Реализация алгоритма Виолы и Джонса на OpenCL

С. А. Гладилин, А. С. Григорьев, А. А. Котов, Д. П. Николаев

Аннотация. В работе обсуждается вопрос SIMD-конформности алгоритма Виолы—Джонса и исследуются возможности его распараллеливания на SIMD-архитектурах, в том числе специализированных графических процессорах (GPU). Разработана SIMD-конформная реализация алгоритма вычисления признаков Хаара в алгоритме Виолы—Джонса. Предложено несколько вариантов GPU-реализации алгоритма Виолы—Джонса, проведены численные эксперименты, позволяющие сравнить данные варианты и последовательную реализацию алгоритма Виолы—Джонса на одном и том же физическом устройстве (APU), совмещающем в себе центральный и графический процессоры.

Ключевые слова: *алгоритм Виолы—Джонса, обнаружение объекта на изображении, зрительный интеллект, SIMD, OpenCL, GPU.*

Введение

Задача *детекции* — обнаружения объектов (например, лиц) на изображении является актуальной для многих приложений, таких, например, как системы безопасности или роботостроение. Решение данной задачи требует значительных вычислительных ресурсов, особенно когда поиск осуществляется на изображениях большого размера. Особенно важна оптимизация алгоритмов детекции для систем реального времени.

В настоящее время одним из возможных средств оптимизации является использование «мелкозернистого» параллелизма, представленного технологией SIMD [1], которая реализована как на центральном процессоре (CPU), так и на специализированном графическом процессоре (GPU). Графические процессоры являются частью всех видеокарт, а достаточно мощные видеокарты содержат GPU, которые выигрывают в производительности у CPU. В связи с этим актуален подход GPGPU (General Purposes GPU) [2], который заключается в использовании графического процессора не только для рендеринга графики, но и для других параллельных вычислений. Использование для вычислений GPU по сравнению с центральным процессором выгодно в силу его высокого параллелизма.

Компания NVIDIA в 2007 году предложила платформу параллельных вычислений CUDA (Compute Unified Device Architecture). Данная модель программирования дает разработчикам доступ к виртуальному набору команд и памяти параллельных вычислительных элементов в графическом процессоре. Следует отметить, что разработчики успешно используют представленную технологию в задачах обнаружения объектов и распознавания образов [3], [4] и [5]. Недостаток данной архитектуры в том, что она не работает на GPU других производителей.

Несколько лет спустя свет увидел OpenCL (Open Computing Language) [6] — новый фреймворк для разработки параллельных программ, которые могут исполняться на гетерогенных платформах, состоящих из центральных, графических и других процессорных устройств. Он позволяет использовать один и тот же программный код для различных архитектур: графических процессоров NVIDIA и ATI, центральных процессоров Intel, AMD и ARM, а также процессоров Cell.

В настоящее время ряд производителей, в том числе Intel, AMD и ARM, создают гибридные вычислительные устройства APU (Accelerated processing unit), совмещающие CPU и GPU на одном кристалле, что увеличивает скорость передачи данных между ними и снижает энергопотребление. В настоящей статье исследуются возможности одного из таких гибридных устройств — AMD G-T56N Processor [7], который совмещает в себе центральный процессор с графическим AMD Radeon HD 6320 Graphics.

1. Алгоритм Виолы и Джонса с вычислительной точки зрения

Алгоритм Виолы—Джонса [8], изначально разработанный для решения задачи поиска лиц на изображениях, хорошо зарекомендовал себя для быст-

рого поиска и распознавания любых условно плоских объектов, обладающих характерным распределением интенсивностей и имеющих при этом незначительное перспективное искажение, как на статических изображениях, так и в видеопотоке [9, 10]. Классическая версия этого алгоритма использует машину статистического обучения AdaBoost, для которой впоследствии было предложено несколько модификаций. В настоящей работе рассматривалась версия алгоритма, основанная на Real AdaBoost [11].

Для решения задачи детекции алгоритм Виолы—Джонса осуществляет перебор возможных положений детектируемого объекта и в каждом из положений решает задачу *классификации* — определения, есть в данном месте изображения искомый объект или нет.

Детектором объекта в алгоритме Виолы—Джонса является *каскад* классификаторов. Каскад состоит из нескольких *сильных классификаторов,* каждый из которых возвращает одно из двух значений: -1 (объекта нет) или 1 (объект есть). Условием обнаружения объекта является положительный ответ всех сильных классификаторов каскада.

Сильный классификатор в свою очередь состоит из нескольких *слабых.* Каждый слабый классификатор также возвращает -1 или 1. Для вычисления ответа сильного классификатора линейная комбинация ответов слабых сравнивается с порогом, при его превышении возвращается 1, иначе -1.

Первичным распознаваемым признаком является наличие и величина контраста между характерными участками изображения (такие признаки называют Хаар-подобными). Слабый классификатор содержит набор прямоугольников, помеченных весами -1 и 1. По этим прямоугольникам проверяется признак Хаара: выполняется проекция прямоугольников на участок входного изображения, вычисляется интеграл яркости входного изображения по каждому из прямоугольников, а полученные интегралы суммируются с учетом веса прямоугольника (1 или -1). Для нормировки результат делится на простую сумму интегралов (без учета веса).

Слабый классификатор также содержит массив *бинов* — заранее заданных чисел, являющихся возможными ответами слабого классификатора. Вычисленное значение признака Хаара масштабируется к отрезку $[0, n-1]$, где n — длина вектора бинов, округляется вниз и используется как индекс в массиве бинов.

При поиске объекта на изображении распознающий каскад независимо применяется к пикселям изображения для проверки наличия объекта в окрестностях каждого пикселя. Это приводит к вычислению всех слабых классификаторов внутри первого сильного, после чего, если сильный ответил отрицательно, вычисление каскада немедленно прекращается, а если положительно — происходит вычисление следующего сильного классификатора. Если все сильные классификаторы ответили положительно, объект считается обнаруженным в данном положении.

Поскольку объект имеет значительный (по сравнению с пикселем) размер, а детектор Виолы—Джонса не слишком чувствителен к нарушению выравнивания каскада относительно искомого объекта (рис. 1), алгоритм можно ускорить, не сильно потеряв в качестве. Для ускорения каскады применяются не к каждой точке изображения, а разреженно с некоторым шагом, в несколько раз меньшим размера искомого объекта.

Наиболее ресурсоемкой операцией при вычислении детектора Виолы—Джонса является вычисление признаков Хаара, т. е. разностей интегралов изображений по парам (обычно смежных) прямоугольников. Вычисление интеграла производится с использованием интегрального изображения, что для изолированного прямоугольника требует 4 доступа к изображению в памяти, 1 сложение и 2 вычитания. Для смежных

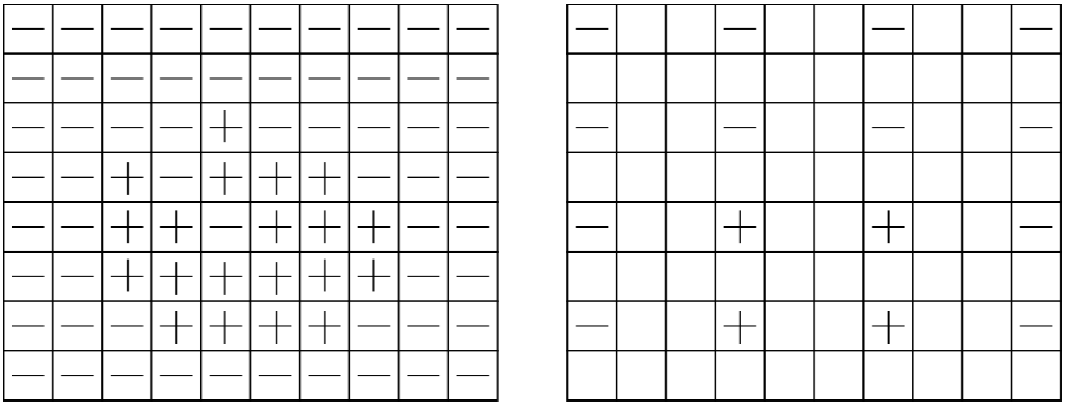

Рис. 1. Вычисление распознающего каскада на плотной (слева) и разреженной (справа) сетках. «+» означает положительное срабатывание, «−» отрицательное, пустая клетка означает, что каскад в данной точке не вычислялся. Шаг разрежения (dx, dy) = (3, 2)

прямоугольников, имеющих общие вершины, доступ к элементу в памяти требуется единственный раз, независимо от того в сколько прямоугольников эта вершина входит.

Наиболее затратными для современных систем здесь являются операции доступа к памяти. Их можно ускорить за счет эффективного использования кэша данных процессора, что требует локальности и предсказуемости обращений к памяти. Последовательность обработки данных в памяти также важна при реализации алгоритма на SIMD-архитектурах, поэтому можно назвать свойство плотности и последовательности обращений алгоритма к памяти *SIMD-конформностью.*

Большинство современных вычислительных устройств обладают SIMD-функциональностью в той или иной степени (все GPGPU и APU, расширения SSE, AVX, Neon на CPU), поэтому актуальна разработка SIMD-конформной реализации алгоритма вычисления признаков Хаара.

Набор смещений на интегральном изображении, соответствующих вершинам прямоугольников Хаара, назовем *шаблоном,* а вершины в нем — *узлами.* Алгоритм Виолы—Джонса, в котором каскады вычисляются на каждой точке изображения, обладает SIMD-конформностью при вычислении первого сильного классификатора, так как каждый узел шаблона последовательно проходит все пиксели на изображении. Однако в случае вычисления признаков Хаара на разреженном изображении SIMD-конформность нарушается.

2. SIMD-конформный алгоритм вычисления признаков Хаара на разреженных точках изображения

Рассмотрим пример, когда шаблон представляет собой 2 ряда по 3 узла, образующих прямоугольники 4 × 3 пикселя (рис. 2, узлы шаблона отмечены черным цветом). Будем вычислять этот шаблон не в каждой точке интегрального изображения, а с шагом разрежения $dx = 3$) (на рисунке положение узлов шаблона на следующем шаге обозначено серым цветом).

Как видно, в процессе вычисления в кэш процессора попадают пиксели интегрального изображения, значения которых не требуются ни на данном ни на последующих в ближней перспективе (на рисунке такие «лишние» пиксели обозначены белым цветом). Это приводит к неэффективному использованию кэша и потере производительности.

Если бы все смещения узлов шаблона имели одинаковый остаток от деления на шаг разрежения (вдоль соответствующих координат), можно было бы сформировать подвыборку интегрального изображения, т. е. уменьшенное изображение, из которого удалены

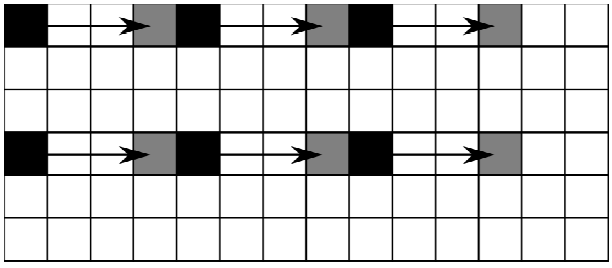

Рис. 2. Паттерн обращений к памяти при переходе к следующей точке интегрального изображения при использовании классической схемы на разреженной сетке. Шаг разрежения (dx, dy) = (3, 2). Белым обозначены точки интегрального изображения, которые не требуются для вычислений, но могут попасть в кэш процессора, что снижает эффективность вычислений

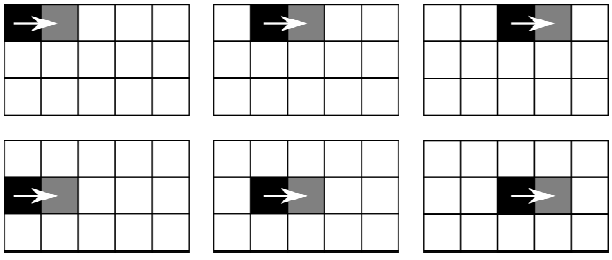

Рис. 3. Паттерн обращений к памяти при переходе к следующей точке изображения при использовании предлагаемой схемы на разреженной сетке. Шаг разрежения (dx, dy) = (3, 2)

все пиксели, кроме имеющих заданный остаток от деления на шаг разрежения. На такой подвыборке алгоритм обладал бы SIMD-конформностью.

Для восстановления SIMD-конформности в общем случае, когда смещения узлов шаблона имеют различные остатки от деления на шаг разрежения, разобьем все смещения и элементы интегрального изображения на группы с одинаковым остатком от деления на шаг разрежения (по каждой из осей). Таким образом, если шаг разрежения по оси x равен dx, по y, соответственно, dy, то получается dx * dy групп.

Для каждой группы сформируем отдельную подвыборку интегрального изображения с соответствующей «фазой» (остатком от деления). Вычисления на данной подвыборке будут SIMD-конформными (рис. 3).

В случае, когда размеры интегрального изображения w (ширина) и h (высота) кратны, соответственно, dx и dy, данная процедура эквивалентна переупорядочению элементов интегрального изображения и преобразованию смещений согласно формулам:

$$X' = x \; mod \; dx;$$

$$x' = x \; div \; dx;$$

$$w' = w \; div \; dx;$$

$$x'' = w' * X' + x'.$$

Применив их (для y и h все аналогично) получим x'' и y'' — координаты, в которые должен быть перемещен элемент интегрального изображения, исходно находившийся в позиции (x, y).

Экспериментальные оценки показывают ускорение предложенного алгоритма по сравнению с вычислением «в лоб» на 5–15 % при разрежении в 2–4 раза с учетом времени, требуемого на переупорядочение интегрального изображения. При большем разрежении разница в производительности снижается. Это можно объяснить увеличением количества различных возможных фаз смещений, что ухудшает локальность обращений к памяти.

3. Оптимизация алгоритма Виолы—Джонса с использованием OpenCL для платформы APU Fusion

Технология OpenCL предполагает разделение кода программы на «последовательную» часть (выполняемую на CPU) и «параллельную», выполняемую на GPU. Слова «последовательный» и «параллельный» взяты в кавычки, так как CPU также обладает SIMD-функциональностью, т. е. выполняемый на нем код также может быть распараллелен, однако в значительно меньшей степени, чем код, выполняемый на GPU.

Все параллельные процессоры в технологии SIMD выполняют один и тот же код, но с разными данными, поэтому задача распараллеливания заключается в разделении исходных данных на части, которые будут параллельно обрабатываться разными потоками. Например, в алгоритме Виолы—Джонса каждый поток должен детектировать наличие объекта в своей области входного изображения.

В SIMD-архитектурах, реализованных на CPU, потоки при обращении к памяти всегда читают последовательно распложенные ячейки (т. е. первый поток читает первое слово памяти, второй — непосредственно следующее за ним и т. д.) — так называемые *coalesced-операции*. Особенностью GPU по сравнению с другими SIMD-устройствами является возможность доступа потоков к произвольно расположенным ячейкам памяти, однако скорость этого доступа очень сильно варьируется в зависимости от многих факторов, и оптимизация алгоритма требует его реализации таким образом, чтобы производить только coalesced-операции доступа к памяти.

При этом особенность работы слабого классификатора в том, что из массива бинов читается элемент, номер которого зависит от результата проверки признаков Хаара — и, таким образом, в разных потоках читаются разные и не предсказуемые заранее элементы, т. е. данное чтение не является coalesced-операцией. Это значительно замедляет GPU-реализацию слабого классификатора. Оптимизация возможна за счет использования особого типа памяти, поддерживаемой GPU — константной текстурной памяти. Элементы данной памяти должны быть особенным образом подготовлены перед запуском GPU-потоков и не могут быть из потоков изменены, но скорость произвольного чтения из данной памяти значительно превышает не-coalesced чтения из обычной памяти. Данная память имеет особенную адресацию, оптимизированную под хранение одно-, двух- и трехмерные изображений. Был проведен ряд тестов, который показал существенный выигрыш в производительности при использовании константной текстурной памяти (в терминах OpenCL — *image*) в качестве контейнера для массива бинов.

Основные принципы оптимизации какого-либо алгоритма с использованием GPU следующие:

1. Разбиение задачи на параллельные потоки. Проведенные авторами тесты показали, что на платформе APU Fusion более высокой производительности удается добиться при увеличении количества потоков.
2. Снижение накладных расходов на передачу управления и данных между CPU- и GPU-кодом.

В работе рассматривалось два варианта разбиение задачи на потоки:

- вариант 1: каждый поток GPU обрабатывает отдельный пиксель изображения (это позволяет реализовать предложенный выше SIMD-конформный алгоритм вычисления признаков Хаара при условии плотного хранения изображения в памяти — отсутствия промежутков между строками, так называемых «страйдов»), разные изображения обрабатываются последовательно;
- вариант 2: несколько изображений склеиваются в одно большое, затем каждый поток GPU обрабатывает один пиксель объединенного изображения.

Второй вариант позволяет увеличить количество параллельных потоков, но увеличивает накладные расходы на сборку изображения из частей.

Снижение накладных расходов предполагает сравнение разных схем передачи управления от CPU к GPU. Рассматривались следующие варианты:

- вариант 1: ядро алгоритма (процедура, выполняемая каждым потоком на GPU) содержит только вычисление слабого классификатора, а два внешних цикла (по всем сильным классификаторам и по всем слабым классификаторам в сильном) вычисляются на CPU;
- вариант 2: ядро алгоритма содержит вычисление сильного классификатора (т. е. цикл по слабым классификаторам), а цикл по сильным классификаторам вычисляется на CPU;

– вариант 3: ядро содержит вычисление целого каскада (т. е. двойной цикл — по сильным и слабым классификаторам);
– вариант 4: так как в большинстве реальных приложений алгоритма Виолы—Джонса в каждой точке изображения вычисляется не один, а несколько независимых каскадов, в четвертом варианте ядро содержало тройной цикл — по каскадам, сильным классификаторам каждого каскада и слабым классификаторам каждого сильного. Так как OpenCL не поддерживает четырехмерных константных текстурных изображений, они были реализованы в виде трехмерных изображений, одним из измерений которого на самом деле являлся двумерный срез.

Отдельным вопросом является порядок вычисления каскадов на GPU. Как уже указывалось, на CPU значительного ускорения удается добиться за счет того, что если хотя бы один сильный классификатор ответил -1, то дальнейшее рассмотрение гипотезы о существовании в данной области искомого объекта бессмысленно, и можно прекращать вычисления всего каскада и переходить к следующему.

Однако каскадная схема предполагает наличие точек ветвления (операторов if-then-else), что в свою очередь на параллельной архитектуре может приводить к существенному замедлению работы алгоритма (так как в случае если в части потоков условие ветвления выполнено, а в части — нет, то нарушается SIMD-конформность). Если же все потоки идут по одной и той же ветке (выйти из цикла по классификаторам или завершить работу), то наличие ветвления приводит к ускорению работы, как и в последовательной реализации алгоритма.

4. Базовый эксперимент

В базовом эксперименте сравнивались 4 описанные варианта схемы передачи управления от CPU к GPU. Для этого они использовались в автоматической системе классификации транспортных средств (АКТС) [12]. АКТС запускалась в специальном режиме, в котором была включена лишь подсистема, предназначенная для нахождения колес на изображениях с помощью алгоритма Виолы—Джонса [13]. Поиск колес алгоритмом Виолы—Джонса производился в области размером 133 × 44 пикселя.

На вход системе подавался одинаковый тестовый видеоролик, содержащий два автомобиля: легковой — с двумя колесными осями и грузовой — с шестью. В качестве показателя быстродействия алгоритма использовалось среднее за проезд автомобиля число кадров в секунду (FPS — frames per second), которые успевала обработать система. Эксперимент проводился на двух операционных системах: Windows и Linux. На Windows использовался компилятор C++, поставляемый со средой разработки «Visual Studio 9 2008», а на Linux — GCC версии 4.6.3.

Таблица 1

Производительность алгоритма (число кадров, обрабатываемых в секунду) (Результаты эксперимента)

ОС	Windows		Linux	
Тип автомобиля	**легковой**	**грузовой**	**легковой**	**грузовой**
Вариант 1	0,6	0,8	0,5	0,6
Вариант 2	1,9	2,2	2,1	2,4
Вариант 3	10,8	15,3	13,1	14,7
Вариант 4	12,8	15,3	10,5	12,9
Исполнения на CPU	7,8	9,5	18,7	20,4

Результаты эксперимента сведены в следующую таблицу (табл. 1).

Все числа в таблице 1 представляют собой усредненное значение FPS работы системы за время проезда транспортного средства, то есть чем выше значение, тем быстрее работает система. Можно заметить, что второй вариант реализации параллельного алгоритма в 3–4 раза опережает первый, а третий — в 5–7 раз второй. Четвертый вариант не дает существенного выигрыша, более того, на операционной системе Linux он даже несколько уступает третьему варианту.

Лидером по быстродействию оказалась последовательная (CPU) реализация в операционной системе Linux. Интересно, что эта реализация опережает больше чем в два раза такую же в системе Windows, хотя код в обоих случаях в точности один и тот же. Отсюда можно сделать вывод о существенной разнице в работе компиляторов.

Неожиданным является то, что каждый из вариантов GPU-реализации замедляется при переходе от Windows к Linux в среднем на 20 % (при использовании совпадающего исходного кода). Следует заметить, что компилятор OpenCL, поставляется компанией AMD в составе драйвера и можно было предположить, что он совпадает, однако экспериментом это не подтверждается.

Было проведено профилирование времени работы алгоритма с помощью утилиты AMD APP Profiler. На рис. 4 можно видеть скриншот работы данной утилиты: сверху находится шкала времени, а снизу представлена диаграмма вызовов OpenCL API — функций и загруженности видеоядра.

На рис. 4 можно увидеть два участка загруженности видеосистемы, которые соответствуют проездам тестовых автомобилей (алгоритм Виолы—Джонса запускается только в случае, если детектирован проезд). Первый короткий участок соответствует проезду легкового автомобиля, а второй, более длинный, — грузового.

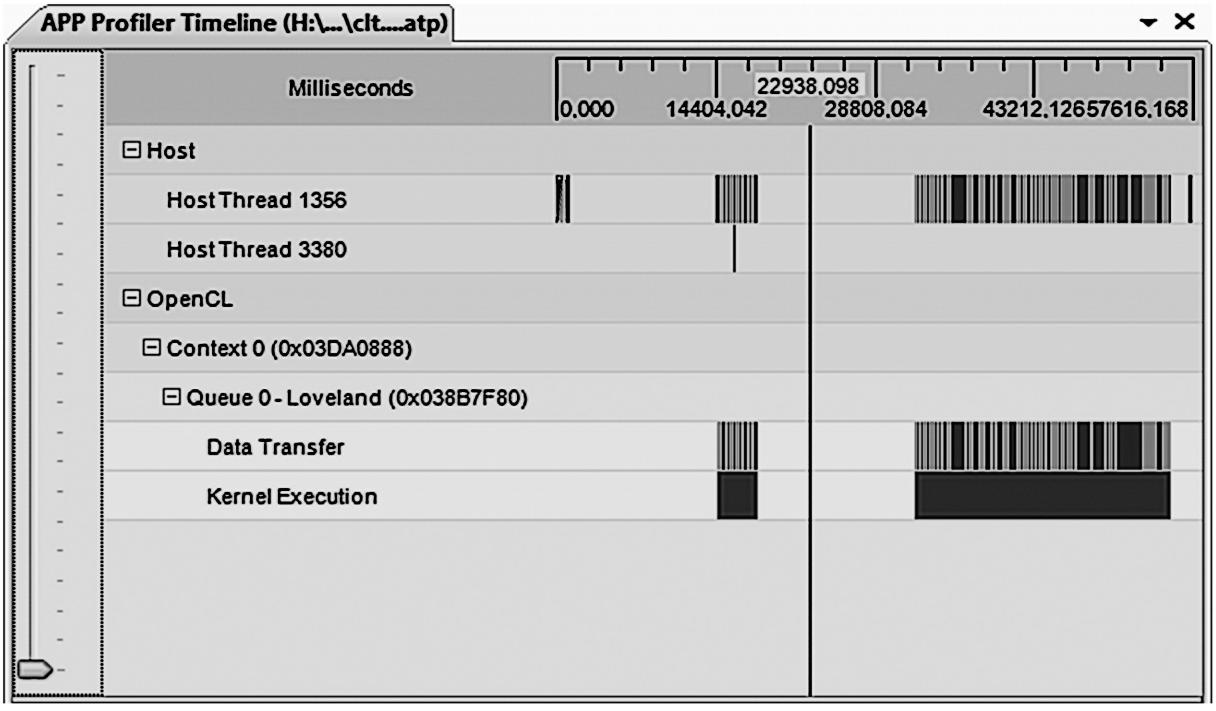

Рис. 4. Скриншот программы AMD APP Profiler

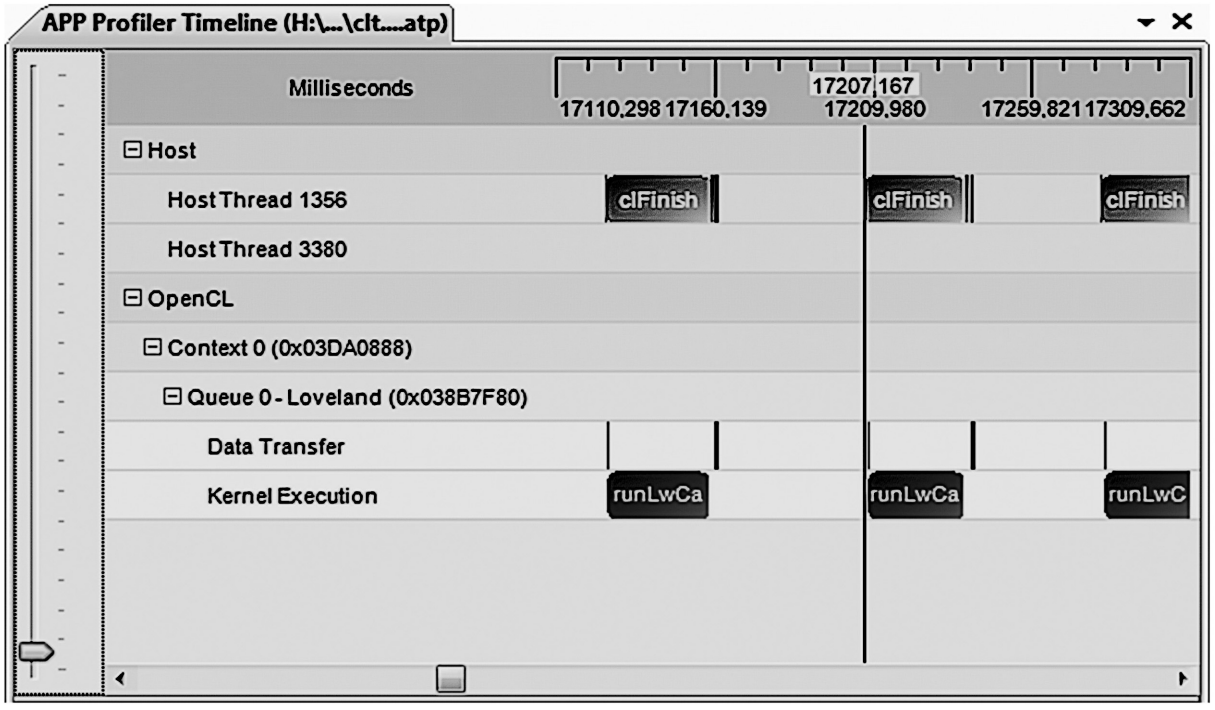

Рис. 5. Скриншот программы AMD APP Profiler для запуска системы на Windows

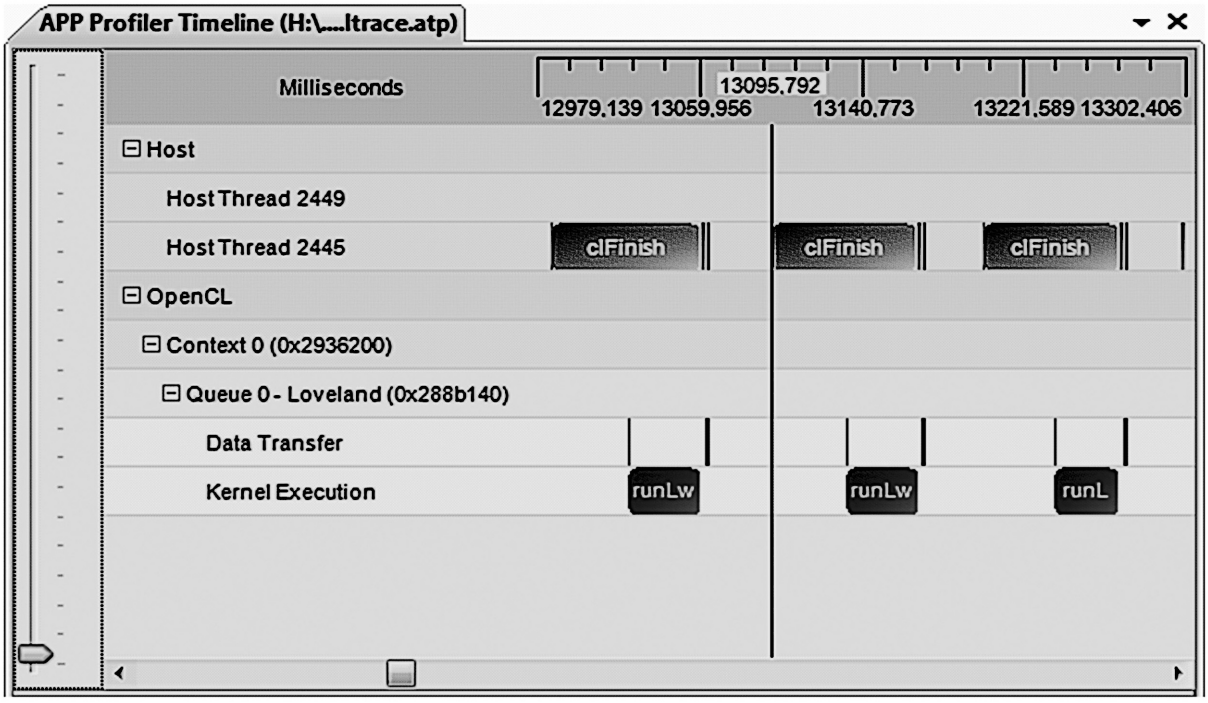

Рис. 6. Скриншот программы AMD APP Profiler для запуска системы на Linux

Увеличим масштаб, чтобы более детально рассмотреть диаграмму при запуске системы на Windows:

На рис. 5 видно, что как только происходит загрузка ядра в очередь на исполнение и вызывается функция clFinish(), ожидающая процесса выполнения ядра на GPU (строка Host Tread), так почти сразу же происходит и вычисление этого самого ядра (строка Kernel Execution).

Однако при запуске системы на Linux картина отличается.

Как можно видеть на рис. 6, с момента добавления ядра в очередь исполнения и запуска функции ожидания исполнения до непосредственного начала исполнения ядра под Linux проходит существенный промежуток времени, который по длительности сравним со временем выполнения ядра. То есть примерно половина времени уходит на накладные расходы, доля которых под Windows незначительна.

5. Исследование влияния применения каскадной схемы

Для исследования влияния на производительность параллельной реализации применения каскадной схемы был проведен эксперимент, в котором запускались те же самые ядра, что и в базовом эксперименте, но в них были убраны все проверки условий «если один сильный классификатор вернул *−1,* то перейти к следующему каскаду».

Таблица 2

Результаты эксперимента влияния применения каскадной схемы

ОС	Windows		Linux	
Тип автомобиля	**легковой**	**грузовой**	**легковой**	**грузовой**
Версия 1	0,5 (0,6)	0,6 (0,8)	0,4 (0,5)	0,5 (0,6)
Версия 2	1,4 (1,9)	1,6 (2,2)	1,5 (2,1)	1,7 (2,4)
Версия 3	2,9 (10,8)	3,2 (15,3)	3,0 (13,1)	3,2 (14,7)
Версия 4	2,9 (12,8)	3,2 (15,3)	1,5 (10,5)	1,6 (12,9)
Исполнение на ЦП	0,3 (7,8)	0,3 (9,5)	0,7 (18,7)	0,7 (20,4)

Результаты эксперимента можно наблюдать в следующей таблице (табл. 2).

В *таблице 2* приведена кадровая частота системы. Для наглядности сравнения результатов в скобках продублированы данные, полученные в базовом эксперименте. Таким образом, можно видеть, что каскадная схема в параллельной реализации дает ускорение от 3 до 5 раз. Глядя на последнюю строку таблицы, можно сделать вывод, что при исполнении на ЦП каскадная схема на том же наборе изображений дает ускорение в 25–30 раз.

6. Исследование влияния размера обрабатываемого изображения

Для исследования, как на производительность повлияет параллельная обработка сразу нескольких кадров, был поставлен эксперимент, в котором входное изображение склеивалось из нескольких (5–70) исходных кадров. Как и в базовом эксперименте, каждый пиксель полученного изображения обрабатывался на GPU отдельным потоком.

Эксперимент проводился с вариантом 3 схемы передачи данных и управления между GPU и CPU, так как она была признана имеющей наибольшую производительность на операционной системе Linux и достаточно высокую на Windows. Измеренный показатель FPS для нормировки умножался на число исходных кадров, включенных в обрабатываемое изображение. Результаты эксперимента представлены в таблице 3.

Таблица 3

Зависимость производительности от размера обрабатываемого изображения

Тип автомобиля	**Легковой**		**Грузовой**	
Количество дублирований	**Производительность, FPS**	**Производительность * количество кадров, FPS**	**Производительность, FPS**	**Производительность * количество кадров, FPS**
1	12,005	12,005	14,358	14,358
5	4,236	21,180	5,118	25,590
10	2,954	29,540	3,020	30,200
15	1,715	25,725	2,031	30,465
20	1,606	32,120	1,675	33,500
25	1,363	34,075	1,357	33,925
30	1,080	32,400	1,086	32,580
35	0,946	33,110	0,947	33,145
40	0,792	31,680	0,874	34,960
45	0,752	33,840	0,732	32,940
50	0,647	32,350	0,664	33,200
55	0,584	32,120	0,599	32,945
60	0,566	33,960	0,560	33,600
65	0,528	34,320	0,529	34,385
70	0,494	34,580	0,495	34,650
75	0,444	33,300	0,441	33,075
80	0,445	35,600	0,447	35,760
85	0,409	34,765	0,409	34,765
90	0,376	33,840	0,375	33,750
95	0,352	33,440	0,351	33,345
100	0,358	35,800	0,359	35,900

Полученные результаты можно наглядно представить в виде графика.

Как видно из таблицы 3 и графика 1, объединение 20–25 кадров в единое изображение позволяет получить производительность в 2–2,5 раза больше исходной производительности.

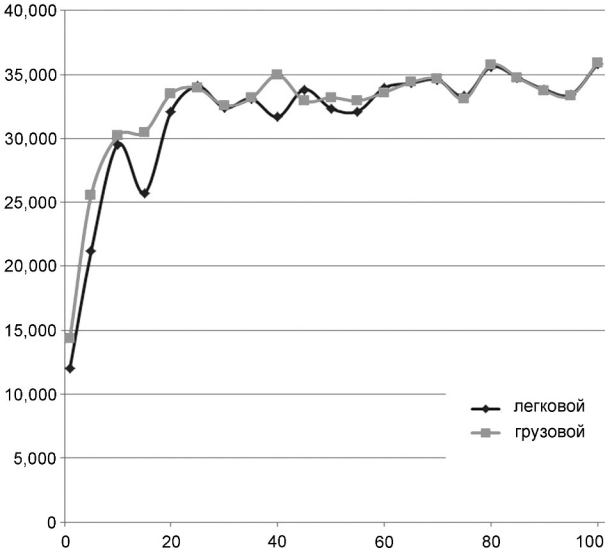

График 1. Зависимость производительности от количества параллельно обрабатываемых кадров

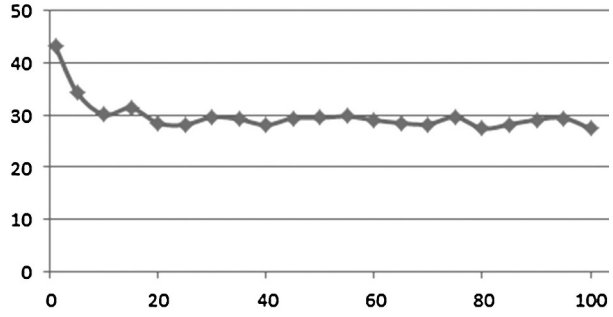

График 2. Удельное время работы алгоритма, мс

Для наглядности полученных результатов можно привести еще один график.

На графике 2 изображена зависимость удельного времени работы алгоритма от количества параллельно обрабатываемых кадров. Другими словами, ордината каждой точки графика представляет собой среднее время работы алгоритма на составном изображении, деленное на количество этих составных частей.

Заключение

Проведенный эксперимент показал, что технология OpenCL принципиально позволяет ускорить работу алгоритма Виолы—Джонса, особенно при

организации параллельной обработки нескольких кадров.

Однако вместе с тем полученные результаты демонстрируют значительную зависимость скорости работы программного кода, использующего OpenCL, от сторонних факторов (таких как выбор операционной системы). Так, в ходе проделанной работы было показано, что текущее поколение APU не дает кратного преимущества по отношению к процессорам общего назначения при условии использования оптимизированного кода под Linux. С нашей точки зрения, это указывает на недостаточную готовность технологии OpenCL к широкому использованию.

Тем не менее, в дальнейшем планируется продолжить работу над увеличением быстродействия алгоритма. Одним из вариантов дальнейшего развития может быть перемещение масштабирования каскадов с CPU на GPU.

Литература

1. *Flynn M.* Very high-speed computing systems // Proceedings of the IEEE. Dec. 1966. Vol.54. No.12, P. 1901,1909.
2. [Электронный ресурс] http://gpgpu.org/
3. *Obukhov A*. Face Detection with CUDA, GraphiCon, 2009.
4. *Hefenbrock D*., *Oberg J*., *Nhat Thanh*, *Kastner R*., *Baden S. B*. Accelerating Viola-Jones Face Detection to FPGA-Level Using GPUs, 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2010. P. 11–18.
5. *Krpec J*., *Němec M*. Face Detection CUDA Accelerating, ACHI 2012. The Fifth International Conference on Advances in Computer-Human Interactions. P. 155–160.
6. [Электронный ресурс] http://www.khronos.org/opencl/
7. [Электронный ресурс] http://www.amd.com/la/Documents/49282_G-Series_platform_brief.pdf
8. *Viola P.* and *Jones M*., «Rapid object detection using a boosted cascade of simple features» // Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). V. 1. No. 1. IEEE Computer Society, 2001. P. 511–518.
9. *Усилин С. А.*, *Николаев Д. П.*, *Постников В. В.* Поиск объектов в видеопотоке при известных кинематике и геометрической модели сцены // Труды 53-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук». Часть IX. Инновации и высокие технологии. М.: МФТИ, 2010. С. 67–69.
10. *Усилин С. А.*, *Николаев Д. П.*, *Постников В. В.* Локализация, ориентация и идентификация документов с фиксированной геометрией на изображении // Труды Института системного анализа РАН. Обработка информационных и графических ресурсов / Под ред. Арлазарова В. Л. М.: Красанд/URSS, 2010. С. 248–261.
11. *Schapire R. E.*, *Singer Y.* «Improved Boosting Algorithms Using Confidence-rated Predictions» // Machine Learning. December 1999. V. 37, Issue 3. P. 297–336.
12. *Grigoryev A., Khanipov T., Nikolaev D.* Determination of axle count for vehicle recognition and classification. 8th Open German-Russian Workshop «Pattern Recognition and Image Understanding»: Workshop proceedings. Nizhny Novgorod, 2011. P. 89–91.
13. *Григорьев А. С.*, *Гладков А.*, *Николаев Д. П.* Система подсчета колесных осей транспортных средств и ее оптимизация с помощью программного пакета NOMAD // Информационные технологии и системы (ИТиС'12). Петрозаводск, 2012. М.: ИППИ РАН, 2012. С. 396–400.

Гладилин Сергей Александрович. Науч. сотр. ИППИ РАН. Окончил в 2002 г. МГУ. К. ф.-м. н. Количество печатных работ: 12. Область научных интересов: зрительный интеллект, алгоритмы обработки изображений. E-mail: gladilin@iitp.ru

Григорьев Антон Сергеевич. Вед. разработчик ООО «Визиллект Сервис». Окончил в 2012 г. МФТИ. Аспирант МФТИ. Количество печатных работ: 8. Область научных интересов: зрительный интеллект, индустриальные распознающие системы. E-mail: me@ansgri.com

Котов Антон Александрович. Окончил в 2013 г. МФТИ. Количество печатных работ: 1. Область научных интересов: зрительный интеллект, алгоритмы обработки изображений. E-mail: kotov.anton.1@gmail.com

Николаев Дмитрий Петрович. Зав. сектором ИППИ РАН. К. ф.-м. н. Окончил в 2000 г. МГУ. Количество печатных работ: 117. Область научных интересов: алгоритмы обработки изображений, зрительный интеллект. E-mail: dimonstr@iitp.ru