электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель: Общероссийская общественная организация "Академия инженерных наук им. А.М. Прохорова". Эл No. ФС77-51036. ISSN 2307-0595

Методы повышения производительности морфологических алгоритмов

 # 03, сентябрь 2018

Бобков А. В.

УДК: 004.932

Россия, МГТУ им. Н.Э. Баумана
skliz@mail.ru

Аннотация: В статье рассматриваются способы повышения производительности морфологических алгоритмов в задачах технического зрения и производится оценка их эффективности для реализации на встраиваемых промышленных процессорах.

Рассматриваются типовые подходы повышения производительности морфологических алгоритмов с использованием параллельных операций над битовой упаковкой данных, логарифмической декомпозиции, метода интегрального изображения, использование кодирования «длина-значение». Проводится сравнительное исследование рассмотренных подходов как с использованием универсальных, так и встраиваемых процессоров, даются рекомендации по выбору того или иного подхода.

Статья может представлять интерес как для инженеров-разработчиков систем технического зрения, так и для студентов, изучающих курс систем распознавания образов.

Ключевые слова: распознавание изображений, техническое зрение, морфологические алгоритмы, бинарная морфология, параллельные алгоритмы.

Введение

В данной работе рассматриваются типовые морфологические алгоритмы и методы повышения их производительности для приложений технического зрения.

Интерес к методам обработки изображений в реальном времени в приложениях технического зрения в последнее время существенно возрос в связи с появлением большого количества доступных встраиваемых вычислительных систем, процессоров и микроконтроллеров, способных решать широкий спектр подобных задач.

Морфологические операции введены в теорию распознавания изображений в классической работе [1] и нашли широкое применение в современных практических приложениях – прежде всего, в задачах технического зрения [12], а также в биомедицинских приложениях [9], задачах распознавания текста [11] и маркировки [6] – практически везде, где имеется возможность заранее отделить точки объекта и фона. Математическая морфология предоставляет полноценный аппарат для всех этапов анализа изображения: для фильтрации, сег-

ментации, определения характеристик и распознавания. Морфологическая фильтрация подчас остаётся единственным средством для удаления крупных шумовых элементов. Вместе с тем, морфологические алгоритмы не слишком требовательны к вычислительным ресурсам, в силу чего занимают особое место в задачах технического зрения: вычислительные мощности, на которых будут работать эти алгоритмы, весьма ограничены, и вместе с тем требуется обеспечить работу в режиме реального времени. Кроме того, морфологические алгоритмы, в силу своей специфики, обладают высоким потенциалом распараллеливания, что позволяет им оставаться одним из основных инструментов обработки изображений в реальном времени.

Современная теория морфологических алгоритмов продолжает активно развиваться. Появляются высокопроизводительные алгоритмы, способные работать с полутоновыми изображениями [7] и использующие современные вычислительные архитектуры, прежде всего – графических ускорителей [8,10].

На данный момент известно большое количество разнообразных библиотек обработки изображений, таких как OpenCV, Leptonica, OCRopus, Image Processing ToolBox, однако использовать их для решения указанных задач зачастую не удаётся. Часть из них оптимизирована под универсальные процессоры и поддержку графических ускорителей, и они не могут работать на микроконтроллерах. Портируемые универсальные библиотеки, способные работать на широком спектре платформ, не могут учесть особенности системы команд и процесса их обработки, не учитывают особенности организации памяти целевого процессора и, как правило, не позволяют достичь требуемой производительности. Наконец, библиотеки непосредственно от производителей аппаратных средств хорошо оптимизированы, однако они содержат лишь небольшой набор базовых функций, которых, как правило, недостаточно для решения широкого спектра задач.

В этих условиях инженеру-разработчику приложений технического зрения важно иметь представление о методике разработки высокопроизводительных алгоритмов, позволяющей правильно выбирать и анализировать алгоритмы, уметь оценивать эффективность того или иного подхода в рамках конкретной задачи для выбранного целевого процессора и не быть зависимым от сторонних разработчиков библиотек.

В работе подытоживается опыт разработки морфологических алгоритмов для процессоров серии Blackfin от фирмы Analog Micro Devices. Это семейство имеет процессоры с архитектурой и набором команд, ориентированных на обработку и распознавание изображений, что позволяет рассматривать в качестве подходящей платформы для задач технического зрения.

В работе ограничимся рассмотрением морфологических операций со структурным элементом в форме квадрата с нечётной длиной стороны (3x3, 5x5, …), как наиболее распространённые на практике и дающие регулярный алгоритм. Однако предлагаемые подходы могут использоваться и для структурных элементов произвольных форм и размеров.

Рассмотрим основные способы ускорения производительности морфологических алгоритмов.

Использование таблиц заранее просчитанных результатов

Это наиболее общий метод, который применим ко всем морфологическим алгоритмам. В его основе лежит табулирование результата для всех возможных вариантов окрестности текущей точки [5].

Работу морфологического алгоритма можно представить следующим образом. Для каждой текущей точки A_0 считываются точки её окрестности $A_1..A_8$ и на основе их анализа принимается решение о значении в точке B_0. Значения в точках анализируемой окрестности принимают бинарные значения 0 и 1, и набор этих значений можно рассматривать как некоторый двоичный код $c=A_0..A_8$. Результирующее значение B_0 для каждого такого кода можно посчитать заранее и занести в таблицу T[c]. Размер таблицы в данном случае составит всего 2^9=512 байт. Алгоритм можно записать в следующем виде:

Для каждой текущей точки A_0 и её окрестности $A_1..A_8$:

-Построить двоичный код $c=A_0..A_8$,

-Прочитать значение B_0 из таблицы: B_0=T[c].

Данный подход является универсальным: различные морфологические алгоритмы будут отличаться только способом заполнения таблицы T.

К сожалению, для рассматриваемых алгоритмов – дилатации, эрозии, выделения границ – этот подход не позволяет достичь серьёзного прироста производительности: эти алгоритмы просты, и доля времени непосредственно на расчет результата, который мы заменили таблицей, весьма мала. Поэтому требуются более эффективные подходы.

Декомпозиция двухмерного структурного элемента на одномерные

Такая декомпозиция позволяет разбить двухмерный структурный элемент на два более простых одномерных: на элемент-строку и элемент-столбец [2,3]. В этом случае морфологическая операция выполняется в два независимых прохода: сначала – по строкам, затем – по столбцам изображения.

Декомпозиция позволяет весьма существенно сократить количество требуемых операций: с n^2 до 2n. Так, для дилатации по области 3x3 требуется 8 операций «И» (для девяти операндов), а при двух проходах по областям 1x3 и 3x1 – потребуется всего 2 операции в каждом проходе, суммарно – 4. При росте размера структурного элемента выигрыш становится ещё большим. Кроме того, более эффективной становится и работа с памятью – обращения выполняются к соседним ячейкам, что ещё больше повышает производительность.

Таким образом, декомпозиция на одномерные элементы является ключевым подходом к повышению производительности.

Использование параллельных вычислений

Может показаться, что параллельные вычисления используются исключительно на мощных универсальных компьютерах и что им не место в маломощных вычислителях, однако это не так. Сигнальные процессоры и промышленные микроконтроллеры также спо-

собны выполнять некоторые операции одновременно. Прежде всего это относится к побитовым логическим операциям «И», «ИЛИ», «НЕ»: они выполняются надо всеми битами машинного слова одновременно и параллельно. Если удается упаковать несколько точек изображения в одно машинное слово и представить алгоритм обработки в виде набора логических операций, то такой алгоритм сможет одновременно обрабатывать сразу насколько точек изображения [2].

Аналогичным свойством отчасти обладают и другие операции: например, сложение: если нет перехода за границы байтов, можно рассматривать сложение машинных слов как параллельное сложение их байтов. Однако здесь степень параллелизма существенно ниже, и использовать её гораздо сложнее.

Данный подход также повышает эффективность работы с памятью: за одну операцию чтения будут получены сразу несколько точек, причём доступ осуществляется сразу по всем линиям шины и к последовательно расположенным ячейкам.

Современные сигнальные процессоры являются, как правило, 32х разрядными. Это позволяет обрабатывать до 32х точек одновременно, повышая производительность до 32х раз. Более того, зачастую имеется возможность объединения пар 32х-разрядных регистров в один 64х-разрядный, позволяющий получить ещё более высокие скорости обработки.

Наряду с декомпозицией, параллельные вычисления – это основа повышения производительности морфологических алгоритмов.

Операции со структурными элементами большого размера

Морфологические операции со структурным элементом большого размера можно заменить последовательностью операций с небольшим окном (3х3). Однако это приводит к многократно повторяющимся совершенно излишним операциям. Поэтому требуются совершенно иные подходы, связанные либо с изменением алгоритма расчета характеристик окрестности (например, метод суммирования по области), либо – с дальнейшим расщеплением структурного элемента на иерархию более мелких компонент, обсчёт которых можно проводить параллельно. Рассмотрим эти подходы.

Использование суммирования по области

Этот подход предполагает сведение морфологических операций к операции суммирования точек окрестности, покрываемых единицами структурного элемента. При этом соседние суммы будут содержать большое количество общих точек, суммирование которых можно провести лишь один раз. В этом случае каждая последующая сумма может быть вычислена исходя из предыдущей путём добавления недостающих элементов и удаления излишних.

Алгоритм суммирования по области становится особенно эффективным в сочетании с декомпозицией структурного элемента на одномерные. В этом случае суммы считаются сначала по строке, а потом – по столбцу, и соседние суммы будут отличаться всего двумя элементами, независимо от размера области суммирования.

Особенность данного подхода – в том, что производительность перестаёт зависеть от размера структурного элемента: всегда требуется по 4 операции сложения на точку. Этот подход может применяться для структурных элементов большого размера.

Вместе с тем, параллельная реализация подобного подхода на современных сигнальных процессорах, видимо, невозможна. Поэтому данный подход зачастую не является наиболее быстрым. Однако он широко применяется на практике, причем не только для морфологических алгоритмов, но и практически повсеместно – для задач фильтрации, для расчета нормировок в задачах поиска объекта, для расчета признаков в задачах распознавания. Поэтому, несомненно, данный подход заслуживает отдельного рассмотрения.

Логарифмическая декомпозиция

Помимо декомпозиции структурного элемента на одномерные, возможна также и дальнейшая декомпозиция самих одномерных структурных элементов [van]. Рассмотрим в качестве примера структурный элемент размера 9: Se=[1 1 1 1 1 1 1 1 1]. Операция дилатации с таким структурным элементом будет сводиться к выполнению следующей логической операции:

$$b_4 = a_0 \mid a_1 \mid a_2 \mid a_3 \mid a_4 \mid a_5 \mid a_6 \mid a_7 \mid a_8 ,$$

где символом «|» обозначена операция «ИЛИ».

Воспользовавшись коммутативностью операции «ИЛИ», расставим скобки следующим образом:

$$b_4 = (a_0 \mid a_1 \mid a_2) \mid (a_3 \mid a_4 \mid a_5) \mid (a_6 \mid a_7 \mid a_8).$$

Операции в скобках соответствуют обычной дилатации со структурным элементом размера 3: [1 1 1]. Операция же над результатами в скобках соответствует дилатации со структурным элементом вида [1 0 0 1 0 0 1] из семи элементов. Этот структурный элемент содержит три близкорасположенные единицы, и соответствующая операция дилатации может выполняться по той же схеме, что и обычная дилатация, без существенного снижения производительности.

В целом, для выполнения дилатации со структурным элементом размера 9x9 подобная схема потребует 4 прохода (2 по строкам и 2 по столбцам) по 2 операции «ИЛИ» в каждом проходе, суммарно – 8 операций на точку. Для сравнения, прямой алгоритм со структурным элементом 9x9 потребовал бы 9^2-1=80 операций «ИЛИ», а алгоритм со структурным элементом 3x3 – 4 прохода по 4 операции в каждом, суммарно – 16 операций. При большем размере структурного элемента выигрыш оказывается ещё существеннее.

Очевидно, что данный подход даёт наибольший прирост производительности при размере структурного элемента, кратном 3^n. При других размерах длина структурного элемента сокращается, но количество единиц в нём, а значит – и количество операций – остаётся прежним.

Разумеется, можно расставить скобки и другим способом, например – по два элемента, по пять. Таким образом возможно оптимизировать алгоритм для желаемого размера структурного элемента.

Логарифмическая декомпозиция полностью совместима с параллельным вычислением и оказывается очень удобным и эффективным инструментом для морфологических операций с окном большого размера.

Алгоритм с кодированием значение-длина

Подход, описанный в работе [4], предусматривает предварительное представление изображения не в виде отдельных изолированных точек, а в виде указания значения пикселя и количества таких пикселей, идущих подряд. Такое представление позволяет обрабатывать не изолированные точки, а сразу большие группы. Операции в этом случае проводятся только над строками. Для операций над столбцами изображение транспонируется до и после обработки.

Операция дилатации по окну NxN будет сводиться к коррекции длин последовательностей нулей и единиц. Последовательности нулей длиной меньше N будут удаляться, а последовательности единиц справа и слева от них будут объединяться с добавлением количества нулевых точек. Длина последовательностей нулей большего размера уменьшается на N симметрично справа и слева, а длина соответствующих последовательностей единиц справа и слева, соответственно, наращивается.

Алгоритм очень эффективно обрабатывает строки данных, однако, опять же, требует существенных затрат на реализацию транспонирования.

Описание алгоритмов

В этом разделе будут описаны алгоритмы, построенные на основе вышеописанных подходов.

Введём следующие обозначения. Пусть A – исходное изображение размера NxN, B – результат его обработки (того же размера). Пусть на изображении точки объектов помечены единицами, а точки фона – нулями.

Обозначим текущую точку как A_0. Каждая точка исходного изображения по очереди будет становиться текущей точкой A_0. Результат обработки точки A_0 обозначим как B_0.

Обозначим точки, соседние с A_0, как $A_1..A_8$. Пусть точка A_1 находится справа от точки A_0, а остальные точки нумеруются против часовой стрелки, как показано на рис. 1.

A_6	A_7	A_8
A_5	A_0	A_1
A_4	A_3	A_2

Рис. 1. Обозначение точек окрестности

Параллельный алгоритм дилатации с окном 3x3

Начнем рассмотрение алгоритмов морфологии с алгоритма дилатации для области размера 3x3. С одной стороны, это наиболее распространённый алгоритм, а с другой – он достаточно прост и позволяет получать регулярные вычисления. Этот алгоритм можно записать в следующем виде:

Для каждой точки A_0:

Если A_0=1 **или** любая из точек 8-окрестности A_i=1,

То B_0 = 1,

Иначе B_0 = 0.

Из алгоритма видно, что результатом B_0 будет результат логической операции «ИЛИ» по всей области 3x3:

$$B_0 = A_0 \mid A_1 \mid A_2 \mid A_3 \mid A_4 \mid A_5 \mid A_6 \mid A_7 \mid A_8 , \quad (1)$$

где символом «|» обозначена операция «ИЛИ».

Легко заметить, что элементы строк $A_6 \mid A_7 \mid A_8$, $A_5 \mid A_0 \mid A_1$ и $A_4 \mid A_3 \mid A_2$ используются в вычислениях трижды: они потребуются не только для элемента B_0, но и для элементов над ним (B_7) и под ним (B_3). Поэтому расчёт B_0 по формуле (1) целесообразно разбить на два прохода: сначала – вычисление «ИЛИ» по строкам, а затем – по столбцам от полученного результата:

$$B_0 = (A_6 \mid A_7 \mid A_8) \mid (A_5 \mid A_0 \mid A_1) \mid (A_4 \mid A_3 \mid A_2), \quad (2)$$

Назовём эти операции дилатацией по строкам и дилатацией по столбцам соответственно.

Будем считать, что и изображение, и результат представлены в формате 1 бит на пиксель. Это условие, в принципе, не является обязательным, однако обеспечивает максимальную производительность. Допустим также, что ширина изображения кратна разрядности целевого процессора (если это не так, то изображение просто дополняется до нужной ширины нулями).

В этом случае дилатация по строкам вида ($A_6 \mid A_7 \mid A_8$) может выполняться параллельно сразу для нескольких элементов. Обозначим элементы строки исходного изображения маленькими буквами $a_0..a_{N-1}$, чтобы отличать их от точек окрестности текущей точки (для первой точки $A_6=a_0$, $A_7=a_1$, $A_8=a_2$). Для вычисления параллельной дилатации по строкам считаем сразу несколько элементов $a_0..a_{n-1}$ в одно машинное слово, сдвинем их вправо и влево на один разряд, и выполним операцию «ИЛИ» (рис.2)

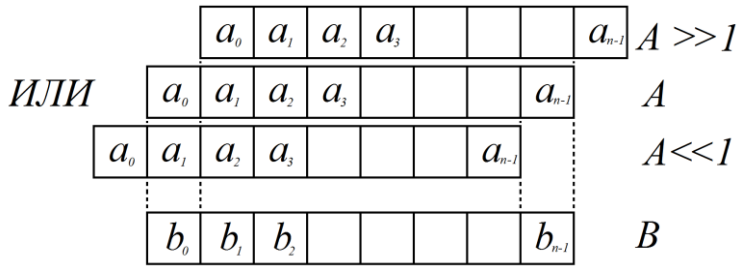

Рис. 2. Параллельная дилатация по строкам

Операция «ИЛИ» будет выполнена одновременно и параллельно надо всеми элементами $a_0..a_{N-1}$, и результат дилатации строк $b_0..b_{N-1}$ также будут получены одновременно.

Однако легко заметить, что не все элементы b будут вычислены корректно: для элементов b_0 и b_{n-1} не хватит одного элемента (a_{-1}..и a_n соответственно). Поэтому необходима дополнительная коррекция: для элемента b_0 выполняется дополнительная операция «ИЛИ» со

старшим разрядом a_{n-1} предыдущей порции данных $a_0..a_{n-1}$, а для b_{n-1} – операция «ИЛИ» с младшим разрядом a_0 последующей порции (рис.3).

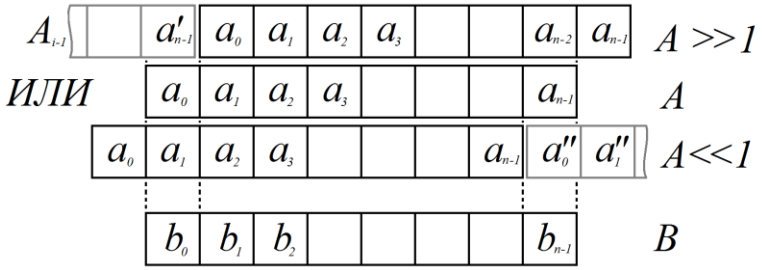

Рис. 3. Учёт крайних разрядов

Заметим, что при реализации коррекции на языке ассемблер целевого процессора учёт крайних разрядов может быть выполнен более просто – например, совмещён с операцией сдвига с использованием флага переноса. В этом случае дополнительные операции «ИЛИ» не потребуются. Однако здесь конкретная реализация уже будет сильно зависеть от целевого процессора.

Величина n выбирается исходя из максимального размера данных, над которыми процессор умеет выполнять сдвиги и операцию «ИЛИ»: обычно это 32 или 64 бита. Таким образом, все 32 (64) операции «ИЛИ» будут выполняться параллельно, обеспечивая ускорение не менее 32 (64) раз.

Последние элементы строк для изображения, размер которого не кратен n, могут быть обработаны обычным последовательным алгоритмом. Их количество крайне незначительно, и они не окажут существенного влияния на общее время вычислений. Альтернативный вариант – заполнить недостающие элементы нулями до ближайшего размера, кратного n. Это никак не повлияет на результат дилатации, но придаст алгоритму более компактный и регулярный вид.

Дилатация по строкам выполняется аналогично, однако она даже не требует операций сдвига. Считываются результаты дилатации строк, полученных на предыдущем этапе, и для них выполняется операция «ИЛИ». Полученный результат $c_0..c_{n-1}$ и есть требуемый результат дилатации (рис. 4)

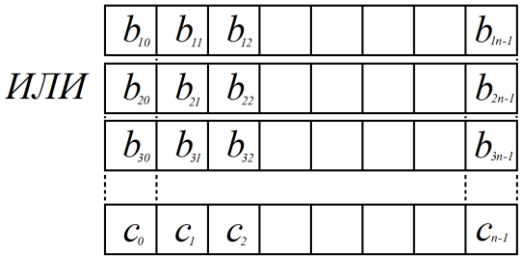

Рис. 4. Параллельная дилатация по столбцам

Эта операция также производится параллельно и одновременно для всех элементов n, обеспечивая ускорение в n раз.

Дилатация по столбцам для первой и последней строки выполняется отдельно: для первой строки будут отсутствовать элементы верхней строки $b_{1,0}..b_{1,n-1}$, для последней – элементы нижней строки $b_{3,0}..b_{3,n-1}$.

Приведённый алгоритм требует дополнительного буфера для хранения промежуточных результатов дилатации строк b_i. Однако можно построить алгоритм таким образом, что он будет использовать только два буфера A и B, как и в обычном алгоритме дилатации. При этом буфер B будет использоваться для хранения результатов дилатации строк, а результат c_i будет записываться обратно в A поверх исходного изображения.

Параллельный алгоритм эрозии с окном 3x3 записывается следующим образом:

Для каждой точки A_0:

Если $A_0=1$ **и** все точки 8-окрестности $A_i=1$,

То $B_0 = 1$,

Иначе $B_0 = 0$.

Видно, что алгоритм эрозии будет полностью аналогичен рассмотренному алгоритму дилатации, за исключением того, что вместо операции «ИЛИ» следует использовать операцию «И»:

$$B_0 = A_0 \,\&\, A_1 \,\&\, A_2 \,\&\, A_3 \,\&\, A_4 \,\&\, A_5 \,\&\, A_6 \,\&\, A_7 \,\&\, A_8 ,$$

откуда

$$B_0 = (A_6 \,\&\, A_7 \,\&\, A_8) \,\&\, (A_5 \,\&\, A_0 \,\&\, A_1) \,\&\, (A_4 \,\&\, A_3 \,\&\, A_2). \qquad (3)$$

Здесь и далее символом «&» обозначена операция «И».

При программной реализации операции дилатации и эрозии будут отличаться способом реализации сдвига. Дело в том, что в свободные ячейки после сдвига заносятся нули: они не влияют на результат операции «ИЛИ» в дилатации, а вот на результат операции «И» в эрозии они влиять будут. Поэтому при реализации сдвига в эрозии (а также в алгоритме детектирования границ, рассмотренном ниже), на место освободившихся при сдвиге разрядов должны быть записаны данные из предыдущего (или последующего) машинного слова:

Для дилатации:

```
B=A2|(A2>>1)|(A2<<1) | (A3<<63)|(A1>>63);
```

Для эрозии:

```
B=A2&((A2>>1)|(A3<<63))&((A2<<1)|(A1>>63));
```

Здесь A1 – предыдущее машинное слово, A2 – текущее, A3 – следующее, B – результат обработки.

Параллельный алгоритм выделения границ

Морфологический алгоритм выделения границ находит широкое применение, поскольку он производит очень качественные границы (тонкие, замкнутые и не содержащие разрывов) и имеет неоспоримые преимущества перед другими детекторами. Поэтому для бинаризованных изображений следует использовать именно этот алгоритм.

Рассмотрим, как можно повысить скорость работы этого алгоритма. Его можно записать в следующем виде:

Для каждой точки A_0:

Если $A_0=1$ **и** хоть одна точка из 8-окрестности $A_i=0$,

То $B_0 = 1$,

Иначе $B_0 = 0$.

Здесь условие – более сложное, чем для эрозии и дилатации, оно не сводимо к какой-то одной бинарной операции, однако и его удаётся преобразовать к виду, допускающему параллельные вычисления.

Как и в случае с дилатацией, добавим условие для 8-окрестности также и центральную точку A_0. Очевидно, что результат от этого не изменится: если $A_0=1$, то это не повлияет на результат проверки окрестности. Если же $A_0=0$, то точка не является точкой объекта (первое условие) и заведомо не содержит границ. Тогда формула для расчета B_0 примет вид:

$$B_0 = A_0 \,\&\, \neg[(A_6 \,\&\, A_7 \,\&\, A_8) \,\&\, (A_5 \,\&\, A_0 \,\&\, A_1) \,\&\, (A_4 \,\&\, A_3 \,\&\, A_2)]. \qquad (4)$$

Второй компонент формулы под знаком логической инверсии «$\neg$» представляет собой ни что иное, как результат эрозии (3). Он вычисляется по рассмотренной выше схеме: сначала вычисляется эрозия по строкам, затем – по столбцам.

Затем результат эрозии следует инвертировать и выполнить операцию «И» с исходным изображением. Инвертирование может быть осуществлено при помощи операции «Исключающее ИЛИ» с набором единиц (для языков высокого уровня), либо при помощи соответствующей команды ассемблера.

Формально мы получаем результат побитного вычитания результата эрозии из исходного изображения, однако делаем это при помощи побитовых операций «И», «ИЛИ», «НЕ», чтобы сделать возможным параллельную обработку всех битов.

Схема параллельного алгоритма выделения границ представлена на рис. 5.

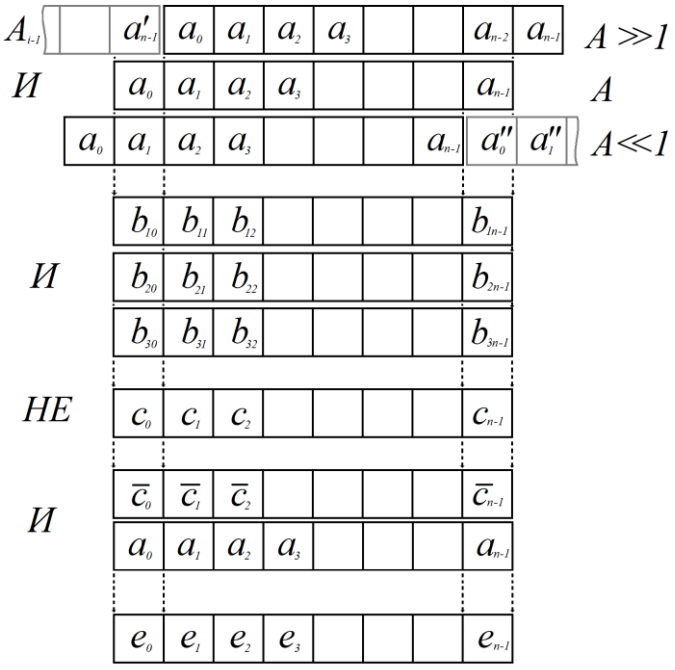

Рис. 5. Схема параллельного алгоритма выделения границ

Данный алгоритм потребует уже четыре буфера – исходное изображение, эрозия строк, эрозия столбцов, результат. Однако подобно алгоритмам эрозии и дилатации, требования к памяти можно немного сократить, если разрешить накапливать результат поверх исходного изображения A. В этом случае результат эрозии строк пишется в массив B, а результат эрозии столбцов инвертируется, конъюгируется с точками исходного изображения и записывается обратно на их место. В этом случае исходное изображение A будет утеряно, а результат обработки будет записан поверх него.

Алгоритмы эрозии и дилатации с суммированием по окну большого размера

Несмотря на то, что рассмотренные выше алгоритмы эрозии и дилатации достаточно производительны, при их многократном применении кратно возрастают и затраты на обращение к памяти. Поэтому при большом количестве проходов (или, что тоже самое, при большом размере окна) алгоритм следует строить по несколько другой схеме.

Перепишем алгоритмы в следующем виде.

Алгоритм дилатации по окну размера pxp:

Для каждой точки A0

Найти сумму σ соседних точек по заданной окрестности

Если σ>0, то B_0=1, иначе B_0=0

Алгоритм эрозии по окну размера pxp:

Для каждой точки A_0

Найти сумму σ соседних точек по заданной окрестности

Если σ=p x p, то B_0=1, иначе B_0=0

Таким образом, для выполнения дилатации и эрозии, теперь для каждой точки необходимо найти сумму σ соседних точек по заданной окрестности.

Как и прежде, дилатацию (эрозию) будем разбивать на последовательность дилатации строк и дилатации столбцов. Однако сами строки и столбцы будем рассчитывать иначе: суммированием элементов строки (столбца).

Пусть для определённости размер окна структурного элемента – нечётный и имеет размер p=2m+1. Дилатация и эрозия с таким размером окна соответствуют m дилатациям и эрозиям с окном 3x3. Тогда i-тая сумма элементов строки будет равна:

$$S_i = \sum_{k=i-m}^{i+m} a_k$$

Первую сумму S_0, соответствующую первому элементу строки a_0, посчитаем обычным суммированием элементов $a_0..a_m$:

$$S_0 = \sum_{i=0}^{m} a_i$$

Точки a_i за пределами изображения будем считать заведомо не принадлежащим объектам изображения, поэтому они будут иметь разметку 0 и не будут учитываться в сумме S_0 и последующих вычислениях.

Следующая сумма S_1 (соответствующая элементу a_1) будет отличаться от S_0 всего на один элемент a_{m+1} (рис. 6а). Поэтому суммировать элементы строки заново нет необходимости:

$$S_1 = S_0 + a_{m+1}.$$

То же самое будет справедливо и для сумм S_2, S_3, … S_m:

$$S_{i+1} = S_i + a_{m+1},\ i=0..m.$$

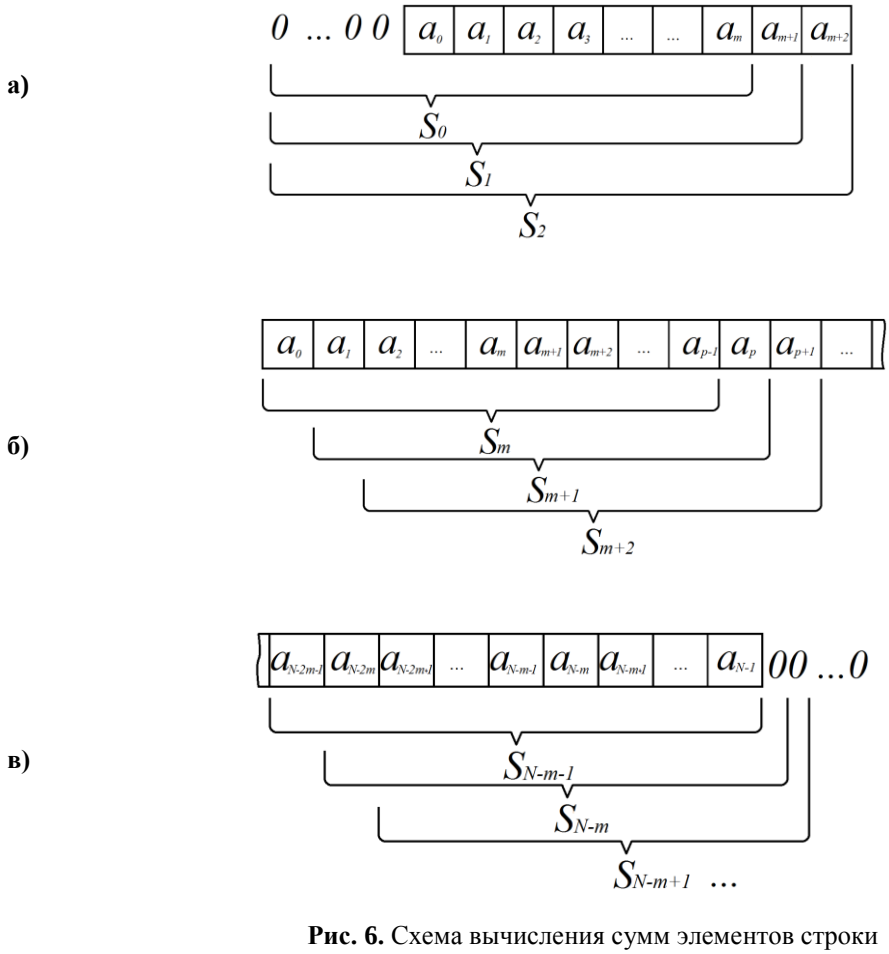

Рис. 6. Схема вычисления сумм элементов строки

Суммы, начиная с S_{m+1}, будет отличаться от суммы S_m уже двумя элементами – a_0 и a_p (рис.6б):

$$S_{m+1} = S_m - a_0 + a_p,$$

$$S_{m+i+1} = S_{m+i} - a_i + a_{p+i-1},\ i=0..N-p-1.$$

Тут хорошо виден смысл замены логических операций И/ИЛИ на арифметические: для логических операций подобная схема невозможна.

Наконец, оставшиеся m сумм вновь отличаются только на один элемент, так как второй попадает за границу изображений, и его разметка принимается равной нулю (рис. 6в):

$$S_{m+i+1} = S_{m+i} - a_i\ ,\ i=N-p-1..N-1.$$

Таким образом, для вычисления сумм строк $S_0..S_{N-1}$ потребуется не более двух операций на точку.

Расчет сумм по столбцам может быть построен полностью аналогично, но теперь вместо исходных данных будут выступать не исходная разметка, а найденные суммы строк.

Вместе с тем, при расчёте сумм столбцов могут проявляться особенности организации памяти целевой архитектуры: время обращения к памяти при последовательном чтении (по строкам) и при произвольном доступе (по столбцам) при обращении к крупным массивам может отличаться на порядки. Такой эффект появляется, например, в защищённом режиме универсальных компьютеров. Поэтому разработчику важно знать, какой алгоритм выбрать в зависимости от целевой архитектуры.

Чтобы преодолеть указанную проблему, есть по меньшей мере две возможности:

1) Транспонировать матрицу сумм строк. В этом случае алгоритм примет предельно простой вид – двукратное суммирование строк с последующим транспонированием результата, однако появятся дополнительные затраты времени на транспонирование.

2) Изменить порядок следования циклов при суммировании в столбцах. При этом сначала будут суммироваться столбцы одной строки, и обращение к памяти будет попадать в соседние ячейки. Однако в этом случае потребуется дополнительный буфер для хранения текущих сумм столбцов. Размер буфера будет равен ширине изображения.

Алгоритм нахождения сумм столбцов в этом случае будет иметь вид:

1.Заполнить буфер Q нулями.
2.Для каждой строки i=0..N-1
    Для каждого столбца j=0..N-1
        Считываем $\sigma = Q_i$ из буфера
        Корректируем:
        σ= sys (S +s, S – s + s, S-s)
        Для дилатации: Если $\sigma>0$, то $B_0=1$, иначе $B_0=0$
        Для эрозии: Если $\sigma=p \times p$, то $B_0=1$, иначе $B_0=0$
        Сохраняем σ в буфер Q_i

Такой алгоритм будет эффективным, если массив Q попадает в кэш или статическую память. В противном случае количество обращений к памяти может оказаться слишком большим и сведёт на нет весь выигрыш в вычислениях; в этом случае стоит предпочесть первый вариант с использованием транспонирования.

Суммирование столбцов также можно выполнять параллельно, используя параллельные возможности сумматора. Для этого отдельные байты буфера Q следует объединить в машинные слова: теперь их можно считывать и суммировать параллельно и одновременно. Результат будет корректным, если величина наибольшей суммы не выходит за пределы байта.

Однако за пределы байта будет выходить уже количество единиц в окне размером 16x16. Но в этом случае есть простой выход: можно поменять местами проходы по строкам и по столбцам: теперь первый проход будет выполняться параллельным суммированием по

столбцам, второй – последовательный по строкам. В этом случае алгоритм может использовать окна размером до 255x255 включительно.

Прирост производительности здесь не столь большой, как для логических операций, поскольку одновременно суммируются 4 либо 8 байт, но тем не менее он есть.

Рассмотренный алгоритм обладает высокой производительностью, практически не зависящей от размера окна. Алгоритм требует порядка четырёх операций на точку и при аккуратном учёте особенностей работы с памятью может обеспечить высокую производительность для окна большого размера.

Экспериментальное определение характеристик

В качестве тестового изображения возьмём массив размера 1024x1024, равномерно заполненный нулями и единицами в произвольном порядке. Для параллельного алгоритма эрозии и дилатации будем использовать упакованное изображение (один бит на пиксель), для остальных – обычное (один байт на пиксель). Каждый алгоритм запускался 1000 раз, замерялось общее время, и вычислялось среднее время, приходящееся на один проход. Для алгоритма параллельной дилатации использовались 64x разрядные слова (unsigned long long). Для алгоритма дилатации с суммированием использовался размер окна 11x11 (он практически не влияет на время обработки). Замеры производились на универсальных процессорах i5-2,27 ГГц, i7-3,6ГГц, а также на микроконтроллере BlackFin BF707. Результаты измерений занесены в таблицу 1.

Таблица 1. Время работы морфологических алгоритмов

№	Название	Время обработки, мс		
		i7-3,6	**i5-2,27**	**BF**
1	Стандартный алгоритм дилатации	45	97	3 400
2	Параллельная дилатация по окну 3x3	0,22	0,37	14
3	Параллельная эрозия по окну 3x3	0,22	0,36	14
4	Параллельное выделение границ	0,22	0,42	18
5	Параллельная дилатация по окну 5x5	0,31	0,56	24,5
6	Параллельная дилатация по окну 9x9 с логарифмической декомпозицией	0,41	0,83	27,4
7	Дилатация суммированием по окну (без оптимизации)	13	26	530
8	Дилатация суммированием по окну (с транспонированием)	9,8	18	900
9	Дилатация суммированием по окну (с изменением порядка циклов и параллельным расчетом столбцов)	2,7	9	310

Из приведённой таблицы видно, что стандартный алгоритм дилатации работает крайне медленно, и плохо подходит для задач реального времени. Алгоритм параллельной дилатации позволяет исправить этот недостаток: производительность повышается в 250 раз. Это происходит как за счет использования параллельных логических операций, так и за счет

разбивки на два независимых прохода. Кроме того, рассмотренный алгоритм очень эффективно работает с памятью: ячейки считываются последовательно по 64х разрядной границе данных, с минимальным количеством циклов чтения и записи. Этот алгоритм оказывается крайне эффективным, и именно его следует брать за основу при прочих равных условиях. Параллельные алгоритмы эрозии и выделения границ, построенные по этой схеме, также обладают высокой производительностью. Это позволяет обрабатывать десятки кадров изображения в секунду, что очень важно для приложений реального времени.

Параллельные алгоритмы можно строить и для большего размера структурных элементов (5х5, 7х7), однако видно, что их эффективность достаточно быстро снижается.

Алгоритм суммирования по большому окну показывает довольно незначительный прирост производительности: он всего в несколько раз быстрее стандартного алгоритма дилатации. Однако беглый анализ показывает, что более 90% времени алгоритм тратит на суммирование столбцов. Эти затраты вызваны, как было указано выше, крайне неэффективным доступом к памяти.

Для исправления ситуации, как было указано выше, матрицу строк можно транспонировать. Производительность алгоритма с транспонированием сразу возрастает почти в полтора раза. Однако само транспонирование оказывается сравнительно медленным: оно также требует обращений по столбцам. Транспонирование занимает почти половину времени работы алгоритма. Для процессора BlackFin ситуация с транспонированием несколько иная: здесь разница между временем чтения строк и столбцов практически отсутствует, и использовать алгоритм с транспонированием не имеет смысла: он будет работать даже медленнее, чем исходный неоптимизированный алгоритм.

Альтернативой является изменение порядка следования циклов и введение буфера сумм для столбцов. Время вычислений в этом случае также сокращается в два раза. При этом суммирование столбцов по-прежнему выполняется медленнее строк, но уже всего в полтора раза. Это плата за операции с буфером.

Для нашего случая дилатация суммированием оказалась менее эффективной по сравнению с параллельной дилатацией: она становится более выгодной только при очень большом количестве проходов (25 и более), что на практике встречается относительно редко.

Из таблицы видно, что гораздо более эффективный подход для операций с большим размером окна связан с использованием логарифмической декомпозиции. Так, параллельный двухпроходный алгоритм с декомпозицией структурного элемента 9х9 оказывается в 1,8 раз быстрее 4х проходов окном 3х3 и в 17 раз быстрее алгоритма суммирования по окну.

Заключение

В этой работе были рассмотрены типовые подходы к повышению производительности алгоритмов математической морфологии в задачах технического зрения, и измерены характеристики, достижимые при использовании этих подходов. Показано, что рассмотренные

подходы позволяют существенно (на несколько порядков) повысит производительность алгоритмов и сделать возможной обработку изображений в режиме реального времени на относительно маломощных вычислительных платформах.

Список литературы

[1] J. Serra. Image Analysis and Mathematical Morphology. Vol. I, Academic Press, London, 1982.

[2] Rein van den Boomgaard, Richard van Balen. Methods for fast morphological image transforms using bitmapped binary images. CVGIP: Graphical Models and Image Processing, 54(3), 1992, pp.252-258.

[3] B. J. H. Verwer, L. J. van Vilet. A contour processing method for fast binary neighborhood operations. Pattern Recognition Lett., 7, 1988, 27-36

[4] Thomas M. Breuel, U. Kaiserslautern. Efficient Binary and Run Length Morphology and its Application to Document Image Processing. 2007

[5] B.K. Lien. Efficient implementation of binary morphological image processing. Optical Engineering, 33(1) pp 3733-3738, 1994

[6] Dita A. I.-C., Otesteanu B. M., Quint C. F. (2011). Scanning industrial Data Matrix Codes. // 19th Telecommunications forum TELFOR 2011, Serbia, Belgrade, November 22-24, 2011 DOI: 10.1109/TELFOR.2011.6143768

[7] Joseph Gil, Ron Kimmel. Efficient Dilation, Erosion, Opening, and Closing Algorithms. // IEEE transactions on pattern analysis and machine intelligence, vol. 24, №12, 2002

[8] Рябых М.С., Сойникова Е.С., Батищев Д.С., Синюк В.Г., Михелев В.М. Высокопроизводительный метод анализа и морфологической обработки изображений // Научный результат. Информационные технологии. - Т.1, №3, 2016. DOI: 10.18413/2518-1092-2016-1-3-16-23

[9] Ryabykh M.S., Soynikova E.S., Batishchev D.S., Mikhelev V.M. Morphological Processing of Fingerprint Images with the Use of Parallel computing on GPUs // Trends in the Development of Science and Education: a Collection of Scientific Papers, based on the materials of the XII International Scientific-Practical Conference". Part 4. Samara: NITS «L-Zhurnal», 2016. 60 p.

[10] Domanski L., Vallotton P., Wang D. Parallel vHGW image morphology on CPUs using CUDA // CSIRO, Mathematical and Informational Sciences, Biotech Imaging.

[11] Dan S. Bloomberg Implementation Efficiency of Binary Morphology // International Symposium for Mathematical Morphology VI. Sydney, Australia, April 3-5, 2002.

[12] Гаврилов А.И., Тхет А. Применение методов сегментации изображений в задачах обнаружения дефектов поверхности сварных соединений // Вестник Московского государственного технического университета им. Н.Э. Баумана. Серия: Приборостроение. 2014. № 5 (98). С. 124-132.