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:

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

… (введение)

Задачей проектирования является создание программы, реализующей параллельную обработку изображения на базе протокола MPI.

Количество используемых вычислительных узлов – 5.

Тип графического формата (ч/б изображение) – bmp.

Размерность исходных данных – 3000*3000.

Этапы обработки:

Подавление зернистого шума графических изображений с использованием медианного фильтра.

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

Язык программирования – C++, библиотека MPI.

Алгоритмы обработки изображений – библиотека OpenCV.

1. РАЗРАБОТКА И ОПИСАНИЕ ПОСЛЕДОВАТЕЛЬНОГО АЛГОРИТМА ПРОГРАММНОЙ СИСТЕМЫ

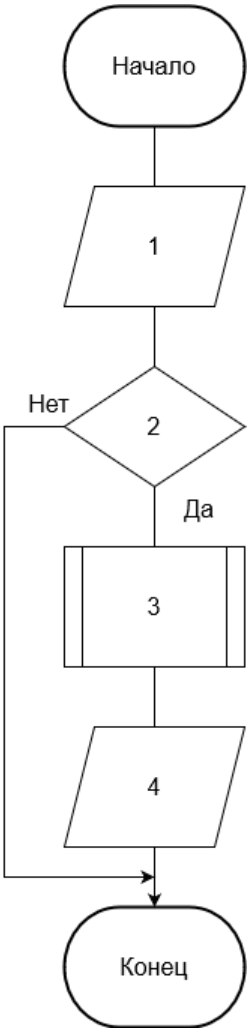

Рисунок 1 – последовательный алгоритм.

1) Чтение входного изображения.
2) Проверка правильности чтения.
3) Обработка изображения.
4) Запись обработанного изображения.

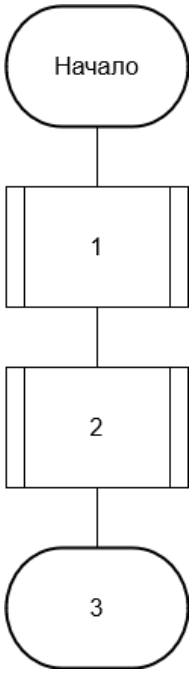

Рисунок 2 – алгоритм обработки изображения.

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

2. РАЗРАБОТКА ПРОГРАММНОЙ СИСТЕМЫ, РЕАЛИЗУЮЩЕЙ ПОСЛЕДОВАТЕЛЬНЫЙ АЛГОРИТМ ОБРАБОТКИ.

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

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

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 *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 потребуется произвести обработку с помощью фильтра Собеля по горизонтали и по вертикали 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

Параллельный алгоритм, согласно заданию, должен иметь 5 узлов. Соответственно, инициализируется 5 параллельных процессов, каждый из которых обрабатывает свою часть изображения, затем отправляет результат обработки в главный (нулевой) процесс. Тот, в свою очередь, собирает 5 частей в одно итоговое изображение и записывает его в файл.

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

4. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ СИСТЕМЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ДАННЫХ

Для реализации параллельного алгоритма будем использовать библиотеку *MPI*.

*int __stdcall MPI_Init(const int *argc, char ***argv)* – функция инициализации MPI.

*int __stdcall MPI_Comm_rank(MPI_Comm comm, int *rank)* – функция получения номера процесса.

*int __stdcall MPI_Comm_size(MPI_Comm comm, int *size)* – функция получения количества параллельных процессов.

rank и *size* нам нужны для определения части изображения, которую должен обработать данный конкретный процесс. Сделать это можно с помощью класса *cv::Rect2i::Rect_(int _x, int _y, int _width, int _height)*, который задаёт начало и размеры участка изображения.

Для межпроцессного взаимодействия воспользуемся функциями

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

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

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

Последовательный алгоритм – `10486 ms`.

Параллельный алгоритм – `2958 ms`.

Параллельный алгоритм превосходит последовательный примерно в 3,5 раза

(дополнить)

ЗАКЛЮЧЕНИЕ

ЛИТЕРАТУРА

1.