Оглавление

[1 Постановка задачи 3](#_Toc97373798)

[2 Алгоритм решения задачи 4](#_Toc97373799)

[3 Вычислительная и коммуникационная сложности алгоритма 5](#_Toc97373800)

[3.1 Вычислительная сложность алгоритма 5](#_Toc97373804)

[3.2 Коммуникационная сложность 5](#_Toc97373805)

[4 Внешние спецификации 6](#_Toc97373806)

[4.1 Наименование и назначение 6](#_Toc97373807)

[4.2 Команда сборки программы 6](#_Toc97373808)

[4.3 Формат вызова программы 6](#_Toc97373809)

[4.4 Входные данные 7](#_Toc97373810)

[4.5 Выходные данные 7](#_Toc97373811)

[4.6 Внешние эффекты 8](#_Toc97373812)

[5 Результаты вычислительного эксперимента 10](#_Toc97373813)

[Вывод 15](#_Toc97373814)

[Приложение А Исходный код программы Multimatrix 16](#_Toc97373815)

# 1 Постановка задачи

Разработать программное средство перемножения матриц с помощью ленточного алгоритма с использованием технологий MPI и OpenMP.

Оценить вычислительную и коммуникационную сложности реализованного алгоритма.

Провести вычислительный эксперимент. Исследовать зависимость времени выполнения программы от:

* числа процессов
* числа потоков;
* размера матриц.

# 2 Алгоритм решения задачи

Для решения задачи даны две исходные квадратные матрицы А и В размера s. Далее происходит разбиение данных матриц на равные ленты. Лента представляет собой часть массива с шириной и длиной s, количество элементов в ленте n, где n вычисляется по формуле:

, где:

* n – количество элементов в ленте;
* s – размер входных матриц;
* p – количество MPI-процессов.

Параметр s принимается из аргументов командной строки. После разбиения входных матриц на ленты начинается перемножение лент в каждом из процессов. Перемножение входных матриц происходит в цикле, и на каждой итерации каждый из процессов содержит по ленте с одинаковым количеством элементов. Количество итераций равно числу MPI-процессов.

Обозначим за A и В входные матрицы, С – выходная матрица, Ak, Bk, Сk – ленты соответствующих матриц, aij, bkm и сht – элементы лент Аk, Вk и Сk соответственно, где i,j,k,m,h,t – индексы в соответствующих лентах, . В результате перемножение лент из матриц А и В на одной итерации получается часть ленты матрицы C размера . После выполнения всего цикла в каждом MPI-процессе получится итоговая лента матрицы С.

Обозначим за der[l] часть ленты из матрицы С размерностью , который вычисляется на каждой итерации, где . Перемножение лент выполняется по следующей формуле:

der[l] =

В каждом процессе параллельно в потоках выполняется данный цикл.

# 3 Вычислительная и коммуникационная сложности алгоритма



## Вычислительная сложность алгоритма

Обозначим: s – размерность входных матриц, p – количество процессов. Перемножение выполняется в цикле по количеству процессов, на каждой итерации вычисляется часть ленты итоговой матрицы размера , и для вычисления каждого элемента требуется s операций, то сложность будет

## Коммуникационная сложность

При вычислении произведения матриц каждый процесс передает свою ленту матрицы В и принимает ленту матрицы В от соседнего процесса на каждой итерации. В каждой ленте элементов, количество итераций обмена p-1, так как после последней итерации осуществляется сбор результирующих лент с каждого процесса. Сложность передачи на одной итерации . Так как после вычислений осуществляется сбор данных и сложность в таком случае будет .

# 4 Внешние спецификации

## 4.1 Наименование и назначение

Программное средство «multimatrix» предназначено для решения задачи перемножения матриц с помощью ленточного алгоритма с использованием технологий параллельного программирования MPI и OpenMP.

## 4.2 Команда сборки программы

Программное средство реализовано с помощью языка программирования C++ с использованием возможностей программного интерфейса передачи сообщений MPI и стандарта распараллеливания программ с общей памятью OpenMP. Сборка программного средства осуществляется с помощью следующей команды:

mpic++ multimatrix.cpp -o multimatrix -fopenmp -std=C++11

В программе применяются последовательные контейнеры, а именно вектор, поэтому необходимо использовать стандарт C++11.

## 4.3 Формат вызова программы

Для запуска программного средства в режиме тестирования необходимо выполнить команду:

mpirun -ppn 1 -np <n> multimatrix <size> <nt> -f <infile>

где:

ppn – количество процессов на вычислительный модуль

size – размер квадратной матрицы

infile – имя входного файла (имя файла, включая либо абсолютный путь к нему, либо относительный от текущей рабочей директории)

n – количество запускаемых MPI-процессов

nt – число потоков

Для запуска в режиме вычислительного эксперимента:

mpirun -ppn 1 -np <n> multimatrix <size> <nt>

где:

size – размер квадратной матрицы

nt – число потоков

ppn – количество процессов на вычислительный модуль

## 4.4 Входные данные

Для тестового режима необходим текстовый файл, содержащий матрицы A и B. Размер матриц передается в аргументах командной строки при запуске программы. Файл должен быть соответствовать следующим правилам:

* ­­­­Матрицы должны быть квадратные (в случае невыполнения будет выдано сообщение об ошибке);
* Матрицы отделяются пустой строкой;
* Каждая строка матрицы представляет собой строку во входном файле;
* Элементы матрицы в строке разделяется пробелом;
* Элементы матрицы – целые и вещественные числа;
* Разделителем дробной и целой части вещественного числа является точка. Пример содержимого входного файла, содержащего две матрицы размера 3x3:

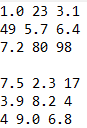


Рисунок 1-пример входного файла, содержащего две матрицы

При выполнении в режиме вычислительного эксперимента входными данными является размер матрицы. Данные о размерности матриц принимаются из аргументов вызова программы.

## 4.5 Выходные данные

В результате выполнения программы будет вычислено произведение матриц A и B. В тестовом режиме в стандартный поток вывода будет выведена результирующая матрица C. Каждая строка матрицы представляет собой строку в стандартном потоке вывода, элементы матрицы в строке разделяются пробелом. Элементы матрицы – целые и вещественные числа, разделителем дробной и целой части вещественного числа является точка.

В режиме вычислительного эксперимента после вычисления результирующей матрицы в стандартный поток вывода будет выведено сообщение вида «The program time: t», где t – время выполнения программы в секундах с точностью до десятых долей.

## 4.6 Внешние эффекты

Информация о возникающих ошибках будет направлена в стандартный поток ошибок. Другие внешние эффекты не предусмотрены. Примеры предусмотренных внешних эффектов приведены в таблице:

|  |  |
| --- | --- |
| Сообщение в стандартном потоке ошибок | Описание |
| Wrong argument | Отсутствуют необходимые аргументы, неверный порядок аргументов, неизвестные аргументы.  Неверно указаны аргументы при запуске программы. |
| «Error. File <filename> is incorrect: <message>» | Ошибка при открытии файла <filename>  <message> - описание ошибки |
| Matrix size error | Размер матрицы во входном файле неверен или не совпадает с размером, указанным при запуске программы в качестве аргумента |
| Еrror element in file at [i][j] in matrix A | Некорректный элемент [i][j] матрицы А |
| Еrror element in file at [i][j] in matrix B | Некорректный элемент [i][j] матрицы B |
| Error size | Указан неверный размер матриц |
| Error number threads | Неверно указано количество потоков |
| Matrix square error | Одна или обе матрицы не являются квадратными |

# 5 Результаты вычислительного эксперимента

При проведении эксперимента исследовалась зависимость выполнения программы от:

* числа потоков;
* размера матриц;
* числа MPI-процессов.

Эксперимент был произведен на вычислительном кластере, взаимодействие с которым осуществляется через сервер доступа. В кластер объединены 8 серверов с установленной операционной системой Linux, соединенных между собой ЛВС (локальной вычислительной сетью) Gigabit Ethernet с использованием витой пары. Сервер доступа также подключен в ЛВС и является отдельным компьютером. На каждом сервере установлено два процессора AMD Opteron Processor 6380. Характеристики каждого сервера: базовая частота процессора 2,5 Ггц, 16 вычислительных ядер, объем памяти кэша первого уровня 2048 Кб, объем памяти кэша второго уровня 16 Мб, объем ОЗУ 8 Гб. В рамках кластера была развернута виртуальная система, которая позволяет производить вычисления пользователю на 24 виртуальных вычислительных узлах. На каждом сервере развернуто 3 виртуальных вычислительных узла, на которых доступно по 8 ядер и по 8 ГБ оперативной памяти.

Вычислительный эксперимент проводился в условиях запуска задания на СУППЗ (Система управления прохождения параллельных заданий), которая располагается на сервере доступа. Задание представляет собой исполняемый файл, входные данные и необходимые ресурсы для выполнения программы. При выполнении команды mpirun СУППЗ выделяет узлы, которые свободны, или ставит в очередь задачу для последующего выполнения. С командой mpirun также передаются количество MPI-процессов для задания с помощью параметра -np и максимальное время с помощью параметра -maxtime, которое может выполняться задание, после чего СУППЗ снимает ее с выполнения по истечении времени. На каждом узле производится запуск одного MPI-процесса. На выбранных СУППЗ узлах производится выполнение задания, тем самым запускается программа, которая порождает один процесс. Внутри процесса было запущено несколько потоков с использованием технологии OpenMP. Количество потоков задается в аргументах командной строки.

В эксперименте исследовалось зависимость времени выполнения программы от количества данных, количества MPI-процессов и количества потоков. Каждый поток выполнялся на 1 ядре. На вход подавалось число потоков и размер массива. Время на выделение памяти и генерацию матрицы не учитывалось. По результатам проведения эксперимента были сформированы таблицы (Таблица 1,2,3) и построены графики (Рисунок 1,2,3).

Таблица 1 – Зависимость времени выполнения программы (в секундах)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество MPI-процессов | Количество потоков | Количество данных | Время выполнения программы |
| 5 | 1 | 1000 | 2,8968 |
| 5 | 2 | 1000 | 2,40781 |
| 5 | 4 | 1000 | 4,67327 |
| 5 | 8 | 1000 | 5,30531 |
| 5 | 16 | 1000 | 2,22677 |

Рисунок 1 – График зависимости времени выполнения программы от количества потоков

Таблица 2 – Зависимость времени выполнения программы (в секундах)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество MPI-процессов | Количество потоков | Количество данных | Время выполнения программы |
| 1 | 4 | 1000 | 9,27702 |
| 2 | 4 | 1000 | 4,36035 |
| 4 | 4 | 1000 | 3,97463 |
| 5 | 4 | 1000 | 8,47948 |
| 8 | 4 | 1000 | 3,60003 |

Рисунок 2 – График зависимости времени выполнения программы от количества процессов

Таблица 3 – Зависимость времени выполнения программы (в секундах)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество MPI-процессов | Количество потоков | Количество данных | Время выполнения программы |
| 5 | 16 | 800 | 1,44743 |
| 5 | 16 | 1000 | 2,00286 |
| 5 | 16 | 2000 | 23,8062 |
| 5 | 16 | 3000 | 64,2675 |
| 5 | 16 | 4000 | 175,49 |
| 5 | 16 | 5000 | 225,26 |
| 5 | 16 | 6000 | 337,865 |

Рисунок 2 – График зависимости времени выполнения программы от количества данных

Из результатов эксперимента видно, что при фиксированных количестве данных и количестве MPI-процессов минимальное время выполнения программы достигается при 16 потоках, наибольше время выполнения программы достигается при 8 потоках.

При фиксированных количестве потоков и количестве данных минимальное время выполнения достигается при 5 MPI-процессов, а максимальное при 1 MPI-процессе.

При фиксированных количестве процессов и потоков время выполнения программы увеличивается с увеличением количества данных.

# Выводы

В результате выполнения долгосрочного домашнего задания были изучены технологии параллельного программирования MPI и OpenMP. Была разработана программа на языке программирования С++ для вычисления произведения матриц. Была осуществлена оценка вычислительной и коммуникационной сложностей ленточного алгоритма перемножения квадратных матриц. Исследована зависимость времени выполнения программы от размера матриц, количества потоков и количества MPI-процессов.

# Приложение А Исходный код программы Multimatrix

Листинг А.1 – исходный код программы Multimatrix.cpp

/\*

Автор: Алифьев Дмитрий - 3 группа

Задание 27. Перемножение матриц с помощью ленточного алгоритма (MPI+OpenMP)

\*/

#include "mpi.h"

#include <fstream>

#include <iostream>

#include <omp.h>

#include <string.h>

#include <time.h>

#include <sstream>

using namespace std;

//Функция проверки строки на вещественное число

bool isNum(const string & s)

{

string::const\_iterator it = s.begin();

while (it != s.end() && (isdigit(\*it) || (\*it)=='.')) ++it;

return !s.empty() && it == s.end();

}

// Функция проверки строки на целое число

bool isInt(const string & s)

{

string::const\_iterator it = s.begin();

while (it != s.end() && (isdigit(\*it) )) ++it;

return !s.empty() && it == s.end();

}

//Функция чтения ленты из файла в зависимости от ранга процесса

bool read\_file(ifstream &in, double \* mass,double \*mass2, int size, int proc,int rank){

string line;

const char delim = ' ';

string item;

// Чтение матрицы А

if(in){

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

getline(in,line);

line.clear();

}

getline(in,line);

stringstream ss(line);

int i = 0;

while (getline(ss,item,delim))

{

if(!isNum(item)){

fprintf(stderr,"Error element in file at [%d] [%d] in matrix A\n",rank,i);

return false;

}

double value = stod(item);

mass[i] = value;

i++;

}

for(int i = rank; i < proc; i ++){

getline(in,line);

line.clear();

}

getline(in,line);

//Чтение матрицы В

double \*tmp = new double[proc];

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

getline(in,line);

stringstream ss1(line);

int j = 0;

while (getline(ss1,item,delim))

{

if(!isNum(item)){

fprintf(stderr,"Error element in file at [%d] [%d] in matrix B\n",i,j);

return false;

}

double value = stod(item);

tmp[j] = value;

j++;

}

mass2[i] = tmp[rank];

}

}

in.close();

return true;

}

//Функция генерация ленты

void randMat(double \*mass,int size,int rank){

srand(rank);

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

mass[i] = (double)rand()/RAND\_MAX;

}

}

//Функция перемножения матриц

bool MatrixMultiplicationMPI(string fileName, double \*&C,

int &Size,int ProcNum,int

ProcRank,int numThreads,bool

isRand,double &time) {

int dim = Size;

int i, j, k, p, ind;

double temp;

MPI\_Status Status;

//Определение ширины ленты

int ProcPartSize = dim/ProcNum;

//Количество элементов в ленте

int ProcPartElem = ProcPartSize\*dim;

//Выделение памяти под ленты в процессах

double\* bufA = new double[ProcPartElem];

double\* bufB = new double[ProcPartElem];

double\* bufC = new double[ProcPartElem];

if(!isRand){

//В тестовом режиме чтение из файла лент

ifstream fin(fileName);

if(!fin.is\_open()){

if(ProcRank == 0){

fprintf(stderr,"Error. File %s is incorrect: %s \n",fileName.c\_str(),strerror(errno));

delete []bufA;

delete []bufB;

delete []bufC;

return false;

}

}

//Чтение из файла if(!read\_file(fin,bufA,bufB,ProcPartElem,dim,ProcRank)){

delete []bufA;

delete []bufB;

delete []bufC;

return false;

}

}else{

//В экспериментальном режиме генерация лент

randMat(bufA,ProcPartElem,ProcRank);

randMat(bufB,ProcPartElem,ProcRank);

}

//Замер времени

double time\_start = MPI\_Wtime();

temp = 0.0;

// Перемножение загруженных в процессы лент

for (i=0; i < ProcPartSize; i++) {

for (j=0; j < ProcPartSize; j++) {

#pragma omp parallel for schedule(static,numThreads) reduction (+:temp)

for (k=0; k < dim; k++)

temp += bufA[i\*dim+k]\*bufB[j\*dim+k];

bufC[i\*dim+j+ProcPartSize\*ProcRank] = temp;

temp = 0.0;

}

}

int NextProc; int PrevProc;

for (p=1; p < ProcNum; p++) {

//Выбор следующего и предыдущего процесса

//для обмена данными

NextProc = ProcRank+1;

if (ProcRank == ProcNum-1)

NextProc = 0;

PrevProc = ProcRank-1;

if (ProcRank == 0)

PrevProc = ProcNum-1;

//Обмен данными

MPI\_Sendrecv\_replace(bufB, ProcPartElem, MPI\_DOUBLE, NextProc, 0, PrevProc, 0, MPI\_COMM\_WORLD, &Status);

temp = 0.0;

// Перемножение принятой ленты матрицы B и

// оставшейся ленты матрицы А

for (i=0; i < ProcPartSize; i++) {

for (j=0; j < ProcPartSize; j++) {

#pragma omp parallel for schedule(static,numThreads) reduction (+:temp)

for (k=0; k < dim; k++) {

temp += bufA[i\*dim+k]\*bufB[j\*dim+k];

}

if (ProcRank-p >= 0 )

ind = ProcRank-p;

else ind = (ProcNum-p+ProcRank);

bufC[i\*dim+j+ind\*ProcPartSize] = temp;

temp = 0.0;

}

}

}

//Сбор результирующих лент в процесс с рангом 0

MPI\_Gather(bufC, ProcPartElem, MPI\_DOUBLE, C, ProcPartElem, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

double time\_end = MPI\_Wtime();

time = time\_end - time\_start;

delete []bufA;

delete []bufB;

delete []bufC;

return true;

}

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

MPI\_Init(&argc,&argv);

int sizeMat;

int i, j, k, p, ind;

double temp;

int ProcNum;

int ProcRank;

//Считывание в переменные количество процессов и ранг

//текущего процесса

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

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

if(argc == 5){

//Тестовый режим

string inputFile(argv[4]);

//Запись в переменную количества потоков

if(!isInt(string(argv[2]))){

if(ProcRank == 0)

fprintf(stderr,"Error number threads\n");

MPI\_Finalize();

return 1;

}

//Запись в переменную размерность массивов

int thNum = atoi(argv[2]);

if(!isInt(string(argv[1]))){

if(ProcRank == 0)

fprintf(stderr,"Error size\n");

MPI\_Finalize();

return 1;

}

sizeMat = atoi(argv[1]);

//Выделение памяти под результирующую матрицу

double \*C = new double[sizeMat\*sizeMat];

MPI\_Status Status;

double time\_exec = 0.0;

//Перемножение матриц

bool res = MatrixMultiplicationMPI(inputFile,C,sizeMat,ProcNum,ProcRank,thNum,false,time\_exec);

if(!res){

return 1;

}

//Вывод результирующей матрицы в выходной файл

if(ProcRank == 0){

for(int i = 0;i<sizeMat\*sizeMat;i++){

cout<<C[i]<<' ';

if((i+1)%sizeMat == 0)

cout<<endl;

}

}

MPI\_Finalize();

return 0;

}else

//Экспериментальный режим

if(argc == 3){

//Запись в переменную количества потоков

if(!isInt(string(argv[2]))){

if(ProcRank == 0)

fprintf(stderr,"Error number threads\n");

MPI\_Finalize();

return 1;

}

int thNum = atoi(argv[2]);

//Запись в переменную размерность массивов

if(!isInt(string(argv[1]))){

if(ProcRank == 0)

fprintf(stderr,"Error size\n");

MPI\_Finalize();

return 1;

}

sizeMat = atoi(argv[1]);

//Запись в переменную размерность массивов

double \*C = new double[sizeMat\*sizeMat];

MPI\_Status Status;

//Переменная для времени выполнения перемножения

//матриц

double time\_exec = 0.0;

//Перемножение матриц MatrixMultiplicationMPI("",C,sizeMat,ProcNum,ProcRank,thNum,true,time\_exec);

//Вывод в файл времени выполнения перемножения

//матриц

if(ProcRank == 0)

cout<<time\_exec<<endl;

MPI\_Finalize();

return 0;

}else{

if(ProcRank == 0)

fprintf(stderr,"Wrong arguments\n");

MPI\_Finalize();

return 1;

}

return 0;

}