###### 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 на языке C*

#include <stdlib.h>  
#include <math.h>  
#include <stdio.h>  
#include <sys/time.h>  
  
#define N 4096  
#define EPS 0.000000001  
#define TAU 0.01  
  
struct timeval tv1,tv2,dtv;  
void time\_start() { gettimeofday(&tv1, NULL); }  
  
long time\_stop(){  
 gettimeofday(&tv2, NULL);  
 dtv.tv\_sec= tv2.tv\_sec -tv1.tv\_sec;  
 dtv.tv\_usec=tv2.tv\_usec-tv1.tv\_usec;  
 if(dtv.tv\_usec<0) { dtv.tv\_sec--; dtv.tv\_usec+=1000000; }  
 return dtv.tv\_sec\*1000+dtv.tv\_usec/1000;  
}  
  
  
void fulling(double\* A, double\* b, double\* x, size\_t size){  
 for (int i = 0; i < size; ++i)  
 for (int j = 0; j < size; ++j)  
 A[i\*size + j] = (i == j) ? 2.0 : 1.0;  
  
 for (int i = 0; i < size; ++i){  
 b[i] = size + 1;  
 x[i] = 0;  
 }  
}  
  
void VECTxSCAL(double\* res, double\* vect, double scal, size\_t size){  
 for (int i = 0; i < size; ++i)  
 res[i] = vect[i] \* scal;  
}  
  
void MATxVECT(double\* res, double\* mat, double\* vect, size\_t size){  
 for (int i = 0; i < size; ++i){  
 double tmp = 0;  
 for (int j = 0; j < size; ++j)  
 tmp += mat[i \* size + j] \* vect[j];  
 res[i] = tmp;  
 }  
}  
  
void VECTsubVECT(double\* res, double\* vect1, double\* vect2, size\_t size){  
 for (int i = 0; i < size; ++i)  
 res[i] = vect1[i] - vect2[i];  
}  
  
void approx(double\* res, double\* xn, double\* b, double\* A, size\_t size){  
 double tau = TAU;  
 MATxVECT(res, A, xn, size);  
 VECTsubVECT(res, res, b, size);  
 VECTxSCAL(res, res, tau, size);  
 VECTsubVECT(res, xn, res, size);  
}  
  
double norm(double\* vect, size\_t size){  
 double res = 0;  
 for (int i = 0; i < size; ++i)  
 res += vect[i] \* vect[i];  
 return sqrt(res);  
}  
  
double condition(double\* xn, double\* b, double\* A, size\_t size){  
 double\* tmp = (double\*)malloc(sizeof(double) \* N);  
  
 MATxVECT(tmp, A, xn, size);  
 VECTsubVECT(tmp, tmp, b, size);  
  
 double n = norm(tmp, size) / norm(b, size);  
 free(tmp);  
 return n;  
}  
  
int main(int argc, char \*\*argv){  
 time\_start();  
 size\_t size = N;  
  
 double\* A = (double\*)malloc(sizeof(double) \* N \* N);  
 double\* b = (double\*)malloc(sizeof(double) \* N);  
 double\* x = (double\*)malloc(sizeof(double) \* N);  
 double\* next\_x = (double\*)malloc(sizeof(double) \* N);  
 fulling(A, b, x, size);  
 double E = condition(x, b, A, size);  
 while (E >= EPS){  
 approx(next\_x, x, b, A, size);  
 double\* tmp = next\_x;  
 next\_x = x;  
 x = tmp;  
 E = condition(x, b, A, size);  
 }  
 long dt = time\_stop();  
 printf("time diff %ld ms\n", dt);  
 free(next\_x);  
 free(x);  
 free(b);  
 free(A);  
 return 0;  
}

# **Приложение 2.** *Листинг программы (векторы x и b дублируются в каждом MPI-процессе)*

#include <stdlib.h>  
#include <math.h>  
#include <stdio.h>  
#include <mpi.h>  
#include <time.h>  
#include <sys/time.h>  
  
#define N 4096  
#define EPS 0.000000001  
#define TAU 0.01  
  
struct timeval tv1,tv2,dtv;  
void time\_start() { gettimeofday(&tv1, NULL); }  
long time\_stop(){  
 gettimeofday(&tv2, NULL);  
 dtv.tv\_sec= tv2.tv\_sec -tv1.tv\_sec;  
 dtv.tv\_usec=tv2.tv\_usec-tv1.tv\_usec;  
 if(dtv.tv\_usec<0) { dtv.tv\_sec--; dtv.tv\_usec+=1000000; }  
 return dtv.tv\_sec\*1000+dtv.tv\_usec/1000;  
}  
  
void fulling(double\* A, double\* x, double\* b, size\_t size, int tid){  
 for(int i = 0; i < size; i++)  
 for (int j = 0; j < N; j++)  
 A[i\*N + j] = (tid\*size + i) == j ? 2.0:1.0;  
 for (int i = 0; i < N; ++i){  
 b[i] = N + 1;  
 x[i] = 0;  
 }  
}  
  
void VECTxSCAL( double\* res, double\* vect, double scal, size\_t size)  
{  
 for (int i = 0; i < size; ++i)  
 res[i] = vect[i] \* scal;  
}  
  
void VECTsubVECT( double\* res, size\_t sRes, double\* vect1, double\* vect2){  
 for(int i = 0; i < sRes; ++i)  
 res[i] = vect1[i] - vect2[i];  
}  
  
void MATxVECT( double\* res, size\_t sRes,  
 double\* vect, size\_t sVect,  
 double\* mat)  
{  
 for(int i = 0; i < sRes; ++i){  
 double tmp = 0;  
 for (int j = 0; j < sVect; ++j)  
 tmp += mat[i \* sRes + j] \* vect[j];  
 res[i] = tmp;  
 }  
}  
  
void approx( double\* res, size\_t sRes,  
 double\* xn, double\* b,  
 double\* A, size\_t sMatG,  
 int tid)  
{  
 int shift = tid \* sRes;  
 MATxVECT(res, sRes, xn, sMatG, A);  
 VECTsubVECT(res, sRes, res, b + shift);  
 VECTxSCAL(res, res, TAU, sRes);  
 VECTsubVECT(res, sRes, xn + shift, res);  
}  
  
double qNorm(double\* vect, size\_t size){  
 double res = 0;  
 for (int i = 0; i < size; ++i)  
 res += vect[i] \* vect[i];  
 return res;  
}  
  
  
double condition( double\* A, size\_t sMatV,  
 double\* xn, double\* b,  
 size\_t sVects, int tid)  
{  
 int shift = tid \* sMatV;  
  
 double\* tmp = (double\*)malloc(sizeof(double) \* sMatV);  
  
 MATxVECT(tmp, sMatV, xn, sVects, A);  
 VECTsubVECT(tmp, sMatV, tmp, b + shift);  
  
 double n1 = qNorm(tmp, sMatV);  
 double n2 = qNorm(b + shift, sMatV);  
  
 double sum1 = 0, sum2 = 0;  
  
 MPI\_Allreduce(&n1,&sum1, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);  
 MPI\_Allreduce(&n2,&sum2, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);  
  
 free(tmp);  
 return sqrt(sum1 / sum2);  
}  
  
int main(int argc, char \*\*argv){  
 int size\_proc, tid;  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size\_proc);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &tid);  
 if (N % size\_proc!=0) {  
 if (tid == 0)  
 printf("Invalid parameters\n");  
 MPI\_Finalize();  
 return 0;  
 }  
  
 time\_start();  
  
 size\_t size = N / size\_proc;  
  
 double\* A = (double\*)malloc(sizeof(double) \* N \* size);  
 double\* b = (double\*)malloc(sizeof(double) \* N);  
 double\* x = (double\*)malloc(sizeof(double) \* N);  
 double\* next\_x = (double\*)malloc(sizeof(double) \* N);  
  
 fulling(A, x, b, size, tid);  
  
 double E = condition(A, size, x, b, N, tid);  
 while (E >= EPS ){  
 approx(next\_x, size, x, b, A, N, tid);  
 MPI\_Allgather(next\_x, size, MPI\_DOUBLE, x, size, MPI\_DOUBLE, MPI\_COMM\_WORLD);  
 E = condition( A, size, x, b, N, tid);  
 }  
  
 long dt = time\_stop();  
 if(tid == 0) printf("time diff %ld ms \n",dt);  
  
 free(next\_x);  
 free(x);  
 free(b);  
 free(A);  
 MPI\_Finalize();  
 return 0;  
}

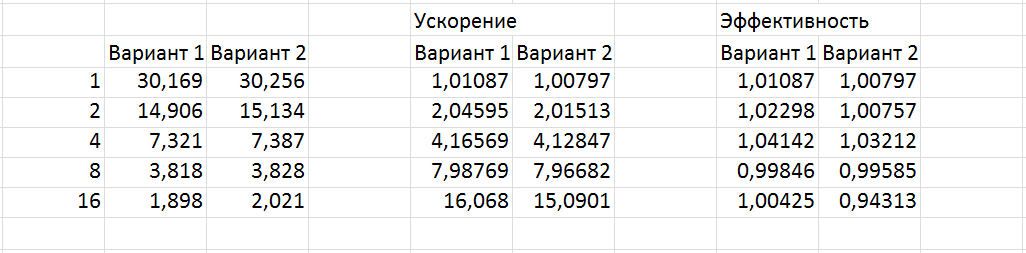
# **Приложение 3.** *Листинг программы(векторы x и b разрезаются между MPI-процессами аналогично матрице A)*

#include <stdlib.h>  
#include <math.h>  
#include <stdio.h>  
#include <mpi.h>  
#include <time.h>  
#include <sys/time.h>  
  
#define N 4096  
#define EPS 0.000000001  
#define TAU 0.01  
  
struct timeval tv1,tv2,dtv;  
void time\_start() { gettimeofday(&tv1, NULL); }  
long time\_stop(){  
 gettimeofday(&tv2, NULL);  
 dtv.tv\_sec= tv2.tv\_sec -tv1.tv\_sec;  
 dtv.tv\_usec=tv2.tv\_usec-tv1.tv\_usec;  
 if(dtv.tv\_usec<0) { dtv.tv\_sec--; dtv.tv\_usec+=1000000; }  
 return dtv.tv\_sec\*1000+dtv.tv\_usec/1000;  
}  
  
void fulling(double\* A, double\* x, double\* b, size\_t size, int tid){  
 for(int i = 0; i < size; i++)  
 for (int j = 0; j < N; j++)  
 A[i\*N + j] = (tid\*size + i) == j ? 2.0:1.0;  
 for (int i = 0; i < size; ++i){  
 b[i] = N + 1;  
 x[i] = 0;  
 }  
}  
  
void VECTxSCAL( double\* res, double\* vect, double scal, size\_t size){  
 for (int i = 0; i < size; ++i)  
 res[i] = vect[i] \* scal;  
}  
  
void VECTsubVECT( double\* res, size\_t sRes, double\* vect1, double\* vect2){  
 for(int i = 0; i < sRes; ++i)  
 res[i] = vect1[i] - vect2[i];  
}  
  
void MATxVECT( double\* res, size\_t sRes,  
 double\* vect, size\_t sVect,  
 double\* mat)  
{  
 for(int i = 0; i < sRes; ++i){  
 double tmp = 0;  
 for (int j = 0; j < sVect; ++j)  
 tmp += mat[i \* sRes + j] \* vect[j];  
 res[i] = tmp;  
 }  
}  
  
void approx( double\* res, size\_t sRes,  
 double\* xn, double\* b,  
 double\* A, size\_t sMatG)  
{  
 double\* X = (double\*)malloc(sizeof(double) \* N);  
 MPI\_Allgather(xn, sRes, MPI\_DOUBLE, X, sRes, MPI\_DOUBLE, MPI\_COMM\_WORLD);  
 MATxVECT(res, sRes, X, sMatG, A); // нужен целый векор xn и только часть res  
 VECTsubVECT(res, sRes, res, b); // нужна только часть b  
 VECTxSCAL(res, res, TAU, sRes); // только часть res  
 VECTsubVECT(res, sRes, xn, res); // только часть res, xn  
 free(X);  
}  
  
double qNorm(double\* vect, size\_t size){  
 double res = 0;  
 for (int i = 0; i < size; ++i)  
 res += vect[i] \* vect[i];  
 return res;  
}  
  
  
double condition( double\* A, size\_t sMatV,  
 double\* xn, double\* b)  
{  
 double\* tmp = (double\*)malloc(sizeof(double) \* sMatV);  
 double\* X = (double\*)malloc(sizeof(double) \* N);  
 MPI\_Allgather(xn, sMatV, MPI\_DOUBLE, X, sMatV, MPI\_DOUBLE, MPI\_COMM\_WORLD);  
  
 MATxVECT(tmp, sMatV, X, N, A); // нужен весь xn, часть tmp  
 VECTsubVECT(tmp, sMatV, tmp, b); // часть tmp, b  
  
 double n1 = qNorm(tmp, sMatV); // часть tmp  
 double n2 = qNorm(b, sMatV); // часть b  
  
 double sum1 = 0, sum2 = 0;  
  
 MPI\_Allreduce(&n1,&sum1, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);  
 MPI\_Allreduce(&n2,&sum2, 1, MPI\_DOUBLE, MPI\_SUM, MPI\_COMM\_WORLD);  
 free(X);  
 free(tmp);  
 return sqrt(sum1 / sum2);  
}  
  
int main(int argc, char \*\*argv){  
 int size\_proc, tid;  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size\_proc);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &tid);  
  
 if (N % size\_proc!=0) {  
 if (tid == 0)  
 printf("Invalid parameters\n");  
 MPI\_Finalize();  
 return 0;  
 }  
  
 time\_start();  
  
 size\_t size = N / size\_proc;  
  
 double\* A = (double\*)malloc(sizeof(double) \* N \* size);  
 double\* b = (double\*)malloc(sizeof(double) \* size);  
 double\* x = (double\*)malloc(sizeof(double) \* size);  
 double\* next\_x = (double\*)malloc(sizeof(double) \* size);  
  
 fulling(A, x, b, size, tid);  
  
 double E = condition(A, size, x, b);  
 while (E >= EPS ){  
 approx(next\_x, size, x, b, A, N);  
 double\* tmp = next\_x;  
 next\_x = x;  
 x = tmp;  
 E = condition( A, size, x, b);  
 }  
 long dt = time\_stop();  
 if(tid == 0) printf("time diff %ld ms \n",dt);  
  
 free(next\_x);  
 free(x);  
 free(b);  
 free(A);  
 MPI\_Finalize();  
 return 0;  
}

# **Листинг 4.** *Графики программ*

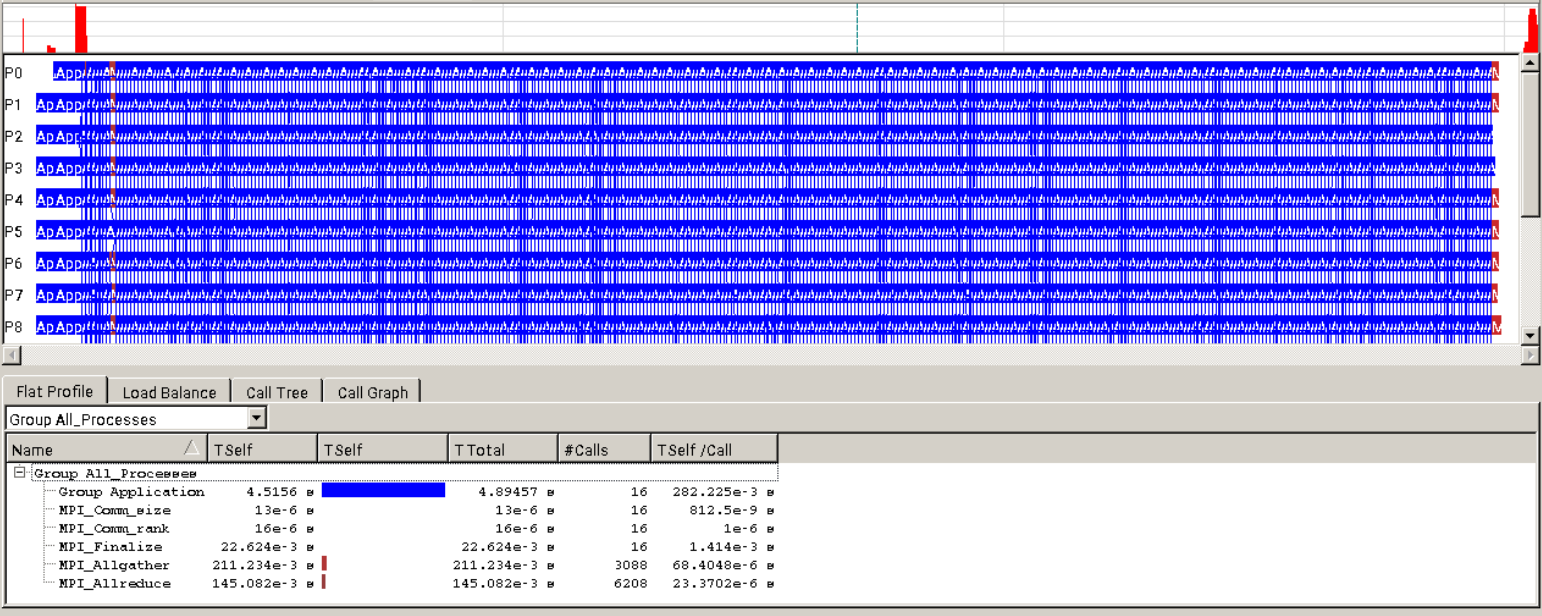
# Время исполнения программы на Си на 1 ядре составляет 30,497 sec.

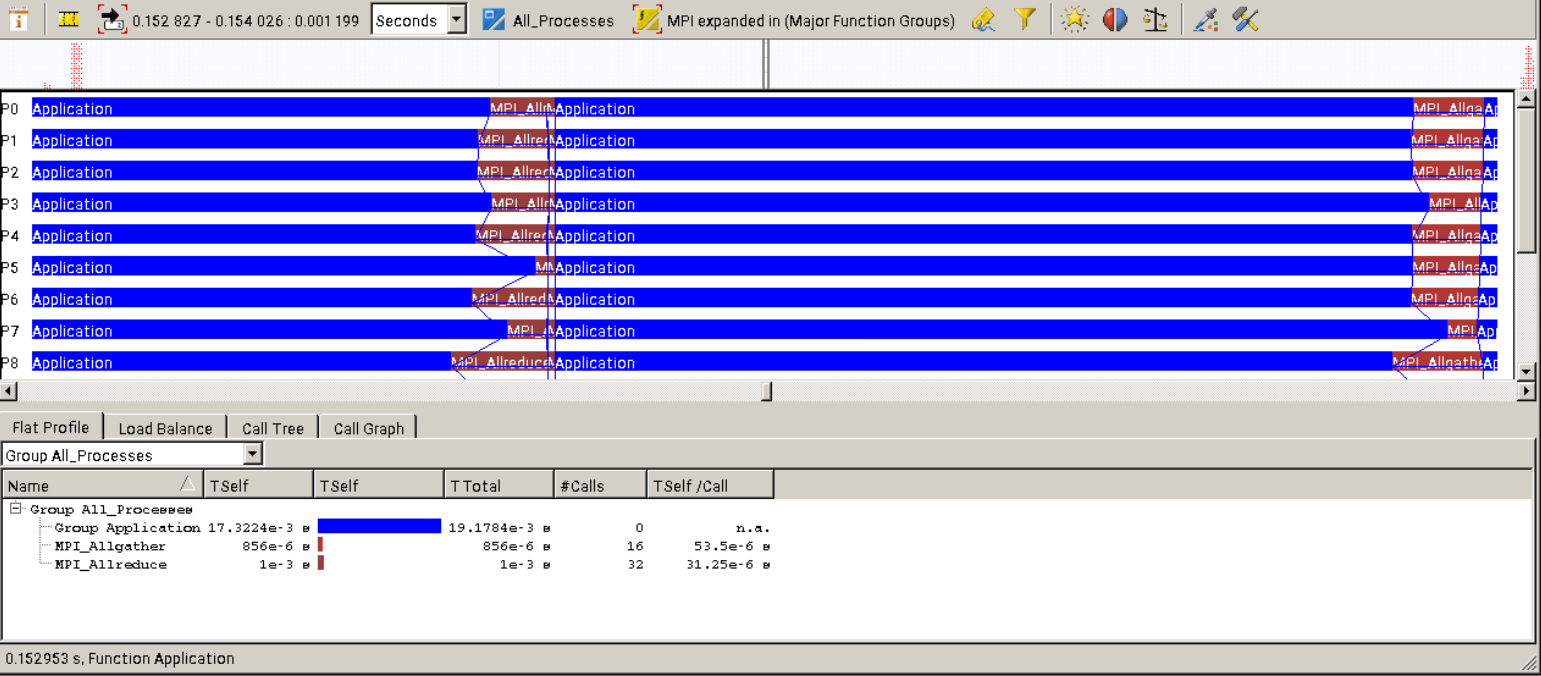
ускорение: Sp = T1 / Tp, где T1 – время работы последовательной программы на 1 ядре. Tp - время работы параллельной программы на p ядрах. Эффективность: Ep = (Sp / p) \* 100%



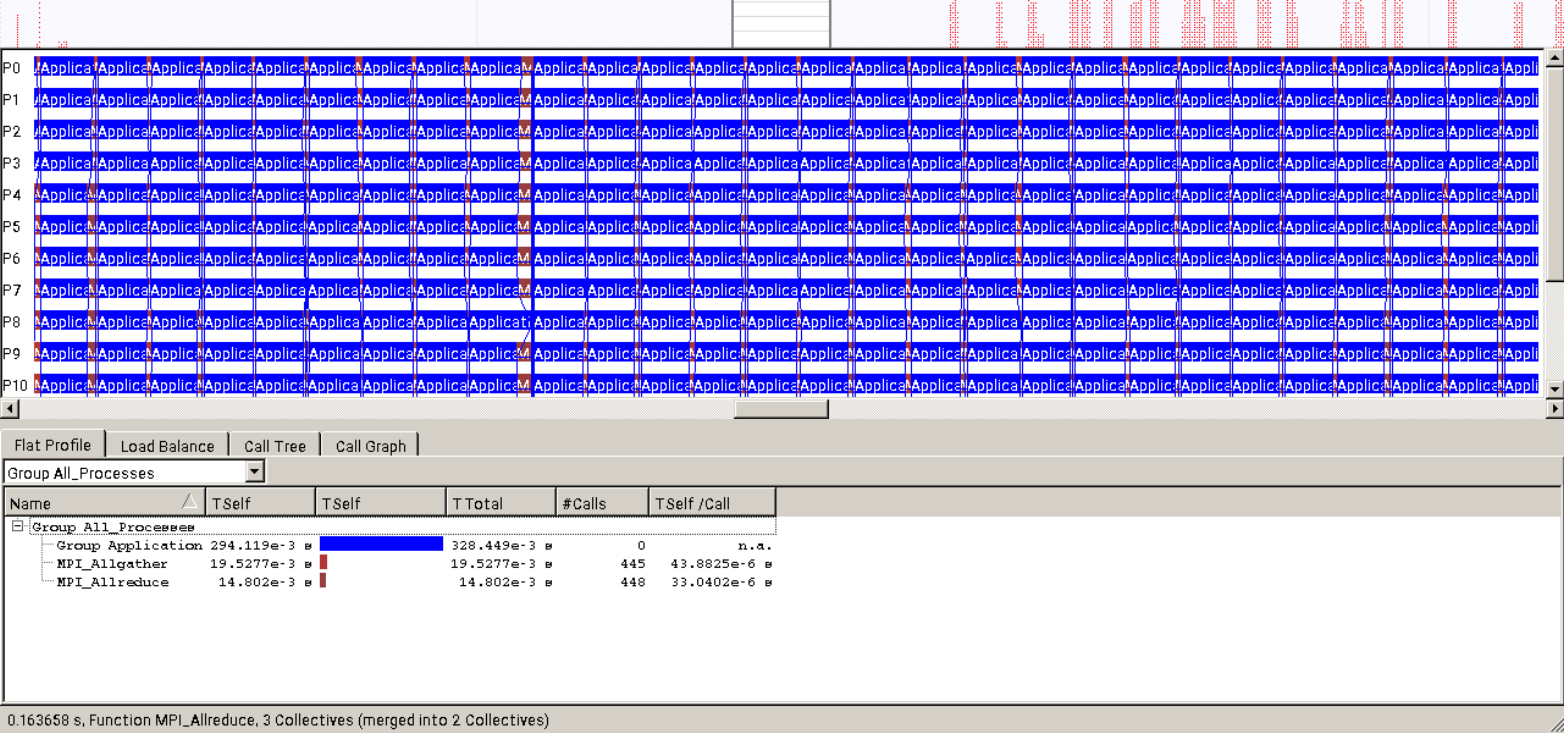
**Профилирование**

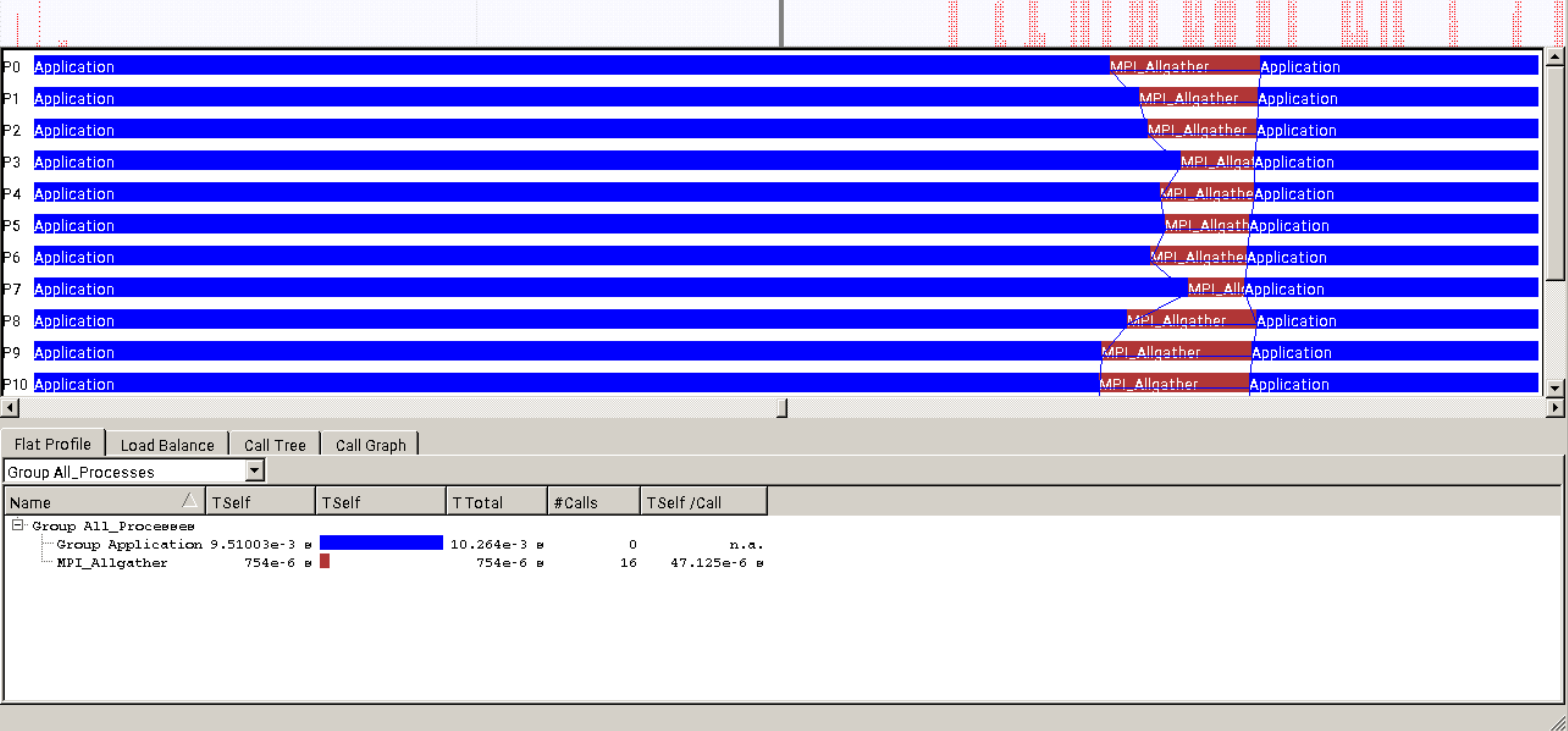
1 вариант:





2 вариант:





# **Вывод о проделанной работе**

В ходе лабораторной работы я познакомилась 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.