Реализация алгоритма Viola-Jones для архитектуры NVIDIA CUDA

И.И. Зиновьев, Т.Е. Овчинникова, П.Ю. Шамин
Владимирский государственный университет имени А.Г. и Н.Г. Столетовых
e-mail: izinoviev@vlsu.ru

Комплекс алгоритмов для обнаружения объектов, представленный в 2001 году Полом Виолой (Paul Viola) и Майклом Джонсом (Michael Jones) [1], логически состоит из двух компонентов: обучение классификатора и непосредственно обнаружение объектов на изображении. На практике основные требования к скорости работы предъявляются ко второму алгоритму. В данной работе рассматривается реализация этого алгоритма для архитектуры NVIDIA CUDA, которая позволяет в несколько раз ускорить его работу по сравнению с реализаций на центральном процессоре в библиотеке OpenCV [7].

Алгоритм Viola-Jones обнаружения объектов хорошо известен. Поэтому перечислим только наиболее значимые для рассматриваемой темы его особенности:

1. Большое количество обращений к памяти. Для вычисления отклика одного классификатора требуется от 11 до 15 обращений к памяти.

2. Вычисление интегрального изображения – особое представление исходного изображения, позволяющее быстрое вычисление характерных черт (особенностей), используемых при классификации изображения. Каждый пиксель интегрального изображения представляет собой сумму значений всех пикселей исходного изображения, расположенных выше и левее текущего пикселя

3. Применение каскада классификаторов, представляющего собой объединение в цепочку классификаторов по такому принципу, что выход одного является входом для другого. Поэтому, если участок изображения не проходит один классификатор, он сразу отбрасывается и не идет по всей цепочке.

Для эффективной реализации алгоритма необходимо также учесть особенности архитектуры NVIDIA CUDA, с которыми можно ознакомиться в [5].

Рассматриваемый алгоритм реализован как одно CUDA-ядро (рис. 1). Это позволило минимизировать пересылки данных через глобальную память графического ускорителя.

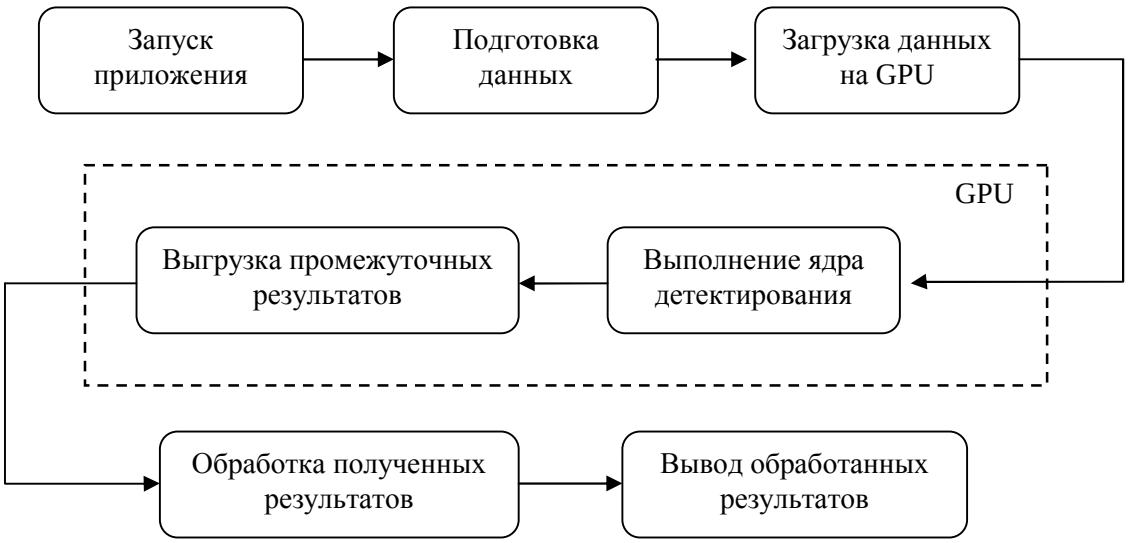

Рис. 1. Схема работы алгоритма Viola-Jones, реализованного с использованием графического ускорителя

Правильный выбор размера блока потоков и количества используемых регистров позволяет достичь значения коэффициента использования мультипроцессора порядка 2/3, что является достаточным для нивелирования влияния латентности при работе с памятью.

Исходные для алгоритма данные, а именно каскад классификаторов и анализируемое изображение, размещаются в текстурной и константной памяти графического ускорителя. Каскад классификаторов является неизменным, необходим всем потокам одновременно, что позволяет эффективно использовать особенности работы константной памяти. Размещение исходного изображения в текстурной памяти позволяет реализовать эффективный алгоритм его масштабирования путем выборки из текстуры с аппаратной билинейной фильтрацией.

Из-за большого числа обращений к памяти при работе рассматриваемого алгоритма необходимо использовать разделяемую память графического ускорителя как управляемый L1 кэш. Все потоки мультипроцессора анализируют частично перекрывающиеся участки изображения, поэтому в разделяемую память копируется часть исходного изображения, достаточная для обеспечения всех потоков данными для классификации.

Узким местом реализации алгоритма на графическом ускорителе является необходимость пересылки данных между хостом и ускорителем. Для поиска лиц разного размера алгоритм Viola-Jones предусматривает анализ изображения в различных масштабах. Полученные промежуточные результаты сразу передаются в оперативную память компьютера. Применение технологии «zero copy», реализованной во всех современных графических ускорителях, позволяет сократить накладные расходы при пересылке данных практически к нулю за счет их асинхронного выполнения.

Как упоминалось выше, масштабирование изображения реализовано, используя возможности текстурной памяти. Масштабированное изображение считывается непосредственно в разделяемую память, где и осуществляется его анализ. Это исключает дополнительные обращения в глобальную память, что положительно сказывается на времени работы алгоритма. Проведенное сравнение с алгоритмом масштабирования изображения, реализованным в библиотеке примитивов NPP, это подтверждает (рис.2).

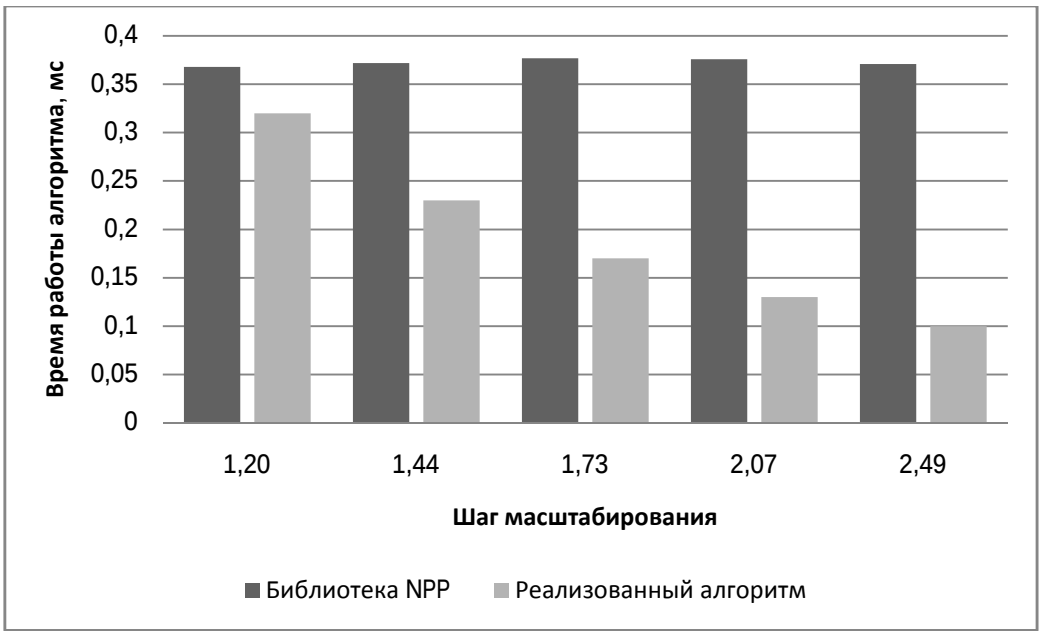

Рис. 2. Сравнение алгоритмов масштабирования изображения

Вычисление интегрального изображения является ключевым шагом в работе алгоритма Viola-Jones. Процесс вычисления интегрального изображения хорошо описан и

изучен. Самым очевидным способом распараллеливания этого процесса является разбиение его на два этапа. На первом этапе проводится независимое суммирование по строкам изображения, на втором – по столбцам. Однако для графического ускорителя реализация такого алгоритма усложняется из-за необходимости синхронизации результатов работы отдельных мультипроцессоров. Для облегчения процесса расчета интегрального изображения в данной работе предлагается использовать частичное интегральное изображение, которое вычисляется независимо для участков исходного изображения, уже размещенных в разделяемой памяти. Это становится возможным, так как в разделяемой памяти каждого мультипроцессора содержится достаточно данных для всех его потоков. Благодаря этому работа отдельных мультипроцессоров становится независимой, что исключает временные затраты на синхронизацию их работы. На рисунке 3 приводится сравнительная характеристика полученного алгоритма вычисления интегрального изображения с аналогичным алгоритмом, содержащимся в библиотеке NPP.

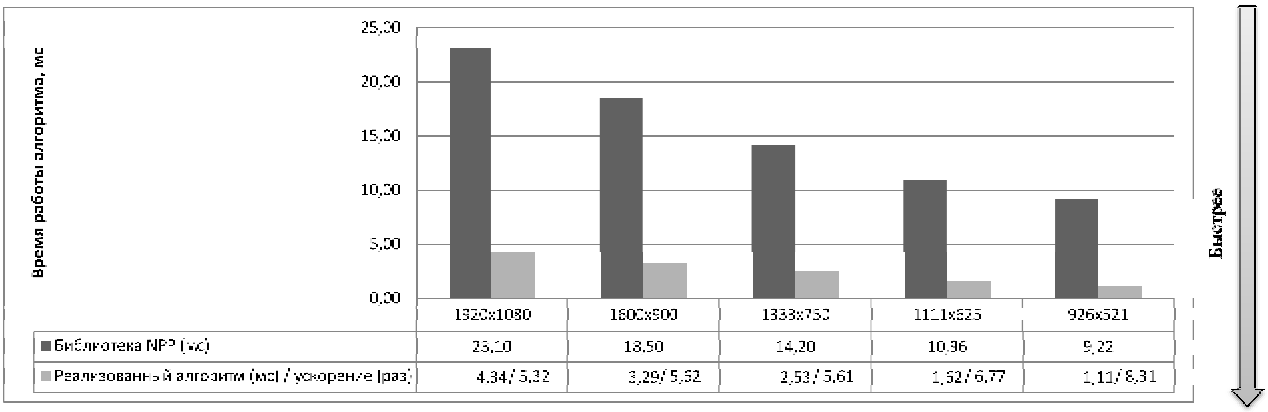

Рис. 3. Сравнение производительности алгоритмов вычисления интегрального изображения

Тестирование полученной реализации алгоритма Viola-Jones проводилось на изображениях трех размеров: 1920x1080, 1080x720 и 640x360. Для тестирования применялся каскад классификаторов, предложенный в работе [2]. Осуществлялся поиск лиц с минимальным размером 20x20 с шагом масштабирования исходного изображения 1,2. Рисунок 4 содержит результаты измерения производительности алгоритма в целом на различных графических ускорителях. Для сравнения приведена производительность алгоритма, реализованного в библиотеке OpenCV.

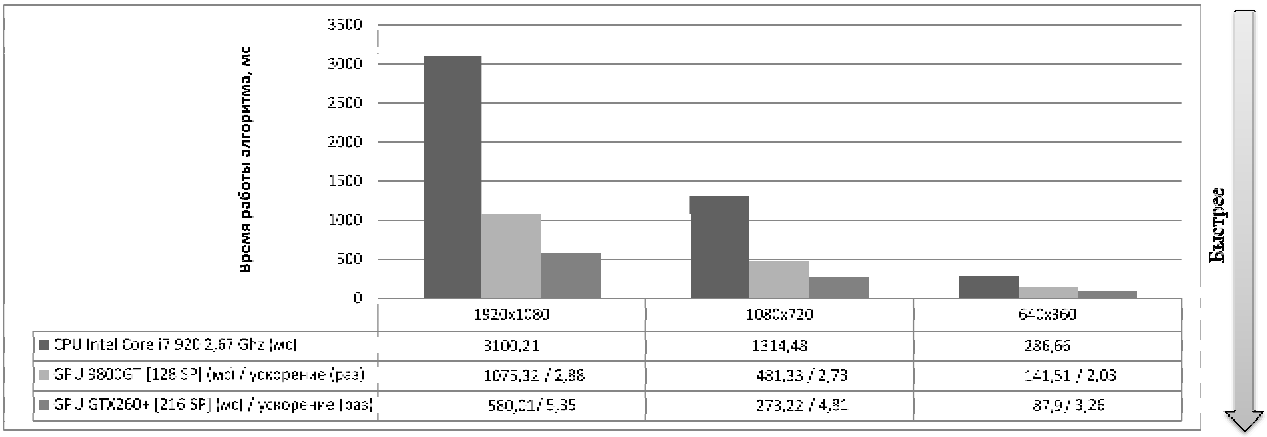

Рис. 4. Результаты измерения производительности алгоритма Viola-Jones

Можно отметить ускорение работы алгоритма на графическом ускорителе от 2 до 5 раз в сравнении с однопоточным вариантом, работающим на центральном процессоре. Такой результат в несколько раз превышает полученный автором работы [4]. Немаловажным является хорошая масштабируемость полученной реализации алгоритма. Это позволяет рассчитывать на еще большее ускорение на более современных графических ускорителях.

Выводы

В данной работе предложен оригинальный вариант эффективной реализации алгоритма Viola-Jones для архитектуры графических ускорителей производства NVIDIA. Следует отметить, что сам алгоритм не слишком приспособлен для реализации на графической карте. Большое число обращений к памяти, простои потоков при работе каскада классификаторов негативно сказываются на производительности. Но даже при этом полученная реализация алгоритма на современных графических ускорителях работает в несколько раз быстрее, чем аналоги на центральном процессоре.

Литература

1. Paul Viola, Michael J. Jones. Robust real-time face detection // International journal of computer vision 57(2), 137–154, 2004
2. Rainer Lienhart, Alexander Kuranov, Vadim Pisarevsky. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.
   URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.139.4825&rep=rep1&type=pdf (дата обращения: 21.07.2010)
3. Квасов Д.С., Зиновьев И.И., Кокорин И.Г., Коробко С.С. Обнаружение и идентификация лиц на растровом изображении с применением метода Виолы и Джонса // Труды XVII Всероссийской научно-методической конференции Телематика-2010, СПб, С. 253-254.
4. Jesse Patrick Harvey. GPU Acceleration of Object Classification Algorithms Using NVIDIA CUDA
   URL: https://ritdml.rit.edu/bitstream/handle/1850/10894/35445_pdf_00B0B24A-DFD8-11DE-9A30-D21AD352ABB1.pdf?sequence=1. (дата обращения: 11.10.2010)
5. NVIDIA CUDA C Programming Guide, 2010. // CUDA Toolkit
6. CUDA C Best Practices Guide, 2010. // CUDA Toolkit
7. Документация открытой библиотеки OpenCV.
   URL: http://opencv.willowgarage.com/wiki/ (дата обращения: 15.11.2010)