Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Нижегородский государственный университет им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

**Отчет**

по лабораторной работе №3

**«Выделение ребер на изображении с использованием оператора Собеля.»**

**Выполнил:**

студент группы 1608

Черненко В.Н.

**Проверил:**

Доцент кафедры МОСТ

Сысоев А.В.

Нижний Новгород

2018

**Содержание**

[Постановка задачи 2](#_Toc531367741)

[Метод решения 3](#_Toc531367742)

[Схема распараллеливания 4](#_Toc531367743)

[Описание программной реализации 5](#_Toc531367744)

[Подтверждение корректности 6](#_Toc531367745)

[Результаты экспериментов 7](#_Toc531367746)

[Заключение 8](#_Toc531367747)

[Приложение 9](#_Toc531367748)

# Постановка задачи

В данной лабораторной работе ставится задача реализовать программу, использующую средства параллельного программирования MPI, которая выделяет ребра на изображении, используя фильтр Собеля.

Программа должна cодержать следующие возможности:

* Ввод размерности изображения через параметры командной строки.
* Генерация изображения исходя из переданных параметров
* Подсчет затраченного времени исполнения последовательной и параллельной части.
* Подтверждение корректности выполнения

# Метод решения

Для выделения ребер на изображении с использованием фильтра Собеля используется оператор Собеля. Оператор использует две матрицы 3\*3:

И

Для каждого пикселя на изображении с координатами вычисляются два значения , используя правила применения матричных фильтров, а затем итоговое значение в новом изображении

# Схема распараллеливания

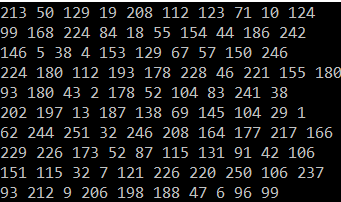
Так как исходное изображение задается в виде одномерного массива, целесообразно для организации параллельной работы отправлять изображение равными блоками, чей размер кратен ширине изображения (Блоки строк). Однако, так как оператор Собеля подразумевает работу с окрестностью пикселя изображения, каждому блоку добавляется по строке «сверху» и «снизу», за исключением первого и последнего процесса, которые получают блок и одну строку «снизу» или «сверху» соответственно. Так первый и последний процессы получают N + 1 строку, а остальные N + 2 строки, где N – результат целочисленного деления количества строк на количество процессов. Если количество строк не кратно количеству процессов, то оставшиеся M строк равномерно распределяются по первым M процессам. После чего выполняется последовательная реализация выделения ребер на изображении, исключая «дополнительные» строки, описанная в методах решения. Исходное изображение собирается со всех процессах на первом.

# Описание программной реализации

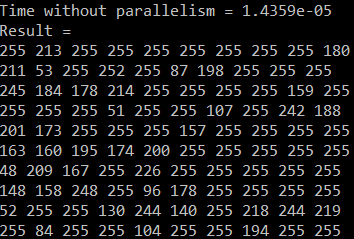
Запуск программы осуществляется через командную строку с помощью

следующей команды: mpiexec –n X “/path/program.exe W H, где:

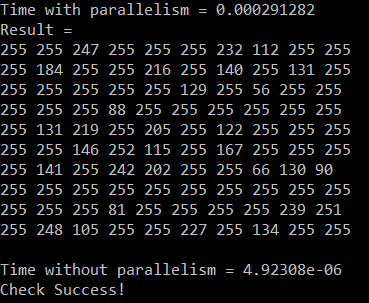
* X – количество процессов
* “/path/program.exe – полный путь к исполняемому файлу в кавычках;
* W H – размер генерируемого изображения (Ширина и высота соответсвенно)

После запуска программа сгенерирует изображение (Каждый пиксель может принимать значение от 0 до 255 включительно) и выведет его на экран:

Затем в случае если процесс всего 1, либо количество строчек недостаточно для корректной работы распараллеливания, программа выполнит линейную реализацию и выведет результат с затраченным временем.



В случае если распараллеливание возможно, первый процесс с помощью функции MPI\_Scatterv сообщит всем остальным процессам, включая себя, свой блок, каждый процесс выполнит последовательную реализацию алгоритма на своем блоке и первый процесс с помощью функции MPI\_Gatherv соберет результат выделения ребер на первом процессе, выведет результат и выполнит проверку корректности.



Код программы можно просмотреть в разделе «[Приложение](#_Приложение)».

# Подтверждение корректности

Для подтверждения корректности в программе после выполнения параллельной части выполняется линейная реализация на первом процессе и выполняется сравнение полученных изображений. В случае, если изображения равны, программа выведет Check Success!, иначе Incorrect Answer!.

# Результаты экспериментов

Для тестирования распараллеливания используются популярные размеры изображения и запуска программы на разных количествах процессов:

Результат экспериментов на персональном компьютере.

Характеристики:

• Процессор i5-3450 3.1GHz

• оперативная память: 8 GB 1866GHz

• OS: windows 10 Домашняя.

|  |  |  |  |
| --- | --- | --- | --- |
| Результаты запуска программы на 4 процессах. | | | |
| Размеры изображений | Время с параллелизмом | Время без параллелизма | Коэффициент распараллеливания |
| 16 \* 9 | 0,0008 | 0,000007 | 0,00875 |
| 160 \* 90 | 0,0086 | 0,0017 | 0,197674 |
| 640 \* 480 | 0,006 | 0,0166 | 2,766667 |
| 800 \* 600 | 0,013 | 0,035 | 2,692308 |
| 1280 \* 800 | 0,032 | 0,066 | 2,0625 |
| 1920 \* 1080 | 0,055 | 0,154 | 2,8 |
| 3840 \* 2160 | 0,215742 | 0,812953 | 3,768172 |
| 7680\*4320 | 1,73912 | 6,34953 | 3,651002 |

|  |  |  |  |
| --- | --- | --- | --- |
| Результаты запуска программы на 8 процессах. | | | |
| Размеры изображений | Время с параллелизмом | Время без параллелизма | Коэффициент распараллеливания |
| 16 \* 9 | 0,001376 | 0,00000615 | 0,004468 |
| 160 \* 90 | 0,000768 | 0,0005678 | 0,739798 |
| 640 \* 480 | 0,016585 | 0,0174753 | 1,053706 |
| 800 \* 600 | 0,02638 | 0,0356788 | 1,352494 |
| 1280 \* 800 | 0,031621 | 0,09080353 | 2,87163 |
| 1920 \* 1080 | 0,087343 | 0,172414 | 1,973992 |
| 3840 \* 2160 | 0,1758 | 0,899087 | 5,114261 |
| 7680\*4320 | 0,946259 | 6,48916 | 6,8577 |

|  |  |  |  |
| --- | --- | --- | --- |
| Результаты запуска программы на 12 процессах. | | | |
| Размеры изображений | Время с параллелизмом | Время без параллелизма | Коэффициент распараллеливания |
| 16 \* 9 | - | 0,00000615 | 0 |
| 160 \* 90 | 0,000996 | 0,00054 | 0,542114 |
| 640 \* 480 | 0,00991 | 0,0148 | 1,493441 |
| 800 \* 600 | 0,022296 | 0,025493 | 1,143395 |
| 1280 \* 800 | 0,02747 | 0,067891 | 2,471471 |
| 1920 \* 1080 | 0,0602 | 0,15649 | 2,599502 |
| 3840 \* 2160 | 0,23665 | 0,923691 | 3,903195 |
| 7680\*4320 | 0,89042 | 7,04929 | 7,916815 |

|  |  |  |  |
| --- | --- | --- | --- |
| Результаты запуска программы на 32 процессах. | | | |
| Размеры изображений | Время с параллелизмом | Время без параллелизма | Коэффициент распараллеливания |
| 16 \* 9 | - | 0,00000615 | 0 |
| 160 \* 90 | 0,004135 | 0,000663 | 0,160331 |
| 640 \* 480 | 0,012614 | 0,01515 | 1,201027 |
| 800 \* 600 | 0,015444 | 0,036821 | 2,38414 |
| 1280 \* 800 | 0,0307 | 0,0655 | 2,13355 |
| 1920 \* 1080 | 0,0589477 | 0,146674 | 2,488206 |
| 3840 \* 2160 | 0,195469 | 0,970174 | 4,963314 |
| 7680\*4320 | 0,872797 | 6,53858 | 7,491524 |

# Заключение

В ходе выполнения лабораторной работы, мы ознакомились с алгоритмами выделения ребер на изображении с использованием оператора Собеля. По

результатам экспериментов видно, что распараллеливание принесло

значимое ускорение работы программы.

В результате все поставленные задачи были выполнены, цели достигнуты.

Мы приобрели навыки в применении параллельного программирования с

помощью MPI. А также исследовали алгоритм применения матричных фильтров на изображении для выделения ребер с помощью оператора Собеля, а также применения функций MPI\_Scatterv и MPI\_Gatherv. Теперь перед нами стоит задача в применении приобретенных знаний в своей будущей профессиональной деятельности.

# Приложение

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <mpi.h>

using namespace std;

int\* getRandImage(int width, int height, int max = 256) {

int\* result = new int[width \* height];

for (int i = 0; i < width \* height; i++) result[i] = rand() % max;

return result;

}

void printImage(int\* image, int wigth, int height) {

for (int j = 0; j < height; j++) {

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

cout << image[i + j\*wigth] << " ";

cout << endl;

}

}

int getPixel(int\* image, int width, int height, int x, int y) {

if (x < 0) x = 0;

if (x >= width) x = width - 1;

if (y < 0) y = 0;

if (y >= height) y = height - 1;

return image[x + y\*width];

}

int getSobelPixel(int\* image, int width, int height, int x, int y, int\* sobelMatrix1, int\* sobelMatrix2) {

int resultX = 0;

int resultY = 0;

for(int i = -1; i <= 1; i++)

for (int j = -1; j <= 1; j++) {

resultX += getPixel(image, width, height, x + i, y + j) \* sobelMatrix1[i + 1 + (j + 1) \* 3];

resultY += getPixel(image, width, height, x + i, y + j) \* sobelMatrix2[i + 1 + (j + 1) \* 3];

}

int result = sqrt(resultX\*resultX + resultY\*resultY);

if (result < 0) return 0;

if (result > 255) return 255;

return result;

};

int\* getSobelImage(int\* image, int width, int height, int\* sobelMatrix1, int\* sobelMatrix2) {

int\* resultImage = new int[width \* height] {0};

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

for (int j = 0; j < height; j++) {

resultImage[i + j\*width] = getSobelPixel(image, width, height, i, j, sobelMatrix1, sobelMatrix2);

}

return resultImage;

}

int\* getPartlySobelImage(int\* image, int width, int height, int startX, int startY, int toX, int toY, int\* sobelMatrix1, int\* sobelMatrix2) {

int\* resultImage = new int[width \* height] {0};

for (int i = startX; i < toX; i++)

for (int j = startY; j < toY; j++) {

resultImage[i + j\*width] = getSobelPixel(image, width, height, i, j, sobelMatrix1, sobelMatrix2);

}

return resultImage;

}

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

srand(time(0));

int ProcRank, ProcSize;

int\* sobelMatrix1 = new int[9]{ -1, -2, -1, 0, 0, 0, 1, 2, 1 };

int\* sobelMatrix2 = new int[9]{ -1, 0, 1, -2, 0, 2, -1, 0, 1 };

MPI\_Init(&argc, &argv);

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

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

if (ProcRank == 0) {

int width = atoi(argv[1]);

int height = atoi(argv[2]);

int RowsForProc = height / ProcSize;

int RestOfRows = height % ProcSize;

MPI\_Bcast(&width, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&height, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&RowsForProc, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&RestOfRows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* image = getRandImage(width, height);

// printImage(image, width, height);

// cout << endl;

if (ProcSize == 1 || RowsForProc == 0) {

double startTime = MPI\_Wtime();

int\* ResultImage = getSobelImage(image, width, height, sobelMatrix1, sobelMatrix2);

double finishTime = MPI\_Wtime();

cout << "Time without parallelism = " << finishTime - startTime << endl;

cout << "Result = " << endl;

// printImage(ResultImage, width, height);

// cout << endl;

delete image;

delete ResultImage;

MPI\_Finalize();

return 0;

}

int\* ResultImage = new int[width \* height];

double startTime = MPI\_Wtime();

int\* sendCounts = new int[ProcSize];

int\* displs = new int[ProcSize];

displs[0] = 0;

if (RestOfRows > 0) {

sendCounts[0] = (RowsForProc + 2)\*width;

--RestOfRows;

} else sendCounts[0] = (RowsForProc + 1)\*width;

for (int i = 1; i < ProcSize - 1; i++) {

displs[i] = displs[i - 1] + sendCounts[i - 1] - 2 \* width;

if (RestOfRows > 0) {

sendCounts[i] = (RowsForProc + 3) \* width;

--RestOfRows;

}

else sendCounts[i] = (RowsForProc + 2) \* width;

}

displs[ProcSize - 1] = displs[ProcSize - 2] + sendCounts[ProcSize - 2] - 2 \* width;

sendCounts[ProcSize - 1] = (RowsForProc + 1) \* width;

int\* WorkImage = new int[sendCounts[0]];

MPI\_Scatterv(image, sendCounts, displs, MPI\_INT, WorkImage, sendCounts[0], MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* ResultWorkImage = getPartlySobelImage(WorkImage, width, sendCounts[0] / width, 0, 0, width, sendCounts[0] / width - 1, sobelMatrix1, sobelMatrix2);

RestOfRows = height % ProcSize;

int\* recvCounts = new int[ProcSize];

displs[0] = 0;

if (RestOfRows > 0) {

recvCounts[0] = (RowsForProc + 1) \* width;

--RestOfRows;

}

else recvCounts[0] = RowsForProc \* width;

for (int i = 1; i < ProcSize; i++) {

displs[i] = displs[i-1] + recvCounts[i - 1];

if (RestOfRows > 0) {

recvCounts[i] = (RowsForProc + 1) \* width;

--RestOfRows;

}

else recvCounts[i] = RowsForProc \* width;

}

MPI\_Gatherv(ResultWorkImage, recvCounts[0], MPI\_INT, ResultImage, recvCounts, displs, MPI\_INT, 0, MPI\_COMM\_WORLD);

double finishTime = MPI\_Wtime();

cout << "Time with parallelism = " << finishTime - startTime << endl;

// cout << "Result = " << endl;

// printImage(ResultImage, width, height);

cout << endl;

startTime = MPI\_Wtime();

int\* CheckResultImage = getSobelImage(image, width, height, sobelMatrix1, sobelMatrix2);

finishTime = MPI\_Wtime();

bool check = true;

for (int i = 0; i < width \* height; i++) if (ResultImage[i] != CheckResultImage[i]) {

check = false;

break;

}

cout << "Time without parallelism = " << finishTime - startTime << endl;

if (check) cout << "Check Success!" << endl;

else cout << "Incorrect Answer!" << endl;

delete image;

delete ResultImage;

delete CheckResultImage;

delete sendCounts;

delete recvCounts;

delete displs;

delete ResultWorkImage;

delete WorkImage;

}

else {

int width, height, RowsForProc, RestOfRows;

MPI\_Bcast(&width, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&height, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&RowsForProc, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(&RestOfRows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (RowsForProc == 0) {

MPI\_Finalize();

return 0;

}

if (ProcRank != ProcSize - 1) {

if (ProcRank < RestOfRows) {

int\* WorkImage = new int[(RowsForProc + 3)\* width];

MPI\_Scatterv(nullptr, nullptr, nullptr, MPI\_INT, WorkImage, (RowsForProc + 3)\* width, MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* ResultWorkImage = getPartlySobelImage(WorkImage, width, RowsForProc + 3, 0, 1, width, RowsForProc + 2, sobelMatrix1, sobelMatrix2);

MPI\_Gatherv(ResultWorkImage + width, (RowsForProc + 1) \* width, MPI\_INT, nullptr, nullptr, nullptr, MPI\_INT, 0, MPI\_COMM\_WORLD);

delete WorkImage;

}

else {

int\* WorkImage = new int[(RowsForProc + 2)\* width];

MPI\_Scatterv(nullptr, nullptr, nullptr, MPI\_INT, WorkImage, (RowsForProc + 2)\* width, MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* ResultWorkImage = getPartlySobelImage(WorkImage, width, RowsForProc + 2, 0, 1, width, RowsForProc + 1, sobelMatrix1, sobelMatrix2);

MPI\_Gatherv(ResultWorkImage + width, RowsForProc \* width, MPI\_INT, nullptr, nullptr, nullptr, MPI\_INT, 0, MPI\_COMM\_WORLD);

delete WorkImage;

}

}

else {

int\* WorkImage = new int[(RowsForProc + 1)\* width];

MPI\_Scatterv(nullptr, nullptr, nullptr, MPI\_INT, WorkImage, (RowsForProc + 1)\* width, MPI\_INT, 0, MPI\_COMM\_WORLD);

int\* ResultWorkImage = getPartlySobelImage(WorkImage, width, RowsForProc + 1, 0, 1, width, RowsForProc + 1, sobelMatrix1, sobelMatrix2);

MPI\_Gatherv(ResultWorkImage + width, RowsForProc \* width, MPI\_INT, nullptr, nullptr, nullptr, MPI\_INT, 0, MPI\_COMM\_WORLD);

delete WorkImage;

}

}

MPI\_Finalize();

return 0;

}