Машинное зрение: от задачи до аппаратной реализации

А. В. Шокуров [1] [2]

Одно дело решить задачу математически и разработать соответствующий метод, а совсем другое переложить его на аппаратные компоненты. Даже на первый взгляд простые задачи требуют тщательного продумывания архитектуры программного обеспечения аппаратной реализации. Нужно понимать устройство оперативной и внутренней памяти, в том числе, аппаратных механизмов прямого копирования данных из различных подсистем, и функционирование отдельных подсистем (контроллеров) большой системы (система на чипе), а также, их взаимодействие. В частности, учитывать, что вычисления можно выполнять не только на процессоре, а, например, на каком-то контроллере. Так, есть тензорные вычисления, а есть ещё более специализированные по цифровой обработке изображений. Даже действенный вывод графической информации требует внимание от разработчика иначе могут возникнуть артефакты (искажение данных) при их визуализации. Для быстродействия методов важна не только математическая составляющая, но и аппаратная поддержка реализации отдельных частей алгоритма. Последнее находит отражение в современных архитектурах вычислительных систем и встроенных процессоров.

Ключевые слова: аппаратная реализация, процессор, оперативная память, контроллеры.

Допустим имеется метод решения некой задачи обладающий важными характеристиками. Замечу, что одно дело его теоретические возможности, а совсем другое показатели на реальном «железе» (компьютер/встроенная система). Последнее влечет важность разработки правильной аппаратной реализации теоретических методов. В данном тезисе делается упор на эффективном использование аппаратных ресурсов системы, в частности, на скорости работы метода.

[1] *Шокуров Антон Вячеславович* — к.ф.-м.н., научный сотрудник лаборатории вычислительных методов механико-математического факультета МГУ им. М.В.Ломоносова, e-mail: shokurov.anton.v@yandex.ru

Shokurov Anton Vyacheslavovich — Ph.D., research associate of Computational Methods Laboratory of Mechanics and Mathematics Faculty of Lomonosov Moscow State University

[2] Исследование выполнено при поддержке Междисциплинарной научно-образовательной школы Московского университета «Мозг, когнитивные системы, искусственный интеллект»

This research has been supported by the Interdisciplinary Scientific and Educational School of Moscow University «Brain, Cognitive Systems, Artificial Intelligence»

1. Система сжатие речи

Рассмотрим задачу сжатия речи человека. Будем исходить из того, что существует кодек (реализация метода сжатия) под рассматриваемый процессор (вычислителное ядро). Опишем «железные» детали, которые нужно учесть для успешной его работы в рамках системы. Успешность понимается в смысле эффективного использования ресурсов, в частности, высокой скорости работы кодека.

Наличие кода означает, что есть некая функция, которой можно передать массив аудио данных определенного размера. Функция возвращает закодированные аудио данные в неком другом массиве. Вызывая данную функцию для последовательных фрагментов аудио потока можно получить закодированные (сжатый) данные. Такой подход имеет простую реализацию на современных CISC процессорах (например, на архитектуре x86 от Intel/AMD) за счет переизбытка ресурсов.

В случае же относительно низкопроизводительных RISC процессоров типа DSP (digital signal processing processor, например, bf60x от Analog Devices или TMS320C6x от Texas Instruments) не все так же просто. Целью использования такого рода систем заключается в экономии денег: зачем платить больше, если можно сделать более дешевое решение, но за счет более продуманной программной архитектуры итоговой системы. Поэтому, естественно, хочется выполнить эффективную реализации итоговой системы на базе рассматриваемого кодека на оных DSP, которые обычно дешевле эквивалентных CISC систем.

Отключение кеша Вычислительное ядро процессора может выставить произвольный адрес (в общем случае, на внешней шине процессора) и тем самым работать с разными типами памяти/устройствами (чипами). Память может быть как внутренняя, так и внешняя, отличающиеся по скорость обработки данных, а точнее по времени необходимом для получения/записи значения ячейки. В момент ожидания данных происходит простой вычислительного ядра, т.е. прекращение выполнения инструкций, что, естественно, снижает его производительность, которая как раз таки существенно важна. Последнее, в частности, происходит при случайном доступе к внешней, например, оперативной памяти.

Для нивелирования данного эффекта обычно используется кэш, являющегося частью процессора. Кэш нужен для ускорения доступа к оперативной памяти, которая как минимум на порядок медленнее. Последнее достигается за счет запоминания значений наиболее часто встречаемых адресов данных. В итоге кэш хорошо подходит для каких-нибудь универсальных методов и, соответствующих, программ, в которых не придется задумываться зачем вообще кэш нужен и даже о самом факте

его существования. Последнее справедливо даже с точки зрения программиста, а не пользователя системы.

Операция поиска адреса была бы очень долгой, если бы кэш был устроен как список актуальных адресов и, соответствующих, значений. Поэтому поступают иначе: по адресу ячейки памяти строится индекс корзинки/набора, которая и содержит обозначенный массив (кэш линий) адресов. Количество элементов в корзинке фиксировано и называется ассоциативностью кеш памяти (обычно 4 или 8). Для эффективной реализации кеша адрес разбивается на разные поля [1]: связка (tag), индекс корзинки (set number) и смещение (offset). Как следствие для каждого используемого адреса достаточно сравнить 4/8 значения (кусок tag адреса), для чего имеется эффективная реализация в кристалле кеша.

Приведем пример на конкретных числах. Стандартный объем кеша около 32 Кб, что на несколько порядков меньше обычно доступной оперативной памяти (1 гигабайт). Кэш строка состоит обычно из 64 ($= 2^6$) байтов. Тогда количество корзинок равно $\frac{32KB}{64B} = 512$, для индексации которых потребуется $9(2^9 = 512)$ бит в поле адреса. В случае 1 гигабайта ($= 2^{30}$) оперативной памяти потребуется $30 - 9 - 6 = 15$ бит для связки (taga).

В каждую корзинку можно поместить (запомнить) не более 4/8 адреса с различающимися tagами, в противном случае происходит коллизия. В случае коллизии нужно будет замещать один адрес другим и, соответствующим, значением. Выбор вытесняемой из кэша строки обеспечивается алгоритмом замещения одних кэш строчек в корзинке, на другие. При обычном использовании вычислительной системы для «пользователя» описанные процессы «прозрачны», т.е. кеш живет своей жизнью.

Алгоритм замещения сложный и поэтому в общем случае в ходе данного процесса может произойти неожиданное (сложно прогнозируемое) вытеснение важного адреса из кэша и какой-то критически важный метод (обработка сигнала/события) сработает позже, чем необходимо. Последнее может привести к катастрофе (в буквальном смысле, если речь идет например об атомной станции или навигационной системе самолета). Так, из-за некорректной работы кэша могут потребоваться много холостых процессорных тактов пока не загрузятся необходимые данные из оперативной памяти. Если работа системы исходно была аккуратно посчитано с точностью до такта, то это будет означать, что актуальный алгоритм не сработает в запланированный момент, что может привести к цепной реакции задержек обработки событий и, в конечном итоге, к сбою системы.

Один из способа нивелирования данной проблемы это использования чрезмерно высокопроизводительного процессора. Но такой процессор видится дороже, по крайней мере дороже чем та система, которая может

быть построена на ручном управлении загрузкой и выгрузкой данных из внутренней памяти процессора.

Хорошо на некоторых системах кэш памяти можно отключить. Что и предлагается сделать в рассматриваемом случае.

Внутренняя память процессора При отключении кэша та область памяти, которая им использовалась под хранение временных строчек в корзинках, может быть использована как обычная память. Последнее означает, что по определенному адресному под-пространству процессора станет доступна внутренняя память процессора. Тип внутренней памяти обычно является SRAM (Static RAM), которую не следует путать с DRAM (Dynamic RAM). Внутренняя память работает быстро (на частоте процессора), гораздо быстрее чем оперативная память.

Существенно разное быстродействие типов памяти связано с их разным технологическим производством. Оперативная память (DRAM) состоит из конденсаторов, а внутренняя память процессор (SRAM) построена на базе транзисторов. В итоге внутренняя память работает на несколько порядков быстрее внешней, оперативной. Отмечу, что цена первой существенно дороже второй.

С памятью связано такое понятие как банк. Банк памяти подразумевает возможность независимой работы с разными частями области (чипа) памяти, что позволяет одновременно эффективно читать/писать в разные банки/области памяти. У внутренней памяти несколько банков, обычно как минимум два. При двух банков можно сделать так, чтобы разные процессы взаимодействовали с разными банками. Тогда конфликтов и, соответственно, ожидания освобождения шины при доступе к памяти не будет.

Речь идет о как минимум двух процессах: кодирование/сжатие аудио потока и загрузка данных во внутреннюю память (в нужный банк). Внутренняя память процессора связана отдельной шиной с оперативной памятью. Данные можно копировать процессором, что крайне неэффективно. Теряется весь смысл задуманного, т.е. создание эффективной системе на базе микропроцессора типа DSP. Такты процессора лучше на вычислительные методы использовать. К внутренней памяти процессора имеет доступ не только вычислительное ядро, но и специальный контроллер DMA, который может выполнить необходимое копирование.

DMA контроллер Данные во внутреннюю память процессора могу быть скопированы благодаря dma (direct memory access) контролеру процессора. Он выполняет операция копирования данных из «источника» в «пункт назначения» минуя вычислительное ядро процессора. Может взаимодействовать не только с внутренней/внешней памятью, но и другими

«устройствами» процессора. Например, взять данные с аудио устройства (например, с многоканального АЦП) или видео (видео аппаратный кодек) и записать во внешнюю/внутреннюю память.

При копировании данных в память DMA контроллером поддерживает разные режимы обхода по массиву. В частности, может заполнить матрицу построчно для растрового изображения. Может и многоканальные (мультиплексированные) аудио данные (например, аудио) автоматизированно записать по отдельным строчкам/массивам матрицы фиксированного размера. В последнем случае каждая строка матрицы будет соответствовать определенному аудио каналу многоканального потока.

В итоге, оперативная память нам и не понадобиться вовсе. Ведь DMA можно настроить так, чтобы он непосредственно брал данных с аудио устройства и копировал сразу внутрь самого процессора попутно разбивая на отдельные аудио каналы. Далее можно запустить процесс кодирования загруженных аудио данных. Последний будет работать оптимально раз аудио данные находятся во внутренней памяти процессора.

Замечу, что помимо ранее указанных недостатков в случае использования кэша, данные бы к тому же бы и лишний раз копировались. Сначала данные копировались бы в оперативную память, а потом автоматизировано загружались бы в кэш при кодировании. Далее закодированный аудио поток копировался бы в обратную сторону, а именно – из кэша в оперативную память, а потом в сеть или карту памяти. Очевидно лишнее копирование снижает эффективность системы (например, увеличивает задержку по кодированию аудио данных). Рассматриваемый в данном разделе подход сразу позволяет скопировать аудио данных во внутреннюю память процессора.

Двойная буферизация Как ранее было указано, эффективная работа не позволяет одновременно использовать один и тот же банк памяти, иначе операции будут друг на друга влиять, например, блокировать операции чтения/записи. Например, могут потеряться записываемые из АЦП аудио данные во внутреннюю память, если в тот же самый момент к этому же банку памяти обращается вычислительное ядро.

Для избежания подобных эффектов предлагается использовать принцип двойной буферизации, в данном случае, необходимо использовать два банка и работать с каждым из банков попеременно и в противофазу.

В случае кодирование аудио потока, пока данные закачиваются с аудио устройства в первый банк, вычислительное ядро тем временем должно работать со вторым банком. И наоборот, пока данные закачиваются во второй банк, вычислительное ядро тем временем работает с первым банком. И так далее. В данном случае, работа вычислительного ядра заключается в кодировании аудио данных. Понятно, что при

кодировании аудио данные будут актуальны так как ранее туда были скопированы.

Всё синхронизируется благодаря четко выбранной грануляции обработки фрагментов аудио данных. Например, в случае 1/8 секунды при 8000 Гц частоте дискретизации и 8 битном режиме будет 1000 байтов на буфер одного канала. Момент переключения на другой банк определяется обработчиком прерывания, который и переинициализирует dma канал (хотя можно включить автоматический режим). При переключении фактически меняются местами указатели на буфера в соответствующих банках. Один из указателей отвечает за то, что должно быть закодировано, а другой – куда скопировать новую порцию аудио данных.

SIMD инструкции DSP подразумевает наличие специальных инструкций позволяющих выполнять вычисления быстро. К таким инструкция, в частности, относятся SIMD (Single Instruction Multiple Data), которые позволяют выполнять векторные операции за такт. Например, поэлементно сложить два 4 элементных вектора из 8-битных чисел. Имеются важные операции и для векторного умножения и многих других важных математических действий. Кодек разработан на базе SIMD инструкций, поэтому все необходимые для кодирования/сжатия аудио потока вычисления делаются вычислительным ядром процессора.

При написании кода кодека не учитывалось устройство кэша, dma контроллера и подобных аппаратных сущностей. Требование к эффективному запуску кода заключается в быстром доступе к памяти, а именно – что при загрузки данных в регистры и, соответственно, выгрузки скорость максимальна. Для выполнения данного условия данные должны находиться во внутренней памяти процессора.

Как только многоканальные аудио данные скопированы DMA контроллером во внутреннюю память для каждого из каналов (буфера, массива данных) выполняется их обработка/кодирование аудио кодеком.

Итоговая система Одно дело написать метод под инструкции процессора, а совсем другое организовать процессы итоговой системы, которые его реализуют. Данную систему можно описать в виде циклограммы, в которой описано, что в какой момент происходит. Циклограмма пишется с точки зрения тактового генератора процессора.

Получается, что данные непосредственно с аудио устройства попадают во внутреннюю память процессора, одновременно с этим происходит кодирование данных, которые ранее были туда скопированы. Далее происходит переключение этих буферов, массивов данных.

Что можно сделать с теми данными, которые были закодированы? Можно по такому же принципу (применив DMA), например, отослать

их на устройство под названием «сетевая карточка» и транслировать по протоколу rtp. Последнее нужно сделать не дожидаясь смены буферов как только данные будут закодированы. Итого, пока данные копируются в дугой банк для данных текущего банка выполняется кодирование, а далее передача их в сеть. Потом ждем прерывания для смены буфера. В последнем случае ресурсов как таковых у такой системы не будет. Она вечная.

2. Специализированные вычислительные блоки

Развитие передовых кодеков идет своей жизнью, но в некоторых ситуациях главный принцип это совместимость, то что он неизменно используется на протяжении десятков лет. Последнее влечет наличие неких эталонных кодеков/методов, более того, они уже достаточно давно различными способами в железе реализуются. Возможно в этом есть какой-то ещё и экономический смысл, ведь чип с аппаратной поддержкой некого метода обычно гораздо дешевле, чем универсальный процессор с программной реализацией того же метода.

Для кодирования видео в рассмотренной ранее схеме можно заменить аудио на видео, аудио кодек на видео - и все будет работать. Но тенденция показывает, что дешевле делать методы в кристалле.

Высокопроизводительные вычисления можно достигать не только инструкциями, но и подсистемами процессора, его специализированными контроллерами. В железе реализуют сжатие изображений, видео и даже вычисления движения (оптический поток/смещения блоков mpeg). Это по сути готовые методы зашитые в процессор и для работы как раз обычно использующие внутреннюю память. Аппаратные реализации конечно имеют большое количество ограничений.

По аналогии с прошлым разделом рассмотрим заявленную идею на примере с кодеком сжатия jpeg.

Вводные Метод JPEG можно реализовать на языке Си или даже на ассемблере (например, с использованием инструкциях simd). Учитывая, настолько этот метод сжатия востребован можно сделать иначе – можно переложить реализацию метода в кристалл (контролер процессора).

При сжатии аудио потока данных требуется немного и поэтому можно, как ранее было описано, сразу их загружать во внутреннюю память процессора для кодирования. Но, если мы берем видео кадры, то они на порядок больше памяти занимают и сразу загружать их во внутреннюю память процессора не всегда эффективно (но можно за счет усложнения кода). Поэтому видео кадры будем сначала целиком загружать в оперативную память, а уже потом от туда копировать во внутреннюю.

Опишем внутреннее устройство оперативной памяти.

Оперативная память DRAM Когда данных много, в частности, при обработки видео кадров (одна картинка обычно большая и их часто много), без оперативной памяти сложно обойтись. В таком случае первоначально данные загружаются в оперативную память.

Необходимо правильно взаимодействовать с оперативной памятью. Оперативная память работает (относительно) быстро, если действовать по некому правильному «протоколу». Например, если мы будем случайно «прыгать» по адресам в памяти, то скорость будет очень медленной, а если идти по последовательным адресам, то скорость обработки будет фактически оптимальной.

Память DRAM организована как матрица некого фиксированного размера. С матрицей связаны такие понятия как индекс ряда и колонки. Каждый ряд является страницей памяти, наименьшим адресуемым внутренним кирпичиком чипа DRAM. При считывании произвольного байта, на самом деле внутренним чипом памяти грузиться вся страница, которая содержит этот байт, а уже потом из неё выбирается нужный байт (адресом колонки). Думаю понятно, что неэффективно строить запросы к памяти так, что для каждого байта грузиться своя страница памяти. Правильнее загрузив страницу обращаться к байтам этой же страницы как можно чаще. Так, если индекс ряда зафиксирован (постоянен), а меняется только адрес колонки, то данные берутся из уже загруженной страницы памяти.

Описанная схема работы чипа DRAM реализуется следующим образом. Физический адрес (идентифицирующий ячейку памяти) разбивается на какое-то количество битовых полей. В частности, есть битовые поля индекса ряда и колонки. Последнее битовое поле фактически отбрасывается/игнорируется ввиду того, что из оперативной памяти (планки) грузятся сразу несколько байтов, соответствующие кэш строке. Помимо данных полей в физическом адресе можно выделить битовое поле отвечающее за индекс/номер банка.

Банк обеспечивает независимую работу со страницами памяти, например, если 4 банка, то можно одновременно иметь подгруженными 4 страницы памяти и, соответственно, 4 независимых потока данных. Последнее позволяет на высокой скорости копировать из одной части памяти в другую или одновременно копировать что-то в память и что-то другое из памяти. Без банков страницы бы постоянно загружались и выгружались бы, что существенно бы снизило скорость.

Итоговая система В случае декодирования видео к памяти будут обращаться как минимум два процесса/контроллера. Один это аппарат-

ный распаковщик jpeg картинок, а другой видео контроллер (отрисовщик кадров). Память не может на такой высокой скорости обрабатывать запросы одновременно. Поэтому необходимо учесть нюансы при работе с банками памяти и гарантировать, что запросы не будут пересекаться. В противном случае, появится «снег» (случайные, обычно белые, точки на экране) – артефакт указывающий на то, что видео контроллер не смог считать значение пикселов из памяти (ввиду занятости шины), поэтому они заменены на случайные значения.

Загрузим видео данные (по одному кадру) непосредственно с камеры в очередной из банков оперативной памяти DRAM. Одновременно с этим копируем из другого банка во внутреннюю память процессора. Дальше по аналогии с обработкой аудио данных происходит кодирование загруженных данных, то есть вызываем некую функцию или программируем некие регистр специализированного контроллера процессора указав адрес оного буфера со сжатой картинкой. После распакованную картинку нужно передать (через DMA) видео контроллеру. Дальше мы повторяем эти же действия после обмена указателей. При наличии оперативной памяти закодированные данные предварительно можно перед дальнейшей обработкой скопировать обратно в неё, а потом, например, записать их на flash-память, т.е. возможны разные уже варианты в зависимости от цели системы.

Обычно для отмеченных ранее процессов составляют циклограмму. Можно заметить, что ввиду того, что ресурсы фактически не используется, система также будет работать бесконечно так в такт.

3. Специализированные инструкции

Тезис начался с того, что есть некий кодек написанный с использованием SIMD. Далее был приведен случай, когда некий кодек реализован в самом процессоре.

В согласии с ранее написанным, процесс удешевления итоговых вычислительных систем подразумевает перенос важных элементов/кирпичиков методов в железо. В данном разделе будет показано, что действительно есть некий промежуточный вариант, есть некие кирпичики.

Специальный тип данных Как частный случай к этому относится создание специальных типов данных. Так, для «обычных» численных вычислений необходимо уметь работать с числами с плавающей точкой. Традиционно именно на вычисления с плавающей точкой делался упор во всяких ускорителях (например, Tesla от nVidia). Но учитывая последние тенденции в области искусственного интеллекта появилась аппарат-

ная поддержка новых типов данных [4]: как 8/4 битные целые числа, так и 16 битные плавающие точка (в которой поля под мантиссу и порядок выбраны особым образом).

Актуальна задача портирования искусственных нейронных сетей на низкопроизводительное железо. Последнее достигается, например, квантованием – использование меньшего количества битов для представления значений весов[6] в нейронных сетях. Благодаря чему, нейронные сети можно строить с использованием целых чисел, а не чисел с плавающей точкой. Последнее позволяет существенно повысить как скорость их обучения, так и предсказания. Учитывая относительную простоту устройства отмеченных вычислительных блоков их можно сделать в достаточно большом количестве в кристалле (процессоре).

Растеризация Наверное, одно из первых значимых примеров аппаратного ускорения связано с отображением трехмерной графики. Традиционный способ отрисовки трехмерных объектов заключается в растеризации (попиксельный проход точек) отдельных треугольников его составляющих. Следовательно необходимо уметь выполнять быструю растеризацию треугольников с учетом перспективной проекции. Алгоритм достаточно прост и легко поддается распараллеливанию и реализации с использованием SIMD [8].

Казалось бы есть алгоритм растеризации треугольников, который можно и распараллелить. Почему бы не ограничится версиями реализации для GPU (graphics processing unit) [9]? Тем не менее даже в самых современных графических карточках есть отдельный кусок кристалла, который отвечает за растеризацию. Потому что считается, что данная специализация задачи важна и не надо все делать универсальным и переносить на gpu. Традиционно последнее позволило существенно увеличить скорость отрисовки кадров трехмерной сцены в сравнении с процессорной реализацией.

Отмечу, что именно трехмерная графика явилась предтечею современных многоядерных графических карточек с поддержкой вычислений общего назначения. Первоначально были реализованы пиксельные и вершинные шейдеры, позволяющие запустить очень короткий кусок кода (порядка 3 инструкций) для каждого, соответственно, пикселя экранного изображения и вершины полигона для достижения определенных визуальных эффектов. Потом, в результате применения этих сущностей не по назначению удалось одно из важнейших универсальных вычислений перенести на них, а именно – Фурье преобразование [7].

Отрисовка отдельных линий является ещё одним специфическим алгоритмом, кирпичиком. Известный алгоритм Брезенхема [5] использует только целые чисела, что позволяет ему иметь хорошие оценки по скоро-

сти работы. Тем не менее, отрисовка линий, тем более, с учетом антиалайзинга хороший кандидат на аппаратную поддержку. Такую поддержку имеют видео карточки предназначенные для дизайнеров (например, quadro от компании nVidia). В обычных карточках, которые покупаются, в частности, для игр, не имеется соответствующей аппаратной поддержки.

Трассировка лучей В современных графических карточках добавлена поддержка и для ray tracing (трассировка лучей) являющимся другим, более точным подходом отрисовки трехмерной графики. Аппаратно ускорено вычисление в какой именно треугольник попадает заданный луч.

Если рассмотреть данный алгоритм, то поймем, что он плохо параллелится ввиду того, что появляются неизбежные условные конструкции [10].

Суть[11] аппаратной реализации алгоритма основана на вложенных коробках (bounding box). Если луч пересекает коробку, то надо рекурсивно обработать его составляющие (внутренние) коробки, а если луч проходит мимо, то эту коробку прекращаем обрабатывать, Листовая коробка будет содержать треугольник. Данный алгоритм аппаратно реализован на графической карточке по причине оптимизации железа, в частности, её цены (не рационально использовать другие ресурсы для реализации трассировки лучей).

Цифровая обработка изображений Если возвращаться к специализированным процессорам (ADSP-BF60x[2] компании Analog Devices и TDA2Px[3] компании Texas Instruments), то там уже достаточно давно (с 2010 годов), появились вычислительные блоки или аппаратная поддержка для традиционных методов цифровой обработки изображений. Например блок, который позволяют выполнить свёртку с произвольным ядром 5 на 5, вычисление угла и величины градиента по производным вдоль двух направлений, вычисление гистограммы и так далее. Все вычисления делаются отдельным контроллером, а не основным вычислительным ядром процессора.

Например, у BF60x на аппаратном уровне строится вычислительный граф из подобных операций. Имеет ограничения, например, фиксированное количество узлов и порядок их следования. Например блок, который по значениям градиента вычисляет дискретный угол (16 направлений) и магнитуду. Так, на вход подаются изображение градиентов (по x и y направлениям), на выходе выдается изображение соответствующих углов. Аналогично вычисление интегрального изображения, которые важны при вычислении суммы значений прямоугольного фрагмента изоб-

ражения. Они тоже могут быть вычислены на аппаратном уровне. Итого, соединив блоки соответствующим образом можно построить и гистограмму градиентов.

Морфологические операторы Из традиционных методов цифровой обработки изображений есть еще тема морфологических операторов. Позволяет решать ряд важных задач по цифровой обработке изображений: убрать шум соль-перец или подсчитать количество дырок/зерен. Морфологические операторы имеют простую аппаратную реализацию, например, через LUT (look up table). Их аналог основанный на чисто программном варианте был бы гораздо сложнее.

Метод поиска связанных компонент на бинарном изображение каждую из связанных компонент заполнения её порядковым номером. Данный алгоритм имеет аппаратную реализацию, но с ограничением (например, 10 или 20) на количество объектов. Количество предполагаемых объектов на изображении фиксируется в момент создания чипа.

Подсчет объектов в бинарном изображении достигается за счет метода поиска связанных компонент. Но, как было пояснено выше аппаратная реализация не может найти динамическое количество связанных компонентов на картинке. Если у нас картинка с шумом соль-перец, то количество связанных компонент будет явно больше пары десятков. Поэтому аппаратная реализация казалось бы будет неприменима. Более того, даже программная реализация, заключающиеся в запоминании координат всех связанных областей и их площадей и поиска среди них только самых больших, потребовало бы большие ресурсы (в частности, памяти).

Морфологические операторы являются грамотным подходом к устранению шума. Воспользуемся морфологическим оператором, который уберет все эти шумовые точки. Его суть заключается в том, что берется двигающиеся окошко 3 на 3 по изображению и, если в нем ненулевой является только точка в середине, то она и удаляется/зануляется. Для данного оператора тоже имеется аппаратная реализация.

После чистки изображения от шума можно запустить аппаратную реализованный алгоритм поиска связано компонент и тем самым решить первоначально поставленную задачу.

С точки зрения морфологических операторов есть ещё интересный пример. Рассмотренные ранее морфологические операторы имели размер 3 на 3, но есть более сложные преобразования, которые вычисляют скелетонизацию (средняя линия внутренней области) связанной компоненты. В данном случае нужно будет либо применить блоки размера 5 на 5, либо пройтись по изображению два раза боком размера 3 на 3 (по-

требуется условная операция). Последнее тоже возможно реализовать в железе.

Постепенный перенос в кристалле Как показывают приведенные примеры есть тенденция постепенного переноса в кристалл устоявшихся методов. На самом деле такое дело давно наблюдается. Так, в одном из реализаций x86 процессора, была инструкция по вычислению матричного произведения.

4. Поиск фитофторы

Продолжая ход мысли предыдущих разделов можно пойти дальше реализации всего метода в кристалле процессора и упростить решение за счет аналоговой части целевой системы. Последнее означает, что некоторые простые алгоритмы можно решить за счет физической предобработки сигнала, например, аналоговым фильтром.

Рассмотрим прикладную задачу из темы искусственного интеллекта. Фитофтора является заболеванием паслёновых культур проявляющиеся на листьях. Задача заключается в выявлении данного заболевания у растений по аэрофотосъемке. Причем, с точки зрения бизнеса необходимо отловить момент заражения, а не когда уже и невооруженным глазом человека будет явно заметны проявления болезни (характерные темные пятна).

Выбранный подход решения задачи Видимому (человеку) свету соответствует диапазон длин волн 300нм-700нм, что в свою очередь соответствует спектру солнца. Отражаясь спектральная кривая принимает изменения характерные поверхности и излучателя, т.е. для каждой длины волны можно измерить излучаемую энергию объекта (за вычетом источника освещения). Последнее влечет, что объекты можно попробовать отличать по их спектральным кривым. Эти кривые могут быть использованы не только для классификации самих объектов, но и определения тех или иных характеристик. Например, сорт, урожайность, болезни сельскохозяйственных культур.

Измерения спектра в точке осуществляется спектрометром. Для получения же изображения каждый пиксель которого это спектр применяют гиперспектральную камеру. При гиперспектральной съемке формируются два изображения: непосредственно гиперспектральное и панхроматическое. Разрешение первого изображения обычно небольшое, например, 64 на 64 пикселя, но зато каждый пикселе есть спектр из, например, 100 длин волн. Такое небольшое изображение не позволяет в полной мере

распознавать объекты в традиционном смысле. Для таких целей камерой и создается одновременно с гиперспектральным панхроматическое изображение. Оно имеет размер, например, 1000 на 1000 пикселов, но значение каждого из которых по сути является суммарной энергией всего рассматриваемого спектрального диапазона.

Нашей целью будет найти спектральный поддиапазон, который выявить наличие фитофторы за счет пороговой функции, которая лежит в основе методов распознавания. Даже в нейронных сетях она используется, называясь функцией активации. Отмечу, что нейронные сети не понадобятся для решения поставленной задачи.

Важность сглаживания изображений Рассмотрим изображение состоящее из двух объектов (для определенности, кругов), каждый из которых состоит из разной плотности облаков пикселов. Тогда, при одинаковой интенсивности значений пикселов пороговой функцией не получится эти два объекта отделить друг от друга.

Решение данной задачи заключается в сглаживание изображения неким цифровым фильтром, например, выполнив усреднение блоком размера 5 на 5. Действительно, плотность расположения пикселов у объектов разная, поэтому, после сглаживания результирующая интенсивность пикселов этих объектов станет разной. Следовательно, после сглаживания можно применить пороговую функцию, где значение порога выбирается на основе гистограммы интенсивностей пикселов.

Панхроматическое изображение фактически является полутоновым. Разницы в интенсивности между пикселями здорового и больного растения в ходе исследования не выявлено. Но было замечено, что применив сглаживание пороговая функция позволит выделить фрагменты изображения с зараженными растениями. Раз панхроматическое изображение это сумма всего спектра, то может есть поддиапазон спектра, который даст более точный результат? Суть ранее обозначенного простого метода заключается в поиске именно этого значимого поддиапазона спектра. Поддиапазон будет искаться на спектрах гиперспектрального изображения.

Поиск спектрального поддиапазона Сформируем два класса: спектры зараженных растений и здоровых. Необходимо найти спектральный поддиапазон, который позволит лучше всего отделить эти два класса.

Для произвольного спектрального диапазона оптимальный порог ищется за линейное время. Действительно, при выборе спектрального диапазона набор спектров превращается в набор чисел. Для классифи-

кации набора чисел применяется как раз-таки пороговая функция, где оптимальное значение порога ищется по гистограмме.

Поиск же интересующего нас значимого спектрального поддиапазона осуществляется полным перебором. Так, перебираются все возможные поддиапазоны спектра и для каждого из них осуществляется описанная выше процедура поиска оптимального порога и, соответственно, оценка качества классификации. Повторюсь, тут нет нейронных сетей, деревьев и подобных методов машинного обучения.

Для рассматриваемых в рамках этой задаче конкретных данных найденным значимым спектральным диапазоном является диапазон длин волн от 760 до 770 нанометров. Именно он позволяет выявить растения с фитофторой.

Итоговая система Выполнив исследование по обнаружению растений зараженных фитофторой по гиперспектральным изображениям был найден значимый спектральный поддиапазон. Если вопрос цены не стоит, то видится следующая итоговая система Для съемок применить гиперспектральную камеру (они стоят дорого), а для подсчета интегральной суммы поддиапазона спектров использовать высокопроизводительный компьютер, который скорее всего и дорогой. В итого получится дорогое решение.

Построение дешевой системы сводится к поиску «простого» решения по получению самих значений энергий длин волн из значимого спектрального поддиапазона и их суммирование. Простое аналоговое решение заключается в применении светофильтров. Можно изготовить датчик со спектральными характеристиками соответствующими требуемым спектральным диапазонам. По сути нужно взять «стандартный» датчик и покрыть его необходимыми светофильтрами. Тогда такой датчик будет воспринимать свет только в данных спектральных поддиапазонах. При этом, считанное значение с датчика и будет искомой суммой энергии рассматриваемого спектрального диапазона. В итоге получаем простое решение, которое не требует дорогой камеры и вычислительной системы.

В случае единственного спектрального поддиапазона прикреплять светофильтр (из-за одного фильтра) к датчику наверное не стоит. Можно прикрепить светофильтр к объективу монохромной камеры, что ещё более удешевит итоговую систему.

5. Компьютерная графика

В данном разделе будет описаны некоторые методы компьютерной графики в разрезе тех идей, которые были до этого изложены. В основе

большинства представленных в данном разделе методов лежит очень сильная, но в тоже время, простая инженерная мысль.

Отрисовка графических объектов Для вывода текста на экран его можно было бы растеризовать в некий (видео) буфер и уже его выводить целиком. При растеризации отрисовывается каждый пиксель по отдельности. Такой подход крайне расточителен как с точки зрения памяти, так и вычислительных ресурсов. Так, фактически многократно выполняются повторные действия при отрисовки одних и тех же букв. С другой стороны, растры отрисованных букв повторяются, что фактически показывает и неэффективное использование памяти. Последнее важно в низкопроизводительных системах или древних/старых компьютерах.

Существенное уменьшение ресурсов достигается за счет хранения выводимого текста в матрице. При выводе на экран графическое ядро вычисляет элемент матрицы накрывающий экранный элемент. Элементу матрицы соответствует некий раст буквы, который хранится отдельно от выводимой информации на экран. Зная элемент матрицы можно экранный элемент сопоставить пикселю растра буквы. Вычисленное значение пикселя и выводится на экране в рассматриваемом элементе. Такой подход и память потребляет существенно меньше, да и растры букв получается не надо каждый раз отрисовывать (например, через кривые Безье) заново.

Аналогично выводятся на экран спрайты – небольшие по размеру растры. Разница, во-первых, заключается в возможности вывода спрайта в произвольной точке экрана, а не в узлах фиксированной сетки как при при выводе текста. Во-вторых, спрайты цветные, а не черное белые буквы. Вывод спрайтов на экран производится поверх основного видео буфера. Подчеркну, вывод срайтов на экран осуществляется на аппаратном уровне, без лишнего копирования данных. Так, при проходе экранных элементов графическим контролером происходит проверка того в какие спрайты рассматриваемый элемент попадает. На экран выводится значение цвета пикселя того спрайта в которого попали, иначе пиксель видео буфера. Спрайт отрисовывается без копирования, а в процессе показа на экране.

Отображение окон Пусть нужно перемещать по изображению высокого разрешения некое окно, которое отображает подобласть/кроп. В современном подходе нужно фактически копировать это кроп (область памяти) в видео кадре на что уходит немалые ресурсы системы. На самом деле все просто при правильном аппаратном решении. Так, достаточно переместить указатель на начало новой отображаемой области. Графическая система будет выводить пикселы экрана относительно нового ука-

зателя. Таким образом при перемещении окна перемещается всего-лишь указатель.

На процессорах с поддержкой аппаратного вывода окошек копировать ничего не потребуется. Подобные технологии есть в DSP системах, в частности, в сотовых (смарт) телефонах и мультимедиа проигрывателях. В них графическая система более эффективна, чем на обычном компьютере. Например, благодаря тем возможностям, которые есть у графического контроллера таких процессоров, можно поверх какого-то окошка (например, поверх видео) отобразить окошко с текстом, в частности, титры.

Нужно будет сделать необходимые аппаратные окошки, которых конечно же поддерживается фиксированное количество, например, не больше 4. Системе нужно будет указать, что первое окошко будет во весь экран в котором показывается видео. Видео (набор jpeg) декодируется в некий буфер в родном для видео цветовом пространстве. Разумеется изображение не нужно переводить в формат rgb (цветной), оно остается в родном формате, например, 410yuv (яркость и цветность с пространственным прореживанием). Далее поверх данного создается окно в текстовом режиме. Для этого указывается координаты лево-верхнего угла и размер, а также и прозрачность. Под словами создается подразумевается, что программируются соответствующие регистры графической подсистемы. Для показа титров достаточно будет обновить текст в текстовом поле (в неком участке памяти). При выводе на экран графическое ядро корректно совместит описанные окна, в частности, титры будут отображаться поверх видео на лету без копирования. Никакого рендеринга в отдельный участок памяти не происходит. Последнее означает, что такой системой меньше ресурсов тратятся.

На обычном компьютере титры придется постоянно копировать вручную/либо графической системой. В итоге тратятся ресурс процессора. В универсальных системах есть что-то близкое в виде поддержки единственного окна через overlay, но по функционалу оно далеко от выше написанной системы во встраиваемых системах.

Современная ресурсоемкая архитектура В современных системах, где есть некий общий буфер, необходимо копировать в него данные для вывода информации на экран. Так есть некий участок памяти, который обрабатывает графическая карточка, точнее выводит его на монитор. Для вывода информации необходимо записать туда информацию, что подразумевает израсходование дополнительных ресурсов системы.

Для некоторых задач компьютерной графики на самом деле есть какие-то простые алгоритмы, но они либо забываются, либо отбрасываются в пользу универсальности. Как мне кажется это одна из при-

чин повышения производительности процессора. Встроенные системы, например, сотовые телефоны видятся более продуманными.

6. Итого

Для решения прикладных задач необязательно стремиться к последним тенденция и использовать мощные методы. Иной раз достаточно применить и традиционный подход. Но даже выбрав правильное «простое» решение нужно суметь его эффективно реализовать в железе. В противном случае, система будет работать на порядок, а то и порядки медленнее.

Список литературы

[1] Ulrich Drepper, "What Every Programmer Should Know About Memory", *Red Hat, Inc.*, 2017.

[2] "Data Sheet Final - ADSP-BF606/ADSP-BF607/ADSP-BF608/ADSP-BF609 Blackfin Dual Core Embedded Processor Data Sheet, Revision A", *Analog Devices, Inc.*, 2014.

[3] "TDA2Px SoC for Advanced Driver Assistance Systems (ADAS)", *Texas Instruments*, 2020.

[4] "NVIDIA Ampere GA102 GPU Architecture", *NVIDIA*, 2021.

[5] Bresenham, J. E., "Algorithm for computer control of a digital plotter", *IBM Systems Journal*, 1965, №4(1), 25-30.

[6] Matthieu Courbariaux, Jean-Pierre David, Yoshua Bengio, "Training Deep Neural Networks with Low Precision Multiplications", *ICLR*, 2015 (Workshop).

[7] Kenneth Moreland, Edward Angel, "The FFT on a GPU", *Conference: Proceedings of the 2003 ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2003.

[8] V. F. Frolov, V. A. Galaktionov, B. H. Barladyan, "Comparative study of high performance software rasterization techniques", *Mathematica Montisnigri*, 2020.

[9] S. Laine, T. Karras, "High-performance software rasterization on gpus", *ACM*, 2011.

[10] Timo Aila, Samuli Laine, "Understanding the Efficiency of Ray Traversal on GPUs", *Conference: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics*, 2009.

[11] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, Philipp Slusallek, "Realtime Ray Tracing on GPU with BVH-based Packet Traversal", *IEEE/Eurographics Symposium on Interactive Ray Tracing*, 2007.

Embedded vision: from objective to hardware implementation
Shokurov A. V.

It's one thing to solve a problem mathematically and develop an appropriate method, but it's quite another to port it to hardware components. Even seemingly simple tasks require careful thought software architecture hardware implementation. It is necessary to understand the device of RAM and internal memory, including hardware mechanisms for direct copying of data from various subsystems, and the functioning of individual subsystems (controllers) of a large system (system on a chip), as well as their interaction. It is necessary to understand the structure of RAM and internal memory, including hardware mechanisms of direct data copying from various subsystems, and functioning of individual subsystems (controllers) of a large system (system on a chip), as well as their interaction. In particular, take into account that calculations can be performed not only on the processor, but, for example, on some controller. So, there are tensor calculations, and there are even more specialized digital image processing based. Even the effective output of graphic information requires attention from the developer, otherwise artifacts (data distortion) may occur during their visualization. For the performance of methods, not only is the mathematical factor important, but also the hardware implementation support of seperate algorithm elements. The latter is reflected in modern computing systems and embedded processors architectures.

Keywords: hardware implementation, processor, random access memory, controller.

References

[1] Ulrich Drepper, "What Every Programmer Should Know About Memory", *Red Hat, Inc.*, 2017.

[2] "Data Sheet Final - ADSP-BF606/ADSP-BF607/ADSP-BF608/ADSP-BF609 Blackfin Dual Core Embedded Processor Data Sheet, Revision A", *Analog Devices, Inc.*, 2014.

[3] "TDA2Px SoC for Advanced Driver Assistance Systems (ADAS)", *Texas Instruments*, 2020.

[4] "NVIDIA Ampere GA102 GPU Architecture", *NVIDIA*, 2021.

[5] Bresenham, J. E., "Algorithm for computer control of a digital plotter", *IBM Systems Journal*, 1965, №4(1), 25-30.

[6] Matthieu Courbariaux, Jean-Pierre David, Yoshua Bengio, "Training Deep Neural Networks with Low Precision Multiplications", *ICLR*, 2015 (Workshop).

[7] Kenneth Moreland, Edward Angel, "The FFT on a GPU", *Conference: Proceedings of the 2003 ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2003.

[8] V. F. Frolov, V. A. Galaktionov, B. H. Barladyan, "Comparative study of high performance software rasterization techniques", *Mathematica Montisnigri*, 2020.

[9] S. Laine, T. Karras, "High-performance software rasterization on gpus", *ACM*, 2011.

[10] Timo Aila, Samuli Laine, "Understanding the Efficiency of Ray Traversal on GPUs", *Conference: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on High Performance Graphics*, 2009.

[11] Johannes Gunther, Stefan Popov, Hans-Peter Seidel, Philipp Slusallek, "Realtime Ray Tracing on GPU with BVH-based Packet Traversal", *IEEE/Eurographics Symposium on Interactive Ray Tracing*, 2007.