

МЕТОДЫ ОБРАБОТКИ ИЗОБРАЖЕНИЙ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ ОБЩЕГО НАЗНАЧЕНИЯ С ПАРАЛЛЕЛЬНОЙ АРХИТЕКТУРОЙ

© 2012 г. В. И. Филатов, студент

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

E-mail: sigal89@mail.ru

Рассматриваются принципы массивно-параллельных вычислений при цифровой обработке сигналов с помощью технологии NVIDIA CUDA на примере таких операций, как инвертирование значений яркости изображения, гамма-коррекция, оператор Собеля. Проведена оценка основных методов цифровой обработки изображений с использованием массового параллелизма вычислений на графическом процессоре. Выполнена реализация этих методов на центральном и графическом процессорах и их сравнение по таким параметрам, как время выполнения обработки, размер используемых изображений, размер используемых архитектурой CUDA блоков памяти.

Ключевые слова: массивно-параллельные вычисления, обработка изображений, варп.

Коды OCIS: 100.0100, 100.2000

Поступила в редакцию 29.05.2012

Введение

Обработка сигналов и изображений на основе современных компьютерных технологий широко используется в биомедицине, системах искусственного интеллекта, микроскопии и других областях науки и техники. С ростом сложности методов и технологий требования к производительности вычислительных систем возрастают. Быстродействие персональных компьютеров связано с тактовой частотой центрального процессора. За время своего существования, более тридцати лет, процессоры архитектуры x86 увеличили свою тактовую частоту почти в 700 раз [1].

Однако рост тактовой частоты связан с повышением энергопотребления пропорционально четвертой степени частоты. В последнее время увеличение числа транзисторов на кристалле стало более затруднительным, что послужило причиной активного развития многопроцессорных систем. В качестве альтернативного варианта рассматривается возможность производить вычисления с помощью графического процессора GPU – Graphics Processing Unit.

Термин GPU был введен для обозначения эволюции графического ускорителя в мощное программируемое устройство (процессор), спо-

собное решать широкий класс задач, не связанных с графикой. В настоящее время графические процессоры представляют собой массивно-параллельные вычислительные устройства с высоким быстродействием (свыше одного терафлопа) и большим объемом собственной памяти (DRAM) [1].

На начальном этапе развития GPU быстро совершенствовались и в настоящий момент их вычислительная способность, реализуемая в виде количества обработанных операций, превышает вычислительные способности центральных процессоров общего назначения.

Принципы архитектуры графического процессора видеокарты отличаются от архитектуры центрального процессора. В отличие от последнего, графический процессор видеокарты предназначен для массивно-параллельных вычислений [2]. При рассмотрении динамики вычислительной мощности графического процессора можно заметить, что рост производительности происходил более быстрыми темпами, чем в случае центрального процессора [1].

Целью данной работы является исследование возможностей применения технологии NVIDIA CUDA к широко используемым методам обработки изображений. Анализируются способы оптимизации методов на основе CUDA

путем выбора наилучших значений основных переменных, характеризующих параллельные вычисления: число нитей, блоков, решеток. CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура, позволяющая производить вычисления с использованием графических процессоров NVIDIA, поддерживающих технологию произвольных вычислений на видеокартах.

Технология CUDA

Применение современных технологий для обработки данных с помощью графического процессора видеокарты позволяет заметно сократить время вычислений за счет массивного параллелизма выполняемых операций. Данная зависимость описывается законом Амдала (табл. 1). В 1967 году Джин Амдал сформулировал закон получения максимального ускорения при распараллеливании программы [3]: если алгоритм разделить на несколько частей, то общее время его выполнения на параллельной системе будет не меньше времени выполнения самого длинного фрагмента. Если разделяемая часть кода f может быть равномерно формально распределена по p процессорам, то указанная закономерность может быть записана в форме

$$S \leq 1/(f + (1-f)/p), \quad (1)$$

где S – коэффициент ускорения, f – доля операций, которые нужно выполнить последовательно, p – число процессоров. Архитектура графического процессора хорошо подходит для использования ресурсоемких вычислительных задач.

Основой дальнейших вычислений в нашей статье является технология NVIDIA CUDA, которая “дает разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя,

Таблица 1. Доля вычислительных операций, описываемых законом Амдала

	$p = 2$	$p = 8$	$p = 1000$
$f = 0,9$	1,05	1,1	1,11
$f = 0,5$	1,33	1,77	2,0
$f = 0,1$	1,81	4,71	9,91

p – количество потоков, f – доля последовательных вычислений.

управлять его памятью и организовывать на нем сложные параллельные вычисления” [1].

Модель технологии CUDA строится с помощью трех основных объемных единиц: потока (нити), блока и сетки (решетки). Поток рассматривается как набор данных, которые необходимо обработать. Совокупность потоков, представляющая двумерный или трехмерный массив, в терминологии NVIDIA CUDA называется блоком. Совокупность блоков описывается терминами “решетка” или “сетка”. Все данные обрабатываются специальной конструкцией в CUDA, названной варпом (группа из 32 потоков). Модель технологии CUDA изображена на рис. 1.

Пользуясь приведенной схемой, можно получить представление о том, что для выполнения закона Амдала и получения высокой степени параллелизма желательно выполнение следующих условий:

- так как потоков в блоке может быть не более 512, эффективным является использование разделения на блоки размерами, кратными 128,
- в силу того, что каждая нить выполняется в решетке параллельно (в первом приближении), оптимально задание для каждой нити

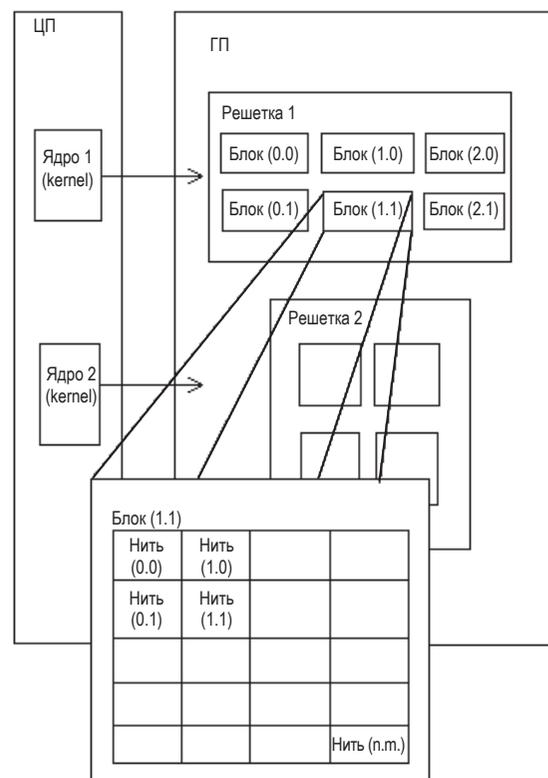


Рис. 1. Модель технологии CUDA.

своей вычислительной операции над пикселем. Таким образом, за один вызов ядра (ядро – функция, вызываемая на GPU, в которой осуществляется весь объем массивно-параллельных вычислений; на одном ядре выполняется одна решетка блоков) можно обработать целое изображение,

- копирование памяти из DRAM на видеокарту и обратно является наиболее ресурсоемким по времени выполнения этапом и поэтому ограничение использования глобальной памяти является одним из существенных условий оптимизации программ с использованием технологии NVIDIA CUDA.

Анализируя вышеперечисленное, можно сказать, что в первую очередь оптимизация параллелизма вычислений достигается за счет сокращения числа обращений к глобальной памяти и выравнивания данных в памяти по 16 и 32 бит (варпы и полуварпы).

Основные методы оптимизации с помощью параллелизма вычислений

Последовательность действий при работе с графическим процессором видеокарты заключается в выполнении следующих этапов:

- вычислительные операции должны выполняться в памяти графического процессора, поэтому первоначально необходимо получить доступ к данным из пространства видеокарты (при использовании глобальной памяти происходит копирование данных в виртуальное пространство, занимаемое графической картой),

- выделение необходимого объема памяти с помощью графического процессора,

- установка количества используемых блоков и нитей (задается параметрами ядра функции),

- произвести запуск ядра (ядром называется функция, выполняемая на графическом процессоре видеокарты),

- после окончания вычислений следует выполнить копирование памяти из памяти графического процессора в память центрального процессора для дальнейшей работы с полученными данными,

- освобождение выделенной с помощью графического процессора памяти.

Минимизация времени, затрачиваемого на копирование памяти, является наиболее важным условием оптимизации. Другими не менее важными условиями являются соблюдение коалесинга и выравнивание данных в памяти.

Желательно и установление для каждой нити своей вычислительной операции. Количество блоков и их размер учитываются относительно количества нитей, а один блок может выполняться параллельно. Таким образом, при необходимости сохранить количество блоков время, затраченное на исполнение, ядра можно сократить.

Простые и сложные операции обработки изображений

Операции по обработке изображений можно разделить на два типа: простые и сложные. Простой называется операция, при которой яркость пиксела вычисляется вне зависимости от локального окружения. Сложной назовем операцию, при которой получаемое значение яркости пиксела зависит от значений яркости других пикселей.

Простые операции. Для сравнения скорости работы алгоритмов, реализованных с помощью центрального и графического процессоров, рассмотрим операцию инвертирования значений яркости изображений и гамма-коррекцию. Данные операции будут относиться к простым. Операция гамма-коррекция изображения – коррекция функции яркости в зависимости от характеристик устройства вывода [4]. Операция инвертирования значений яркости выполняется в виде простой обработки изображения, где для каждого канала цвета (при использовании цветного RGB изображения) значения яркости каждого пиксела инвертируется относительно максимального значения из диапазона яркостей.

Время, затраченное на выполнение операций инвертирования (рис. 2б) и гамма-коррекции изображения (рис. 2в), представлено в табл. 2. В данной статье для всех вычислений были использованы двухъядерный процессор CPU – Intel® Atom™ N550 1.50GHz и видеокарта NVIDIA GeForce ION 2 512 Mb.

По значениям из табл. 2 следует, что время, затраченное на выполнение операции инвертирования изображения с помощью графического процессора, составляет 8,04 мс. Это почти в 30 раз меньше времени, затраченного при использовании центрального процессора. На производительность вычислений могут влиять различные факторы. Например, обработка квадратных изображений и изображений кратных 16 или 256, осуществляется наилучшим образом. Это связано с архитектурой строения гра-



Рис. 2. Исходные данные и результат работы в виде инвертирования значений яркости пикселей и операции гамма-коррекции (а – исходное изображение, б – операция инвертирования изображения, в – гамма-коррекция).

Таблица 2. Сравнительные данные времени выполнения операций с помощью CPU и GPU

Название операции	Размер изображения в пикселях	Время на центральном процессоре t_{cpu} , мс	Время на графическом процессоре t_{gpu} , мс	t_{cpu}/t_{gpu}
Инвертирование изображения	512×512	8,04	240,65	30
Гамма-коррекция	512×512	501,7	17,86	28
Гамма-коррекция	1024×768	1114	52,67	22
Гамма-коррекция	2448×3264	9363	521,77	17

фических процессоров и параметрами технологии CUDA.

Влияние размера блоков при использовании простых операций. Время выполнения операций удастся сократить за счет изменения размера блоков. Рассмотрим табл. 3, содержащую полученную зависимость времени выполнения операций обработки изображения от размера блоков. Так как физически парал-

лельно происходит имплементация одной решетки (сетки) данных, изменение количества нитей и количества блоков позволяет программисту распределить параллелизм обработки данных. При неизменном количестве нитей увеличение количества блоков будет способствовать меньшей взаимосвязи между соседними блоками. Взаимосвязь, как будет показано ниже, в большей степени необходима

Таблица 3. Зависимость времени выполнения операций от размера блоков для простых операций

Название операции	Размер изображения, пиксел	Размер блоков	Время на графическом процессоре, мс
Гамма-коррекция	512×512	4×4	31,6
Гамма-коррекция	512×512	8×8	18,4
Гамма-коррекция	512×512	16×16	17,86
Гамма-коррекция	512×12	32×32	Обработка не произошла



Рис. 3. Исходное изображение и результат его обработки оператором Собеля.

для сложных операций. В истинности этого высказывания можно убедиться, рассмотрев табл. 3.

С увеличением размера блоков время, затрачиваемое на обработку, уменьшается за счет использования разделяемой памяти (общая память для всех блоков изображения; ее количество ограничено архитектурой видеокарты; в данной статье использовалась видеокарта с 16 кБ разделяемой памяти). При размере блоков 32×32 обработка изображения не происходит, так как исчерпывается объем доступной разделяемой памяти. Создается меньшая взаимосвязь между блоками и происходит экономия времени (меньшая взаимосвязь не требует времени на распределение ресурсов на дополнительные нити).

Сложные операции обработки изображений. В качестве примера сложной операции рассмотрим операцию оценки модуля градиента с помощью оператора Собеля. Исходные данные – полутоновое изображение 512×512. На рис. 3 приведены исходное и результирующее изображения. Время, затраченное на выполнение данной операции, составляет 11,07 мс на графическом процессоре и 217,71 мс – на центральном процессоре (отличие в 18,5 раз).

Для сложных операций достижение коалесинга (а именно, обращения каждой нити к своему адресу, т. е. аналогу биекции, если образ – это адрес операции, а прообраз – нить) является более трудоемким процессом, так как приходится учитывать способ оценки количества используемых нитей и их взаимосвязь при различных размерах маски.

Влияние размера блоков при использовании сложных операций. Оптимизация времени выполнения операций при изменении размера блоков для сложных операций представлена в табл. 4. Аналогично рассмотренному расчету зависимостей размера блоков от времени, затраченного на его выполнения для простых операций, важным условием является достижение большей взаимосвязи блоков (при большей взаимосвязи будут учитываться зависимости от локальных пикселей, и распределение нитей по адресам произойдет другим, более эффективным, образом).

Проанализировав результаты, приведенные в табл. 4, можно увидеть обратную зависимость в отличие от зависимости при простых операциях. При использовании блоков размером 32×32 обработка не происходит по той же причине, что и при использовании простых операций.

Таблица 4. Зависимость времени выполнения операций от размера блоков для сложных операций

Название операции	Размер изображения, пиксел	Размер блоков	Время на графическом процессоре, мс
Оператор Собеля	512×512	4×4	11,07
Оператор Собеля	512×512	8×8	11,82
Оператор Собеля	512×512	16×16	20,04
Оператор Собеля	512×512	32×32	Обработка не произошла

Заключение

В работе рассмотрены основные методы обработки цифровых изображений с использованием массового параллелизма вычислений на графическом процессоре, произведена реализация методов на центральном процессоре и графическом процессоре, и выполнено их сравнение по таким параметрам, как время выполнения обработки, размер используемых изображений, размер используемых архитектурой CUDA блоков памяти. Все методы разделены на два класса: простая обработка изображений и сложная обработка изображений. На основе проведенных исследований можно сделать следующие выводы:

- полученные результаты показывают эффективность реализации программ, поддающихся хорошей степени параллелизма вычислений, с помощью технологии CUDA,

- использование видеокарты (с поддержкой архитектуры CUDA высоких уровней – 2.0 и выше) для реализации вычислений является более выгодным решением по сравнению с использованием центрального процессора, как по

экономическим, так и по вычислительным причинам,

- при применении изображений квадратной формы, обладающих размером кратным 128, достигается наибольшая эффективность вычислений. Для обработки серий изображений или больших изображений самым важным является время, затраченное на копирование памяти (с центрального процессора на графический и в обратном направлении). В таком случае желательно использовать текстурную память [1, 2]. Для простых операций более эффективным способом оптимизации является увеличение размера блоков, а для сложных – их уменьшение.

Подводя итог вышесказанному, отметим, что переход к реализации программ и выполнение вычислений с помощью графических процессоров общего назначения является более эффективным, чем использование расчетов с помощью центрального процессора применительно к областям, связанным с обработкой изображений.

Работа выполнена при финансовой поддержке Министерства образования и науки Российской Федерации.

* * * * *

ЛИТЕРАТУРА

1. Боресков А.В., Харламов А.А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. 232 с.
2. Sanders J., Kandrot E. CUDA by Example. An Introduction to General-Purpose GPU Programming. Addison-Wesley, 2010. 313 p.
3. Барановская Т.П., Лойко В.И., Семенов М.И., Трубилин А.И. Архитектура компьютерных систем и сетей. М.: Финансы и статистика, 2003. 256 с.
4. Фисенко В.Т., Фисенко Т.Ю. Компьютерная обработка и распознавание изображений: учеб. пособие. СПб.: СПбГУ ИТМО, 2008. 192 с.