Санкт-Петербургский государственный университет

Направление Математическое обеспечение и администрирование информационных систем
Профиль Технология программирования

Верещагин Константин Алексеевич

Параллельные алгоритмы бинаризации изображений

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Макаров А. А.

Рецензент:
к. ф.-м. н., доцент Евдокимова Т. О.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Technology in Programming

Konstantin Vereshchagin

Parallel algorithms for image binarisation

Bachelor's Thesis

Scientific supervisor:
professor Anton Makarov

Reviewer:
docent Tatjana Evdokimova

Saint-Petersburg
2016

Оглавление

Введение

Бинаризация изображения — преобразование исходного изображения в градациях серого в бинарное изображение, элементы которого могут принимать только два значения. Процесс бинаризации представляет значительный интерес для различных прикладных задач анализа изображений, поскольку позволяет некоторым образом классифицировать объекты на изображении, выделить полезную информацию. Практическое применение бинарные изображения находят в цифровой картографии, геоинформационных системах, пространственном анализе и других задачах, связанных с распознаванием образов [5], видеонаблюдением и видеокодированием [4]. Кроме того, цвет пиксела бинарного изображения кодируется одним битом информации, что даёт возможность эффективно сжать исходное изображение, понизив его битовую глубину.

Известно, что многим алгоритмам обработки изображений присущ естественный параллелизм, основанный на декомпозиции данных, позволяющий создавать параллельные реализации изначально последовательных алгоритмов. Для таких реализаций широко используются многоядерные процессоры, гетерогенные многопроцессорные системы, например, графические процессоры; распределённые системы и т.д. [10] Учитывая большие временные затраты, необходимые для обработки массивов изображений, требование к увеличению скорости бинаризации остается весьма актуальным.

1. Цель работы

- Изучение современных алгоритмов бинаризации и сравнение их характеристик.
- Параллельная реализация блочного алгоритма бинаризации на CPU при помощи технологии OpenMP.
- Параллельная реализация блочного алгоритма бинаризации на GPU при помощи технологии CUDA.
- Сравнение характеристик различных реализаций.

2. Существующие решения

Существуют различные подходы к бинаризации изображений, условно их можно разделить на две группы:

- пороговые методы,
- методы, основанные на равенстве яркостей.

2.1. Пороговые методы

В пороговых методах ищется некоторая характеристика (порог), позволяющая разделить пикселы изображения на два класса: фоновые и объектные. Порог может быть постоянным или адаптивным, зависеть от различных параметров изображения, например, гистограммы яркостей. Одним из самых быстрых и эффективных по качеству работы считается метод Оцу [11], основанный на минимизации внутриклассовой дисперсии.

Рис. 1: Исходное изображение.

Рис. 2: Бинарное изображение, полученное методом Оцу.

На примере работы алгоритма (рис. 2) видно, что метод Оцу, как и любые пороговые методы, предназначен в большей степени для задач, связанных с выделением объектов на изображении, распознаванием образов. При этом алгоритм игнорирует многие мелкие детали,

фон и объекты не всегда разделяются адекватно поставленной задаче, нарушается визуальная непрерывность изменения яркости вдоль изображения.

2.2. Методы, основанные на равенстве яркостей

В методах такого типа бинарное изображение строится так, чтобы его яркость была равна яркости исходного изображения. Под яркостью цифрового изображения здесь подразумевается сумма значений всех его пикселов. Такая расстановка позволяет добиться большего визуального сходства с оригиналом, выделить мелкие и слабоконтрастные детали, которые теряются при применении пороговых методов. Например, метод Байера [7] предлагает использовать определённые бинарные шаблоны с разной яркостью для построения бинарного изображения.

Рис. 3: Исходное изображение.

Рис. 4: Бинарное изображение, полученное методом Байера.

При этом разные блоки исходного изображения могут кодироваться одинаковыми шаблонами, что приводит к появлению характерной структуры – вертикальных или горизонтальных полос на стыке шаблонов (рис. 4). Более качественным с точки зрения визуального вос-

приятия является метод распространения ошибки [9], который позволяет точнее воспроизводить детали и лишён недостатков метода Байера (рис. 6).

Рис. 5: Исходное изображение.

Рис. 6: Бинарное изображение, полученное методом распространения ошибки.

Характерной особенностью метода распространения ошибки является его последовательное выполнение, так как результат обработки каждого следующего пиксела зависит от обработки предыдущего. На изображениях большого разрешения выполнение этого алгоритма может занять много времени, поэтому остаётся актуальной задача поиска сравнимого по качеству работы и эффективно распараллеливаемого метода.

Приведённые алгоритмы хорошо изучены, и для них существуют эффективные реализации, например, функция пороговой бинаризации cv::threshold в библиотеке OpenCV для языка C++ и функция DITHER в MatLab, реализующая метод распространения ошибки.

3. Блочный алгоритм бинаризации

Основным алгоритмом, рассматриваемым в данной работе, является блочный алгоритм бинаризации [6], который относится к методам, основанным на условии равенства яркостей.

Пусть $f = f(x, y)$, $x = 1..m$, $y = 1..n$ — исходное полутоновое изображение размера $m \times n$, а $g = g(x, y)$ — изображение, полученное в результате бинаризации.

1. Исходное полутоновое изображение разделяется на N неперекрывающихся блоков f_i, $i = 1..N$. Элементам блока, имеющим пустое пересечение с исходным изображением, присваивается нулевое значение. Соответствующий бинарный блок g_i имеет тот же размер, что и f_i, на данном этапе он заполняется нулями. Каждый блок после разбиения обрабатывается независимо.

2. В каждом блоке вычисляется суммарная приведённая яркость, определяющая количество единиц u_i в соответствующем бинарном блоке g_i:
$$u_i = \left\lfloor 2^{-k} \sum_{x,y \in f_i} f(x, y) \right\rfloor$$

3. В бинарном блоке g_i единицы ставятся на позиции, в которых находятся u_i наибольших значений яркостей $f(x, y)$ блока f_i , начиная с максимального значения. При этом если найдено несколько одинаковых наибольших значений, то среди всех вариантов один выбирается случайно в соответствии с некоторой вероятностью.

4. Из бинарных блоков g_i собирается полное бинарное изображение.

4. О распараллеливании блочного алгоритма

Рассмотренный алгоритм обладает естественной распараллеливаемой структурой, так как обработка каждого блока происходит независимо от остальных.

Важными характеристиками параллельного алгоритма являются время его работы на параллельной системе с p вычислительными модулями и ускорение, получаемое на такой системе [2]:

$$T_p = \frac{T_1}{p} + T_0, \qquad S_p = \frac{T_1}{T_p},$$

где T_1 – время последовательного исполнения, а T_0 – накладные затраты, например, на коммуникацию процессов.

При рассмотрении блочного алгоритма можно считать $T_0 = 0$, так как разные вычислительные модули обрабатывают выделенные им блоки независимо друг от друга. Для оценки времени выполнения остаётся найти T_1. Для обработки изображения размера $n \times m$ при выбраном значении длины стороны блока a потребуется

$$N = \left(\left\lfloor \frac{n}{a} \right\rfloor + 1\right)\left(\left\lfloor \frac{m}{a} \right\rfloor + 1\right) \approx \frac{mn}{a^2}$$

блоков. Рассмотрим вычислительную систему с $p = N$ процессорами. В блоке f_i выполняется a^2 операций сложения для подсчета суммарной приведённой яркости u_i (см. описание алгоритма). После этого выполняется поиск u_i максимумов, для чего требуется не более $a^2 u_i$ операций сравнения. Таким образом, общее время обработки одного блока равно

$$\tau_i = (u_i + 1)a^2 t,$$

где t-время выполнения элементарной операции. Для всего изображения получаем:

$$T_1 = \sum_{i=1}^{N} \tau_i = \sum_{i=1}^{N} (u_i + 1)a^2 t = \left(N + \sum_{i=1}^{N} u_i\right) a^2 t.$$

Тогда справедливо следующее равенство:

$$T_p = \frac{N\left(1 + \max\limits_i u_i\right) a^2 t}{p} = \left(1 + \max_i u_i\right) a^2 t.$$

Первое равенство обусловлено тем, что время работы параллельной системы определяется самым долгим временем обработки одного блока. Таким образом, можно оценить ускорение, получаемое при распараллеливании алгоритма на p вычислительных модулях:

$$S_p = \frac{T_1}{T_p} = \frac{\left(N + \sum\limits_{i=1}^{N} u_i\right) a^2 t}{\left(1 + \max\limits_i u_i\right) a^2 t} \leq p.$$

Из неравенства следует, что если $p \max_i u_i = \sum_{i=1}^{N} u_i$, то ускорение достигает p, и алгоритм является оптимальным по стоимости, так как $T_1 = pT_p$.

5. Параллельная реализация блочного алгоритма на CPU

Для проверки справедливости полученных соотношений на практике были разработаны параллельные CPU и GPU реализации блочного алгоритма.

5.1. Используемые технологии

Для реализации блочного алгоритма был выбран язык C++ с библиотекой OpenCV, широко применяемой для решения задач, связанных с обработкой изображений. Она предоставляет удобный программный интерфейс для работы с изображениями, например, процедуры и функции для матричного представления изображения в разных форматах, выделения отдельных частей изображения, применения различных фильтров. Также OpenCV предоставляет широкие возможности для реализации алгоритмов машинного обучения, видеообработки и компьютерного зрения [8].

Для организации параллельного выполнения программного кода использовалась библиотека OpenMP, предоставляющая набор директив препроцессора для распараллеливания последовательного алгоритма, набор переменных окружения и процедур. OpenMP позволяет легко тестировать параллельную программу, так как при отключении директив компилятор игнорирует их и рассматривает программу как последовательную.

Параллельная реализация блочного алгоритма состоит из двух этапов: сначала происходит предобработка, границы исходного изображения расширяются до значений, кратных длине стороны блока, после чего в цикле для каждого блока изображения вызывается процедура обработки, реализующая этапы алгоритма, описанные в разделе 3. Для распараллеливания независимых итераций цикла используется директива $\#pragma\ omp\ parallel\ num\_threads(n)$, где n - число параллельных потоков.

5.2. Результаты распараллеливания для разного числа потоков

Тестирование работы параллельной программы с различными параметрами производилось на компьютере с процессором Intel Core i5-6200U с двумя физическими ядрами и поддержкой технологии Hyper-Threading. В качестве исходного полутонового изображения было выбрано изображение размера 16200×10500.

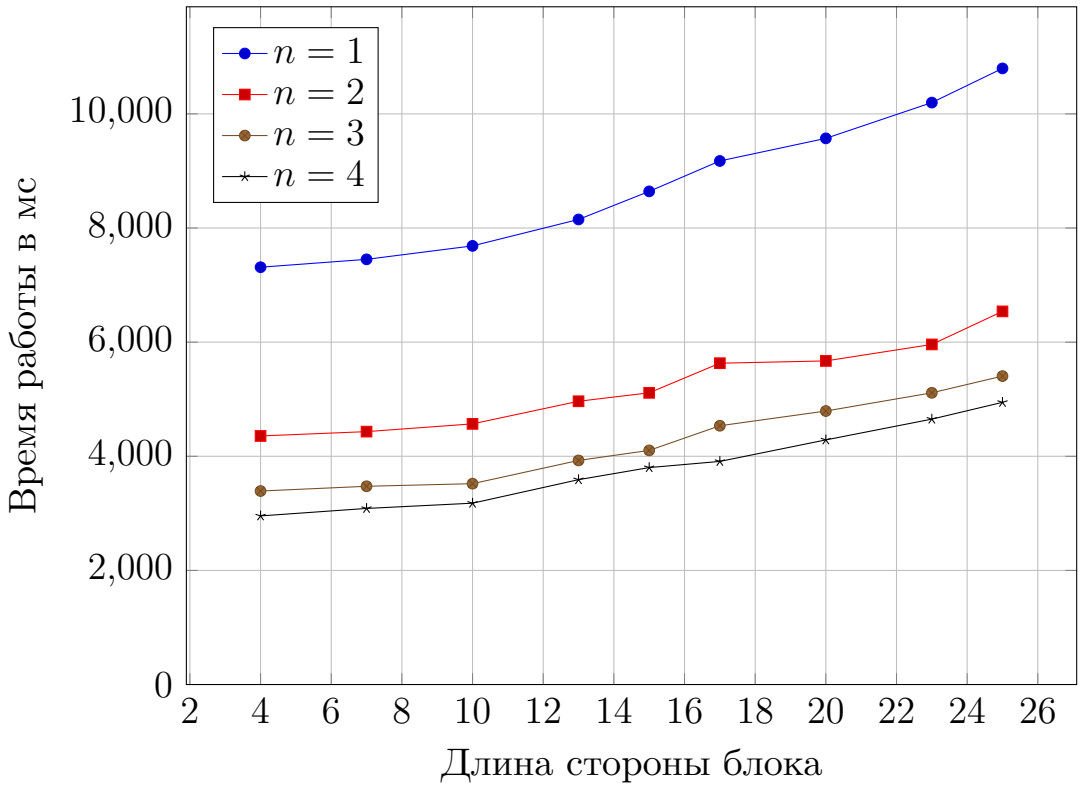


Рис. 7: Время работы алгоритма на n параллельных потоках

По рис. 7 видно, что на двухъядерном процессоре в целом удалось добиться ускорения в два раза, что подтверждает теоретическую оценку. Разница между временем работы при разном числе потоков оказалась небольшой, так как технология Hyper-Threading не позволяет в полной мере распараллелить процесс на 4 потока.

6. Параллельная реализация блочного алгоритма на GPU

Значительным недостатком при параллельной реализации для CPU является наличие малого числа вычислительных модулей в центральных процессорах, даже многоядерных, что не позволяет в полной мере добиться теоретического ускорения алгоритма. Эту проблему можно частично решить, используя возможности распараллеливания, предоставляемые современными графическими процессорами.

6.1. Параллельные вычисления на GPU

Изначально GPU (Graphics Processing Unit) использовались непосредственно для задач обработки компьютерной графики, трёхмерного ускорения, но, благодаря своей архитектуре, стали активно применяться в параллельных вычислениях. В отличие от CPU, графические процессоры обладают массивно-параллельной структурой, содержат большое число легковесных вычислительных модулей и имеют более сложную иерархию памяти. Для программной реализации параллельных алгоритмов на GPU используют специально разработанные средства, такие как кроссплатформенный фреймворк OpenCL и программно-аппаратную архитектуру CUDA для видеокарт фирмы NVidia.

Параллельные вычисления на графических процессорах применяются для решения разных задач компьютерного моделирования, математической физики [3], для нейросетевого обучения [1] и т.д.

Для реализации блочного алгоритма бинаризации использовалась видеокарта NVidia GeForce GTX 950m, поэтому был выбран программный интерфейс CUDA. Типичная программа на CUDA представляет из себя последовательную программу, работающую на CPU (host), которая вызывает специальные функции, запускаемые на графическом процессоре (device) и работающие по принципу SIMD (Single Instruction Multiple Data). При этом, необходимые данные, содержащиеся в опера-

тивной памяти, должны быть переданы в глобальную память GPU для дальнейшей обработки. При запуске этих функций программист может задать число и структуру потоков и блоков, в которые эти потоки объединяются. Блоки задают набор вычислительных модулей, которые будут обрабатывать конкретные потоки, выделяет для них разделяемую (shared) память и определяет набор операций, выполняемых в пределах одного блока, таких как барьерная синхронизация потоков.

6.2. Реализация блочного алгоритма

Подобная программная архитектура хорошо подходит для реализации блочного алгоритма: каждый пиксель изображения может обрабатываться одним потоком, все операции суммирования и нахождения максимума в одном блоке изображения выполняются соответствующим блоком потоков. Теоретически, эти операции можно выполнять параллельно, используя схему сдваивания [2], однако, на практике оказалось, что это не даёт никакого выигрыша из-за накладных расходов на копирование данных в разделяемую память и организацию циклов. Поэтому вместо реализации схемы сдваивания использовались атомарные операции суммы и максимума, предоставляемые CUDA.

Узким местом при реализации является этап случайной расстановки пикселов на определённые места в блоке, так как внутри части кода, выполняемого на GPU, не может быть вызвана стандартная функция языка C rand(). Это связано с тем, что данная функция использует некоторые состояния процессора, которые недоступны для видеокарты. Для генерации псевдослучайных чисел в CUDA есть специальная библиотека curand, но она работает довольно медленно, что значительно увеличивает общее время работы алгоритма. Более эффективным решением стала расстановка пикселей в том порядке, в котором срабатывают соответствующие им потоки при вызове операции атомарного уменьшения на 1 для счетчика оставшихся пикселей. Это позволило добиться высокой скорости, но ухудшило случайность распределения, так как потоки одного вычислительного модуля срабатывают одновремен-

но, из-за чего некоторые атомарные вызовы могут быть сериализованы в строгом порядке.

Код программы доступен в GitHub репозитории `https://github.com/KostyaVeresh`.

6.3. Эффективность распараллеливания

Для тестирования программы использовалась видеокарта NVidia GeForce GTX 950m, исходное изображение имеет размер 16200×10500.

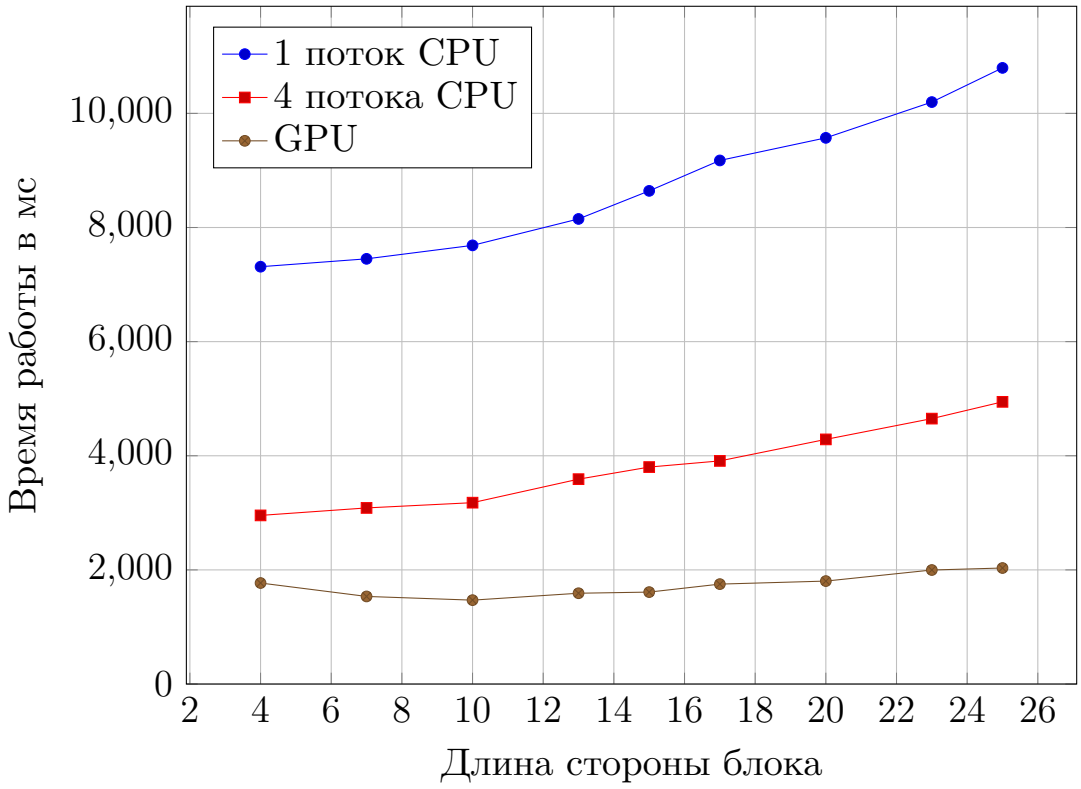
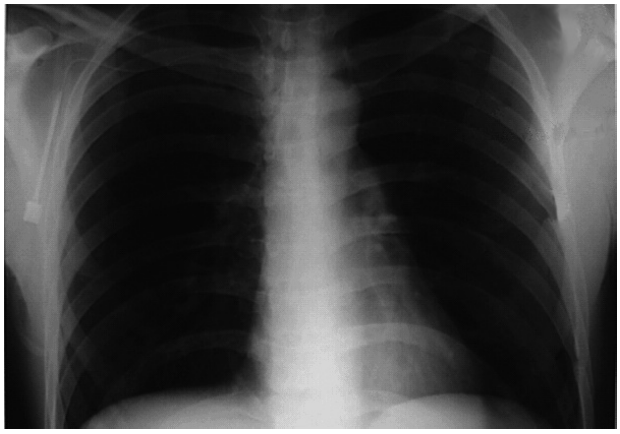
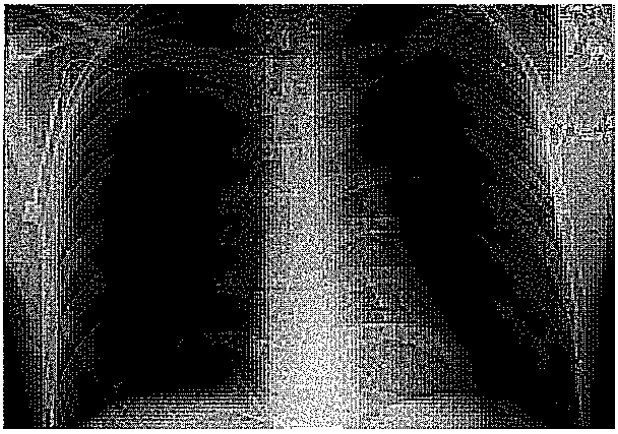


Рис. 8: Сравнение времени работы различных параллельных реализаций алгоритма

По рис. 8 видно, что GPU реализация оказалась более эффективной, алгоритм работает в среднем в 2 раза быстрее чем при реализации на CPU. Кроме того, время работы GPU алгоритма растёт заметно медленнее при увеличении размера блока. Поэтому при больших размерах блока ускорение достигает 5.5 раз.

7. Примеры работы блочного алгоритма





Заключение

В процессе работы был рассмотрены основные алгоритмы бинаризации изображений, для блочного алгоритма приведены оценки времени работы и ускорения, получаемого при распараллеливании на системе с p вычислительными модулями. Написаны параллельные реализации данного алгоритма для многоядерного CPU и для графического процессора. На практике было получено значительное ускорение, достигающее 5.5 раз при распараллеливании на GPU.

Список литературы

[1] Боресков А. В., Харламов А. А. Основы работы с технологией CUDA. — ДМК Пресс, 2016.

[2] Бурова И. Г., Демьянович Ю. К. Алгоритмы параллельных вычислений и программирование. — СПб.: Изд-во С.-Пб. ун-та, 2007.

[3] Перепёлкин Е.Е., Садовников Б.И., Иноземцева Н.Г. Вычисления на графических процессорах (GPU) в задачах математической и теоретической физики. — URSS, 2014.

[4] Ричардсон Я. Видеокодирование. H.264 и MPEG-4 – стандарты нового поколения. — Техносфера, 2005.

[5] Фурман Я. А., Юрьев А. Н., Яншин В. В. Цифровые методы обработки и распознавания бинарных изображений. — Красноярск: Изд-во Красноярск. ун-та, 1992.

[6] Яковлева Е. С., Макаров А. А. О свойствах блочного алгоритма бинаризации цифровых изображений // Компьютерные инструменты в образовании. — 2015. — № 4. — С. 26–36.

[7] Bayer B. An optimum method for two-level rendition of continuous-tone pictures // IEEE International Conference on Communications. — 1973. — Vol. 1. — P. 11–15.

[8] Dawson-Howe K. A Practical Introduction to Computer Vision with OpenCV. Wiley-IS&T Series in Imaging Science and Technology. — 1 edition. — Wiley, 2014.

[9] Floyd R., Steinberg L. An adaptive algorithm for spatial gray-scale // Proceedings Society Information Display. — 1976. — Vol. 17, no. 2. — P. 75–78.

[10] Juhasz Z. An analytical method for predicting the performance of parallel image processing operations // The Journal of supercomputing. — 1998. — Vol. 12. — P. 157–174.

[11] Otsu N. A threshold selection method from gray-level histograms // IEEE Trans. on System, Man and Cybernetics. — 1979. — Vol. SMC-9, no. 1. — P. 62–66.