Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ ПО ДИСЦИПЛИНЕ «Компьютерные системы и сети»

Тема: «Реализация параллельной обработки для кластерных / мультипроцессорных систем на базе протокола MPI»

КП.ПО-9.1-40 01 01

Листов: 17

Выполнил:
Студент 2-го курса,
ФЭИС,
Группы ПО-9
Харитонович З. С.
Нормоконтроль:
Савицкий Ю. В.
Проверил:
Савицкий Ю. В.

Брест 2023

ВВЕДЕНИЕ. АНАЛИЗ ЗАДАЧ ПРОЕКТИРОВАНИЯ

Целью данного курсового проектирования является приобретение навыков по разработке и реализации параллельных алгоритмов для многокомпонентных систем обработки данных, разработка параллельных приложений с использованием интерфейса MPI, сравнительный анализ временных характеристик выполнения обработки данных.

В процессе выполнения проектирования необходимо создать программу, реализующую параллельную обработку изображения на базе протокола MPI. Обработка изображения должна происходить в два этапа:

Подавление зернистого шума графических изображений с использованием медианного фильтра.

Выделение границ элементов чёрно-белого графического изображения на основе дифференциального оператора.

Для достижения поставленной задачи необходимо сначала разработать последовательный алгоритм обработки изображения, затем произвести его параллелизацию с помощью протокола MPI.

MPI (Message Passing Interface, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

1. РАЗРАБОТКА И ОПИСАНИЕ ПОСЛЕДОВАТЕЛЬНОГО АЛГОРИТМА ПРОГРАММНОЙ СИСТЕМЫ

Согласно варианту задания, необходимо реализовать обработку чёрно-белого изображения формата *bmp* размером 3000 на 3000 пикселей. Составим последовательный алгоритм приложения. Обработку изображения, для удобства последующей разработки, вынесем в отдельную функцию.

Программа получит на вход изображение (считает его), обработает его и запишет результат обработки в новый файл.

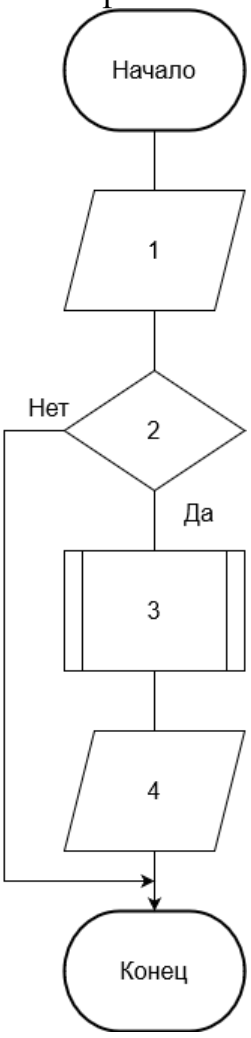

Рисунок 1 – последовательный алгоритм.

1) Чтение входного изображения.
2) Проверка правильности чтения.
3) Алгоритм обработки изображения (рис. 2).
4) Запись обработанного изображения.

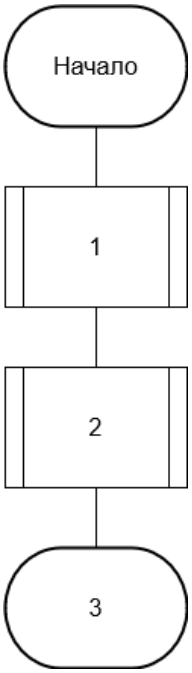

Рисунок 2 – алгоритм обработки изображения.

1) Медианный фильтр.
2) Дифференциальный фильтр.
3) Возврат обработанного изображения.

2. РАЗРАБОТКА ПРОГРАММНОЙ СИСТЕМЫ, РЕАЛИЗУЮЩЕЙ ПОСЛЕДОВАТЕЛЬНЫЙ АЛГОРИТМ ОБРАБОТКИ.

Для разработки приложения будет использоваться язык программирования *C++*. Данный язык программирования имеет огромное количество функционала, уже реализованного в виде отдельных библиотек.

Для чтения, записи, хранения и обработки изображения воспользуемся библиотекой *OpenCV*.

Само изображение будет храниться с помощью класса *cv::Mat*, который представляет из себя матрицу, которая хранит пиксели изображения и имеет набор методов для работы с ним.

Приступим к реализации последовательного алгоритма (рис. 1)

В первую очередь, считываем изображение.

cv::Mat cv::imread(const cv::String &filename) – функция чтения изображения из файла.

Обработка изображения, для последующего комфортного использования, вынесена в отдельную функцию (рис. 2) *cv::Mat imageProcessing(cv::Mat image)*. Данная функция, в свою очередь, производит обработку изображения в два этапа согласно заданию:

1) Подавление зернистого шума графического изображения с использованием медианного фильтра. Библиотека *OpenCV* предоставляет возможность использования уже реализованного алгоритма медианного фильтра – *void cv::medianBlur(cv::InputArray src, cv::OutputArray dst, int ksize)*.

2) Выделение границ элементов чёрно-белого графического изображения на основе дифференциального оператора. Для реализации дифференциального фильтра средствами библиотеки *OpenCV* потребуется произвести обработку с помощью фильтра Собеля по горизонтали и по вертикали посльедовательно с помощью функции *void cv::Sobel(cv::InputArray src, cv::OutputArray dst, int ddepth, int dx, int dy).* Затем, полученные наборы данных необходимо объединить в конечное изображение с помощью функции *void cv::magnitude(cv::InputArray x, cv::InputArray y, cv::OutputArray magnitude).*

Вышеописанная функция принимает входное изображение и возвращает уже полностью обработанное.

Затем результат обработки записывается в файл.

bool cv::imwrite(const cv::String &filename, cv::InputArray img) – функция записи изображения в файл.

3. РАЗРАБОТКА И ОБОСНОВАНИЕ ВАРИАНТА СХЕМЫ ПАРАЛЛЕЛИЗАЦИИ АЛГОРИТМА

В настоящее время существуют два основных подхода к распараллеливанию вычислений. Это параллелизм данных и параллелизм задач.

Основная идея подхода, основанного на параллелизме данных, заключается в том, что одна операция выполняется сразу над всеми элементами массива данных.

Стиль программирования, основанный на параллелизме задач подразумевает, что вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей.

В случае с обработкой изображения целесообразнее будет разработать параллельный алгоритм, основанный на параллелизме данных. Входное изображение будет разделено на отдельные равные куски, которые будут независимо обработаны в каждом из процессов и возвращены в главный процесс для последующего объединения.

Параллельный алгоритм, согласно варианту задания, должен иметь 5 узлов. Соответственно, инициализируется 5 параллельных процессов, каждый из которых обработает свою часть изображения, затем отправит результат обработки в главный (нулевой) процесс. Тот, в свою очередь, соберёт 5 полученных частей в одно итоговое изображение и запишет его в итоговый файл.

В остальном параллельный алгоритм повторяет последовательный алгоритм (ввод, обработка, вывод).

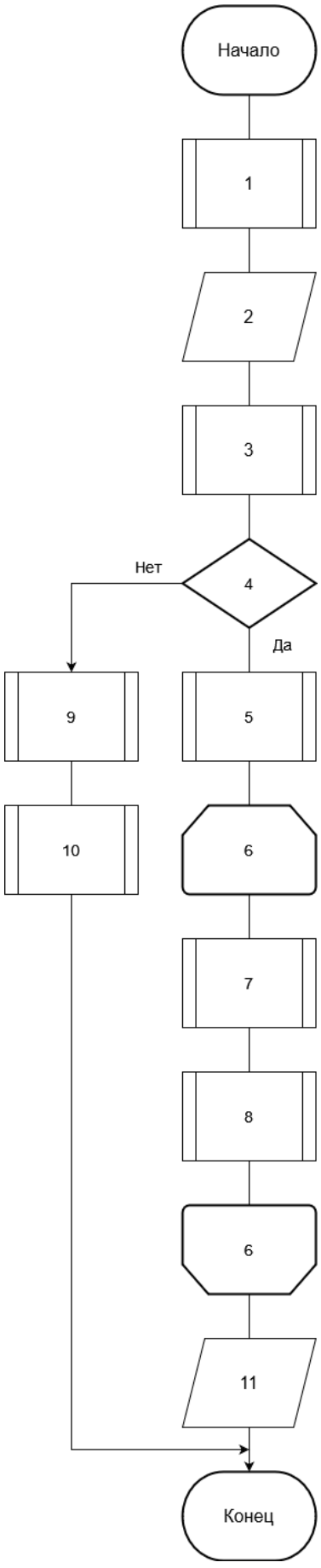

Рисунок 3 – параллельный алгоритм.

1) Инициализация MPI, получение ранга, количества процессов.
2) Чтение входного изображения.
3) Вырезка части изображения в соответствии с рангом.
4) Проверка текущего процесса на «главность» (ранг == 0).
5) Создание объекта итогового изображения, вставка своей части обработанного изображения.
6) Цикл i = 1..5
7) Получение сообщения от соответствующего текущему индексу процесса.
8) Присоединение полученной части к итоговому изображению.
9) Обработка части изображения (рис. 2).
10) Отправка части изображения главному (нулевому) процессу.

4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ СИСТЕМЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДАННЫХ

Для реализации параллельного алгоритма будем использовать библиотеку *MPI*.

Во-первых, инициализируем библиотеку MPI с помощью вызова функции *MPI_Init(&argc, &argv).*

Для обеспечения корректной работы программы необходимо получить общее количество процессов параллельного приложения и ранг текущего процесса. Производятся указанные действия с помощью функций *MPI_Comm_size (comm, *size)* и *MPI_Comm_rank (comm, *rank)* соответственно. Ранг (*rank*) и количество процессов (*size*) нужны для определения части изображения, которую должен обработать данный конкретный процесс, и последующего корректного соединения обработанных частей.

Разделение изображения на части происходит с помощью объекта класса *Rect (int _x, int _y, int _width, int _height)*, который задаёт точку начала и размеры участка изображения, который будет сохранён в качестве входного.

Далее происходит обработка изображения, порядок которой был описан в последовательном алгоритме.

Для межпроцессного взаимодействия воспользуемся процедурами:

*MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)* – отправка сообщения.

*MPI_Recv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)* – приём сообщения.

Главный процесс циклически принимает сообщения от остальных процессов и объединяет полученные части изображения в одно целое. Так как изображение разделено на равные части по горизонтали, то соединение будет происходит с помощью функции горизонтальной конкатенации *cv::hconcat(cv::InputArray src1, cv::InputArray src2, cv::OutputArray dst).*

5. ТЕСТИРОВАНИЕ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ СИСТЕМ С ПОСЛЕДОВАТЕЛЬНОЙ И ПАРАЛЛЕЛЬНОЙ ОБРАБОТКОЙ.

Рисунок 4 – исходное (входное) изображение.

Рисунок 5 – итоговое (результирующее) изображение.

Запустив полученную программную систему и отправив на вход изображение (рис. 4) получим в результате итоговое изображение (рис. 5)

Произведём тестирование производительности последовательного и параллельного алгоритма и занесём результаты испытаний в таблицу.

№	Последовательный алгоритм	Параллельный алгоритм
1	4766	1202
2	4795	1231
3	4754	1173
4	4764	1179
5	4785	1196
6	4767	1212
7	4773	1186
8	4765	1167
9	4762	1162
10	4754	1186
Среднее	4768,5	1189,4

Таблица 1 – результаты тестирования производительности программных систем с последовательной и параллельной обработкой.

Средняя производительность последовательного алгоритма:
$\lambda_1 = 1/T_1 = 1 / 4768,5 = 0,000209709552$
Средняя производительность параллельного алгоритма:
$\lambda_p = 1/T_p = 1 / 1189,4 = 0,000840760047$
Коэффициент увеличения производительности за счёт параллелизации:
$m = \lambda_p/\lambda_1 = T_1/T_p = 4768,5 / 1189,4 = 4,009164284513$

Вывод: параллелизация программы позволила увеличить её производительность чуть больше чем в 4 раза. Данная схема параллелизации эффективна.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проектирования были приобретены навыки по разработке и реализации параллельных алгоритмов для многокомпонентных систем обработки данных, разработке параллельных приложений с использованием интерфейса MPI.

В процессе выполнения проектирования была создана программа, реализующую параллельную обработку изображения на базе протокола MPI.

Был произведён сравнительный анализ временных характеристик выполнения обработки данных.

Приобретены знания и навыки обработки изображений с использованием библиотеки OpenCV.

ЛИТЕРАТУРА

1. OpenCV // Официальная документация. – 2023. [Электронный ресурс]. URL: https://docs.opencv.org/4.x/index.html (дата обращения: 03.05.2023)
2. Савицкий, Юрий Викторович. Методические указания для выполнения курсового проекта по дисциплине «Компьютерные системы и сети» для студентов специальности 1-40 01 01 «Программное обеспечение информационных технологий» – Брест, 2022 – 30 с.