содержание

Введение. Анализ задач проектирования 4

1. Разработка и описание последовательного алгоритма программной системы 5

1.1. Общий алгоритм программы 5

1.2. Алгоритм создания инверсного изображения 5

1.3. Алгоритм подавления зернистого шума 5

2. Разработка программной системы, реализующей последовательный алгоритм обработки 6

3. Разработка и обоснование варианта схемы параллелизации алгоритма 7

3.1. Общий алгоритм программы 7

3.2. Алгоритм реализуемый процессами 7

4. Разработка программных модулей системы параллельной обработки данных с использованием MPI 8

5. Тестирование и сравнительный анализ производительности программы систем с последовательной и параллельной обработкой 9

Заключение 12

Литература 13

ПРИЛОЖЕНИЕ А ТЕКСТ ПРОГРАММЫ

Приложение Б Графический материал

Приложение В СХЕМА РЕСУРСОВ СИСТЕМЫ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ

# ВВЕДЕНИЕ

Целью данного курсового проекты было сравнение системы последовательной обработки данных с системой параллельной обработки данных. Оценивание будет произвоцдиться на основе времени затраченного каждой системы на идентичные входные данные.

В качестве входных данных было выбрано монохромное (черно-белое) изображение размером 5000 на 5000 пикселей в формате bmp. Код формирует инверсное изображение и подавляет зернистый шум с использованием медианного фильтра. Для параллельной системы использовалось 6 процессов.

Оценка будет производиться по времени выполнения кода и качеству выходного изображения. Ожидается прирост производительности без ухудшения качества изображения.

1. **Разработка и описание последовательного алгоритма программной системы.**

Ниже будут описаны алгоритмы последовательной обработки данных.

**1.1. Общий алгоритм программы.**

Шаг 1. Открываем изображение.

Шаг 2. Засекаем начальное время.

Шаг 3. Вызываем функцию, создающую инверсное изображение и подавляющую зернистый шум.

Шаг 4. Засекаем конечное время и вычитаем из него начальное, чтобы получить время выполнения.

Шаг 5. Сохраняем обработанное изображение.

Шаг 6. Конец алгоритма.

**1.2. Алгоритм создания инверсного изображения.**

Шаг 1. Итерируется по пикселям цикл.

Шаг 2. Выгружаются значения пикселей.

Шаг 3. Записывается новые значения, представляющие собой 255 –(минус) выгруженное значение пикселя.

Шаг 4. Конец алгоритма.

**1.3. Алгоритм подавления зернистого шума.**

Зернистый шум подавлялся при помощи медианного фильтра.

Шаг 1. Создается цикл итерации по пикселям.

Шаг 2. Выгружаются значения пикселей.

Шаг 3. В итерируемый пиксель записывается среднее арифметическое значение окружающих его пикселей.

Шаг 4. Записывается значение пикселя.

Шаг 5. Конец алгоритма.

1. **Разработка программной системы, реализующей последовательный алгоритм обработки.**

**Выбор средств программирования**

Программа разрабатывалась в среде PyCharm на языке python. Операционная система Windows 10.

В программе использовались библиотеки:

time – для использования времени.

PIL – для работы с изображениями.

Из стандартных функций нашли свое применение:

Image – для загрузки, сохранения и базовых преобразований изображения.

ImageDraw – для возможности рисования на открытом изображении.

time() – для определения настоящего времени.

1. **Разработка и обоснование варианта схемы параллелизации алгоритма**

**3.1. Общий алгоритм программы**

Шаг 1. Открываем изображение.

Шаг 2. Засекаем начальное время.

Шаг 3. Создаем 4 процесса.

Шаг 4. Делим изображение на 4 равных части.

Шаг 5. Вызываем функцию, создающую инверсное изображение и подавляющую зернистый шум.

Шаг 6. Засекаем конечное время и вычитаем из него начальное, чтобы получить время выполнения.

Шаг 7. Сохраняем обработанное изображение.

Шаг 8. Конец алгоритма.

**3.2. Алгоритм реализуемый процессами.**

Шаг 1. Открываем изображение.

Шаг 2. Создаем копию изображения.

Шаг 3. Считываем начальную координату «полосы» и ее ширину с высотой.

Шаг 4. Итерируем по пикселям секцию.

Шаг 5. Инвертируем секцию по алгоритму описанному в пункте 1.2.

Шаг 6. Заново проводим итерацию и подавляем шум по алгоритму описанному в пункте 1.3.

Шаг 7. Возвращаем обработанную секцию.

Шаг 8. Конец алгоритма.

1. **Разработка программных модулей системы параллельной обработки данных с использованием MPI.**

**Выбор средств программирования**

Программа разрабатывалась в среде PyCharm на языке python. Операционная система Windows 10.

В программе использовались библиотеки:

time – для использования времени.

PIL – для работы с изображениями.

multiprocessing – для создания процессов и работы с ними.

math – для математических функций.

Из стандартных функций нашли свое применение:

Image – для загрузки, сохранения и базовых преобразований изображения.

ImageDraw – для возможности рисования на открытом изображении.

time() – для определения настоящего времени.

Pool() – для создания процессов.

floor() – для округления в меньшую сторону.

**Разработка модулей**

Была определена функция **parral(tuple)**, возвращающая обработанную секцию изображения.

tuple – кортеж содержащий ширину секции и начальную координату.

Выходные значения: координата секции и сама секция.

1. **Тестирование и сравнительный анализ производительности программы систем с последовательной и параллельной обработкой.**

Тестирование производилось в PyCharm Community Edition 2019.1.2 x64 на ноутбуке HP Pavilion Power с операционной системой Ubuntu 21.04.

Процессор: Intel Core i5-8300H, 3900 MHz

Для тестирования было взято изображение размером 5000 на 5000 пикселей в формате bmp.



Рисунок 5.1 – изображение, использованное для тестирования

**Результат тестирования на системе последовательной обработки(func doit):**

Время: 82.26862692832947

Результат:



Рисунок 5.2 – изображение, обработанное последовательной системой

**Результат тестирования на системе параллельной обработки(parallel func):**

Время: 22.894911766052246

Результат:



Рисунок 5.3 – изображение, обработанное параллельной системой

# Заключение

В ходе выполнения данного курсового проекта был замечен рост производительности в 3 раза, однако нужно учитывать тот факт, что тратилось время на переключение между процессами и склеивание сегментов.

Изображения, полученные в результате работы вышли идентичными, из чего можно сделать вывод, что распараллеливание не влияет на качество результата.

Следовательно, использование параллельных алгоритмов является более выгодным чем использование последовательных, особенно при решении трудоемких задач.

**Список используемых источников**

1. ГОСТ 7.1-2003. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – Минск, 2004. – 48 с.
2. ГОСТ 7.82-2001. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – Минск, 2001 – 22 с.
3. 16.6. multiprocessing — Process-based “threading” interface // Python Documentation [Электронный ресурс]. – 2019. – Режим доступа: https://docs.python.org/2/library/multiprocessing.html. – Дата доступа: 10.04.2019.
4. Играемся с изображениями Python // Habr [Электронный ресурс]. – 2012. – Режим доступа: https://habr.com/ru/post/163663/. – Дата доступа: 05.04.2019.
5. Pillow // Pillow [Электронный ресурс]. – 2011. – Режим доступа: https://pillow.readthedocs.io/en/stable/. – Дата доступа: 08.04.2019.