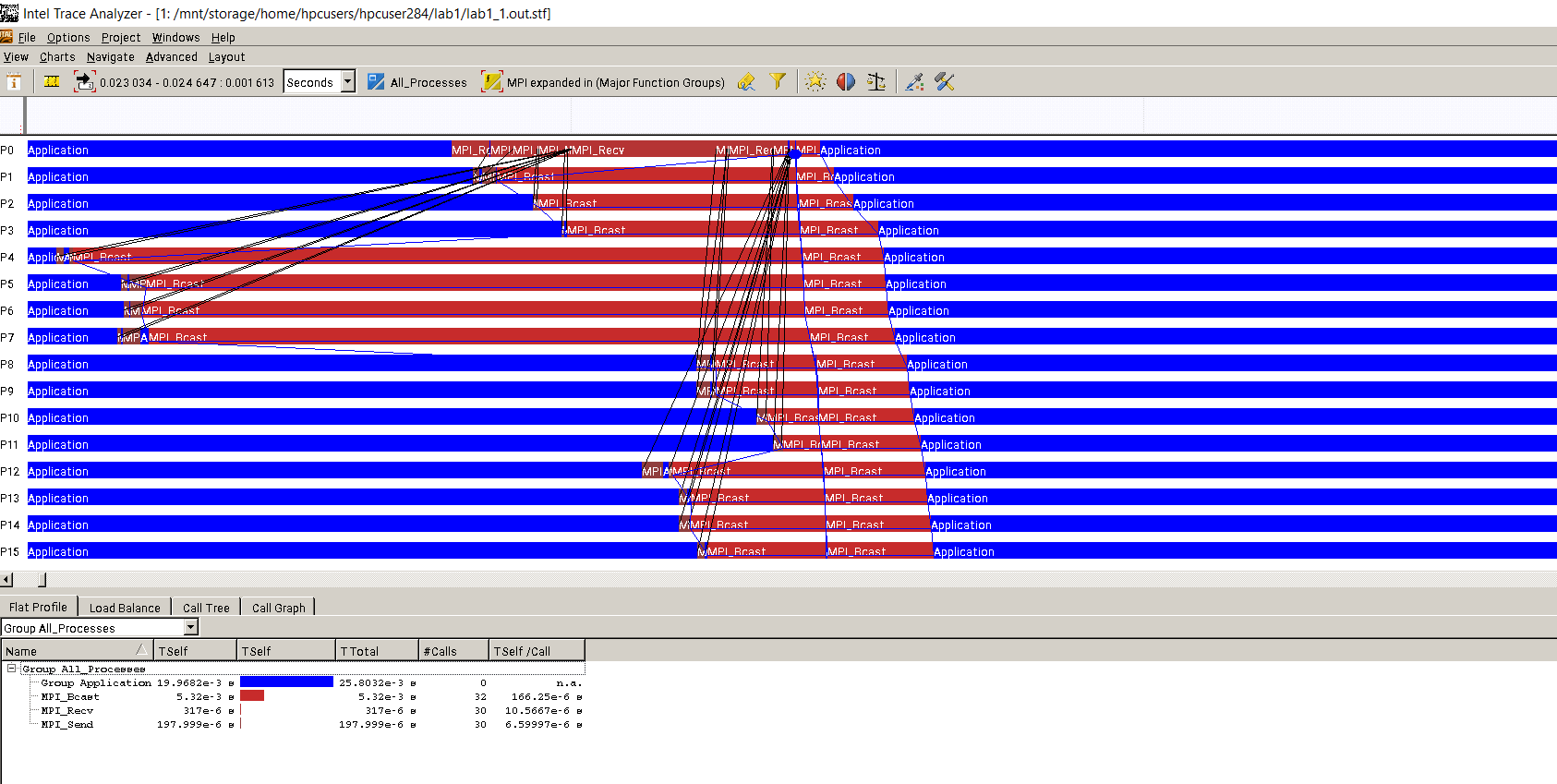
ускорение: Sp = T1 / Tp, где T1 – время работы последовательной программы на 1 ядре. Tp - время работы параллельной программы на p ядрах. Эффективность: Ep = (Sp / p) \* 100%



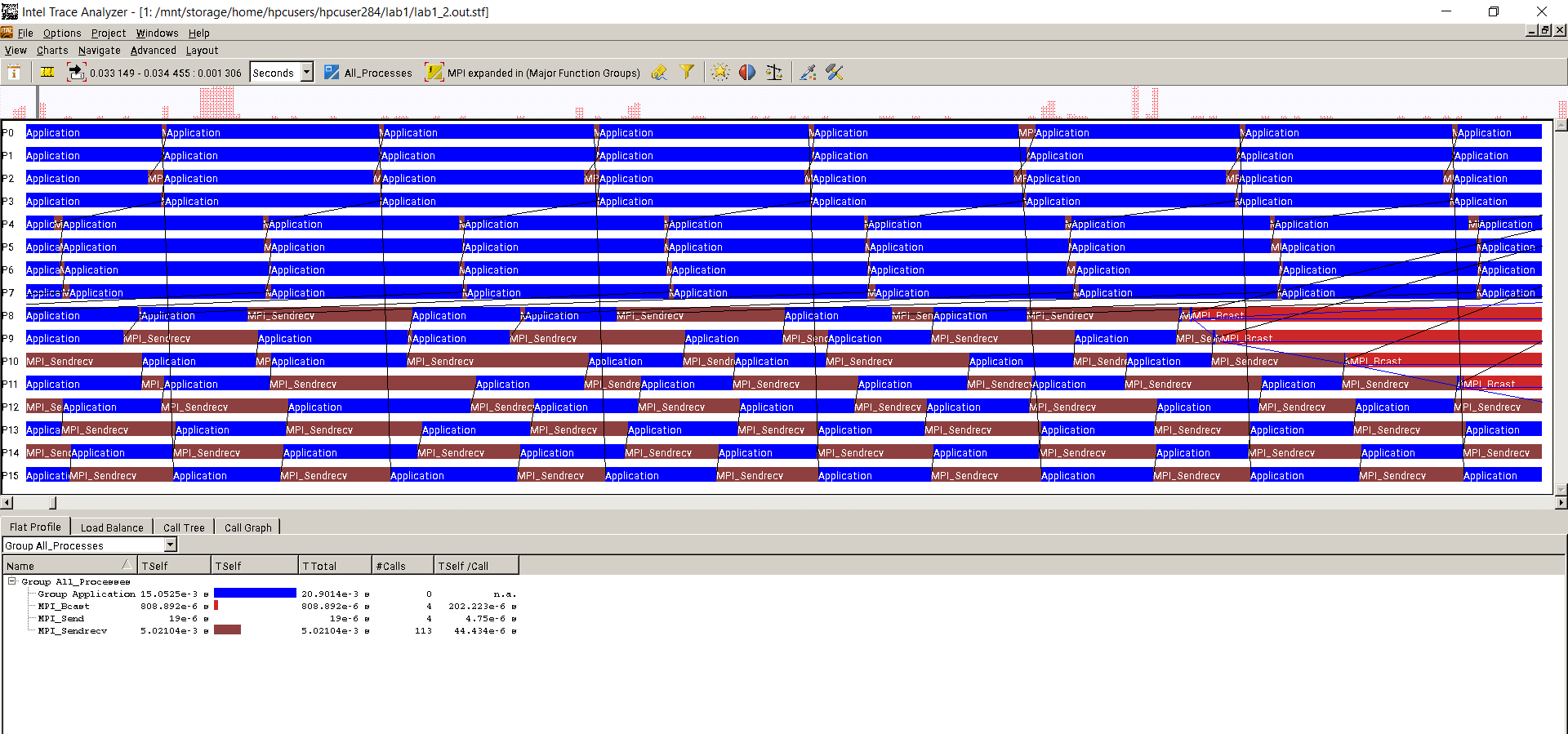
­

**Профилирование**

1 вариант:



2 вариант:



# **Вывод о проделанной работе**

В ходе лабораторной работы я познакомился с MPI. В результате время, потраченное на реализацию алгоритма при помощи MPI быстрее, чем обычная программа на Си, без использования программного интерфейса(MPI), данная программа выдает хорошую скорость.