###### т 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 ПРАКТИЧЕСКОЙ РАБОТЫ

**Два вектора**

студента 2 курса, группы 22204

Соломенникова Николая Александровича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

доцент

А.Ю.Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 2](#__RefHeading___1)

[ЗАДАНИЕ 2](#__RefHeading___2)

[ОПИСАНИЕ РАБОТЫ 3](#__RefHeading___3)

[ЗАКЛЮЧЕНИЕ 5](#__RefHeading___6)

[ИСПОЛЬЗУЕМЫЕ КОМАНДЫ 5](#__RefHeading___5)

[ИСХОДНИКИ 5](#__RefHeading___7)

# ЦЕЛЬ

Ознакомиться с базовыми принципами работы параллельных программ и сравнить время работы при разном количестве запущенных процессов.

# ЗАДАНИЕ

I Написать 3 программы, каждая из которых рассчитывает число *s* по двум данным векторам *a* и *b* равной длины *N* в соответствии со следующим двойным циклом:

for (i = 0; i < N; i++)

for(j = 0; j < N; j++)

s += a[i] \* b[j];

a) последовательная программа

b) параллельная, использующая коммуникации типа точка-точка (MPI\_Send, MPI\_Recv)

c) параллельная, использующая коллективные коммуникации (MPI\_Scatter, MPI\_Reduce, MPI\_Bcast)

II Замерить время работы последовательной программы и параллельных на 2, 4, 8, 16, 24 процессах. Рекомендуется провести несколько замеров для каждого варианта запуска и выбрать минимальное время.

III Построить графики времени, ускорения и эффективности.

IV Составить отчет, содержащий исходные коды разработанных программ и построенные графики.

**Требования:**

- длину векторов выбирать таким образом, чтобы время работы последовательной программы было не менее 30 сек;

- в параллельных программах изначально оба вектора должны полностью инициализироваться на 0-м процессе. Для параллельного расчета 0-й процесс должен раздавать части одного из векторов остальным, а второй вектор передавать полностью каждому процессу;

- в параллельных программах результат *s* должен выводиться на экран 0-м процессом.

**Комментарий**

**Ускорение**: Sp = T1 / Tp, где *T1* - время работы **ПОСЛЕДОВАТЕЛЬНОЙ** программы. *Tp* - время работы параллельной программы на *p* процессах/потоках.

**Эффективность**: Ep = Sp / p \* 100%

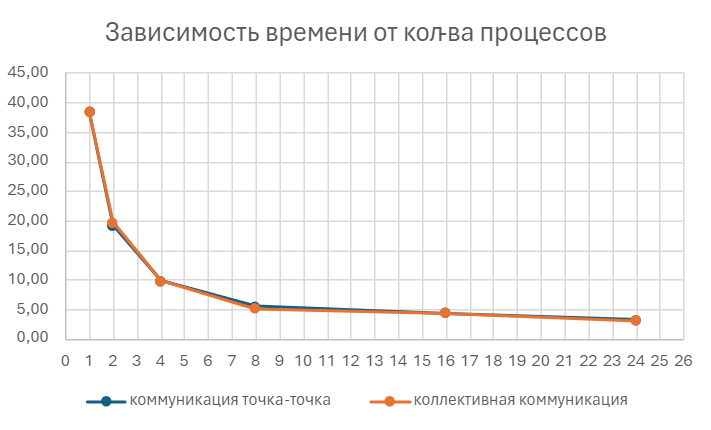
Выполнять работу можно на кафедральном сервере или вычислительном кластере НГУ.

# ОПИСАНИЕ РАБОТЫ

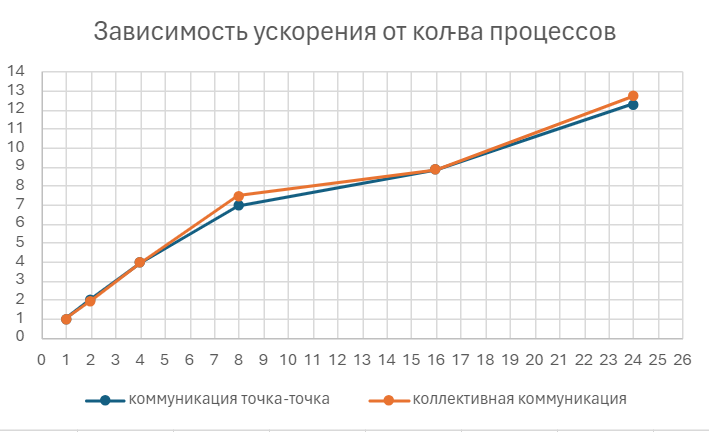
Реализацию программ можно посмотреть в исходниках.

Зависимость времени выполнения программы в зависимости от реализации (последовательная, параллельная с коммуникацией точка-точка или параллельная с коллективной коммуникацией):

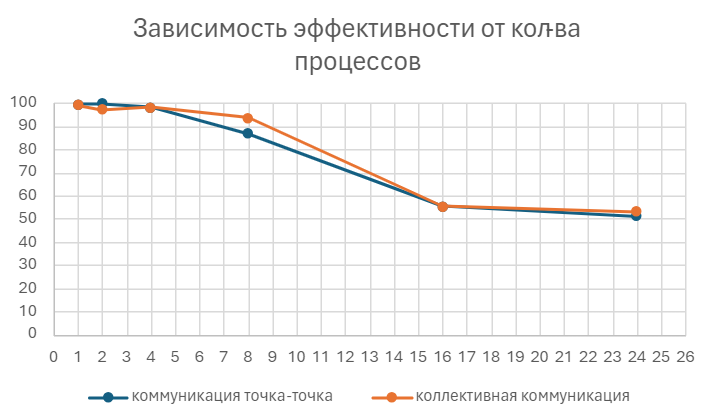
|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во процессов | Последовательная программа | Коммуникации точка-точка | Коллективные коммуникации |
| 1 | 38.1 | 38.4 | 38.5 |
| 2 | - | 19.1 | 19.6 |
| 4 | - | 9.7 | 9.7 |
| 8 | - | 5.5 | 5.1 |
| 16 | - | 4.3 | 4.3 |
| 24 | - | 3.1 | 3.0 |



Ускорения, рассчитываемые по формуле Sp = T1 / Tp, получились следующие:



Эффективности (в процентах):



# ЗАКЛЮЧЕНИЕ

Времена работы параллельной программы с коммуникацией точка-точка и с коллективной коммуникацией получились одинаковыми (с точностью до небольшой погрешности). Это вполне ожидаемый результат, так как коллективные методы основаны на базовых методах коммуникации, а процессы мало связаны между собой, и они почти не тратят время на коммуникацию.

Также очевидно по графикам, что распараллеливание программы улучшает время исполнения в разы.

# ИСПОЛЬЗУЕМЫЕ КОМАНДЫ

Для установки библиотеки mpich

$sudo apt update

$sudo apt install mpich

Для того, чтобы компилятор обнаружил mpich можно задать переменную окружения

$export CPLUS\_INCLUDE\_PATH=/usr/include/x86\_64-linux-gnu/mpich/

Либо можно задать нужный путь до библеотеки через флаг

$mpicxx prog.cpp -o prog.out -I/usr/include/x86\_64-linux-gnu/mpich/

Нужный путь до mpi.h ищется с помощью команды

$find /usr -name mpi.h

Также можно задать переменную окружения для оболочки bash навсегда, написав в конце файла ~/.bashrc следующую команду

export CPLUS\_INCLUDE\_PATH=$CPLUS\_INCLUDE\_PATH:/usr/include/x86\_64-linux-gnu/mpich/

И после этого, чтобы изменения вступили в силу написать

$source ~/.bashrc

Запуск программы:

$mpiexec -n 1 ./prog.out

или

$mpirun -np 1 ./prog.out

# ИСХОДНИКИ

// В файле settings.h хранится только размер векторов:

constexpr int N = int(1e5);

// prog1.cpp (последовательная программа)

#include <iostream>

#include <time.h>

#include "settings.h"

int main()

{

    // Инициализация данных

    double \*a = (double \*)malloc(sizeof(double) \* N);

    double \*b = (double \*)malloc(sizeof(double) \* N);

    for (int i = 0; i < N; ++i)

    {

        a[i] = 1.0;

        b[i] = 1.0;

    }

    double s = 0;

    time\_t startTime = clock();

    // Вычисления

    for (int j = 0; j < N; ++j)

    {

        for (int i = 0; i < N; ++i)

        {

            s += a[i] \* b[j];

        }

    }

    time\_t endTime = clock();

    std::cout << "Total time: " << double(endTime - startTime) / CLOCKS\_PER\_SEC << "\n";

    return 0;

}

// prog2.cpp (коммуникация точка-точка)

#include <iostream>

#include <mpi.h>

#include "settings.h"

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

{

    int size, rank;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    int vecPartSize = N / size; // Количество элементов вектора b, которые будут передаваться другим процессам

    if (rank == 0)

    {

        // Инициализация данных

        double \*a = (double \*)malloc(sizeof(double) \* N);

        double \*b = (double \*)malloc(sizeof(double) \* N);

        for (int i = 0; i < N; ++i)

        {

            a[i] = 1.0;

            b[i] = 1.0;

        }

        double s = 0;

        double startTime = MPI\_Wtime();

        // Распределение вектора a и части вектора b другим процессам

        for (int p = 1; p < size; ++p)

        {

            MPI\_Send(a, N, MPI\_DOUBLE, p, 1, MPI\_COMM\_WORLD);

            MPI\_Send(b + vecPartSize \* (p - 1), vecPartSize, MPI\_DOUBLE, p, 2, MPI\_COMM\_WORLD);

        }

        // Вычисления с участием нераспределённого куска вектора b

        for (int j = vecPartSize \* (size - 1); j < N; ++j)

        {

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

            {

                s += a[i] \* b[j];

            }

        }

        // Приём результатов вычислений процессов

        MPI\_Status st;

        double partOfSum;

        for (int p = 1; p < size; ++p)

        {

            MPI\_Probe(MPI\_ANY\_SOURCE, 3, MPI\_COMM\_WORLD, &st);

            MPI\_Recv(&partOfSum, 1, MPI\_DOUBLE, st.MPI\_SOURCE, 3, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            // MPI\_Recv(&partOfSum, 1, MPI\_DOUBLE, p, 3, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

            s += partOfSum;

        }

        double endTime = MPI\_Wtime();

        std::cout << "Total time: " << endTime - startTime << "\n";

        std::cout << "Check result: " << s << "\n";

        free(a);

        free(b);

    }

    else

    {

        // Инициализация данных

        double \*a = (double \*)malloc(sizeof(double) \* N);

        double \*b = (double \*)malloc(sizeof(double) \* vecPartSize);

        // Приём данных

        MPI\_Recv(a, N, MPI\_DOUBLE, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

        MPI\_Recv(b, vecPartSize, MPI\_DOUBLE, 0, 2, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

        // Вычисления

        double partOfSum = 0;

        for (int j = 0; j < vecPartSize; ++j)

        {

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

            {

                partOfSum += a[i] \* b[j];

            }

        }

        // Отправка результата

        MPI\_Send(&partOfSum, 1, MPI\_DOUBLE, 0, 3, MPI\_COMM\_WORLD);

        free(a);

        free(b);

    }

    MPI\_Finalize();

    return 0;

}

// prog3.cpp (коллективная коммуникация)

#include <iostream>

#include <mpi.h>

#include "settings.h"

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

{

    int size, rank;

    MPI\_Init(&argc, &argv);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    double startTime, endTime;

    int vecPartSize = N / size;

    // Выделение памяти

    double \*a = (double \*)malloc(sizeof(double) \* N);

    double \*b = rank == 0 ? (double \*)malloc(sizeof(double) \* N) : nullptr; // Если процесс не корневой, то этот массив не нужен

    double \*rec = (double \*)malloc(sizeof(double) \* vecPartSize);           // Для получения частей вектора b

    // Инициализация для нулевого процесса

    if (rank == 0)

    {

        for (int i = 0; i < N; ++i)

        {

            a[i] = 1.0;

            b[i] = 1.0;

        }

        startTime = MPI\_Wtime();

    }

    // Передача вектора a всем процессам

    MPI\_Bcast(a, N, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    // Распределение вектора b по всем процессам

    MPI\_Scatter(b, vecPartSize, MPI\_DOUBLE, rec, vecPartSize, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    // Вычисления

    double partOfSum = 0;

    for (int j = 0; j < vecPartSize; ++j)

    {

        for (int i = 0; i < N; ++i)

        {

            partOfSum += a[i] \* rec[j];

        }

    }

    if (rank == 0)

    {

        for (int j = vecPartSize \* size; j < N; ++j)

        {

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

            {

                partOfSum += a[i] \* b[j];

            }

        }

    }

    // Суммирование результатов

    double s = 0;

    MPI\_Reduce(&partOfSum, &s, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

    if (rank == 0)

    {

        endTime = MPI\_Wtime();

        std::cout << "Total time: " << endTime - startTime << "\n";

        std::cout << "Check result: " << s << "\n";

    }

    free(a);

    free(b);

    free(rec);

    MPI\_Finalize();

    return 0;

}