

# ВЫБОР ОПТИМАЛЬНОГО РЕЖИМА ЧТЕНИЯ ДАННЫХ ИЗ ПАМЯТИ ДЛЯ АЛГОРИТМОВ ОБРАБОТКИ ИЗОБРАЖЕНИЙ ПРИ РЕАЛИЗАЦИИ НА АРХИТЕКТУРЕ CUDA

*А.А. Уваров, Д.Ю. Перцев*

Белорусский государственный университет информатики и радиоэлектроники, кафедра ЭВМ  
ул. Бровки, 6, Минск, Республика Беларусь  
телефон(ы): +(017)2938039; e-mail: Uvarov.Andrey@gmail.com, DmitryPertsev@gmail.com  
web: www.bsuir.by

Целью данного исследования является поиск типа данных, который наилучшим образом подходит для решения задач в области обработки изображений при программировании видеоадаптеров с использованием технологии NVidia CUDA. Для этого проводится анализ типов памяти, доступных разработчику, а также измеряется время, необходимое для выполнения операции чтения некоторого блока данных.

**Ключевые слова** – видеоадаптер, глобальная память, разделяемая память, текстурный блок, транзакции.

## 1 ВВЕДЕНИЕ

Разработка программного обеспечения с использованием видеоадаптера является достаточно новым и малоизученным направлением. Тем не менее, оно является перспективным, т.к. позволяет значительно уменьшить время решения задачи. Такая возможность связана с особенностями архитектуры видеоадаптера и распространяется только на те фрагменты кода программы, которые возможно выполнить параллельно. Корпорация NVidia® первой сумела разработать технологию, которая значительно упрощает процесс написания программы и являющуюся надстройкой над популярным языком программирования С.

Однако при разработке программы по-прежнему приходится учитывать достаточно большое количество особенностей архитектуры видеоадаптера, несоблюдение которых приводит к значительной потере в скорости исполнения кода. Одной из них является работа с памятью.

Основная задача, которая ставилась в данном исследовании – выбор оптимального типа данных для работы с алгоритмами обработки изображения. Это направление выбрано неслучайно, т.к. очень многие алгоритмы в данной области легко поддаются распараллеливанию и, соответственно, их решение с использованием возможностей видеоадаптеров является наиболее эффективным с точки зрения быстродействия. Но применение технологии CUDA в данном случае создает дополнительные трудности. Это связано с тем фактом, что единицей изображения является один пиксель, соответствующий одному байту. В дальнейшем будет показано, что операция чтения из памяти побайтно является неэффективной.

## 2 ТИПЫ ПАМЯТИ, ДОСТУПНЫЕ В АРХИТЕКТУРЕ CUDA

В отличие от универсальных процессоров память доступная программам графического процессора не является однородной. Существуют следующие типы памяти: регистровая, глобальная, разделяемая и константная. Вкратце опишем каждую из них.

Регистровая память является самой быстрой. На ее основе формируется регистровый файл для каждого исполняемого треда. Разработчику этот тип памяти напрямую не доступен.

Следующий тип памяти – разделяемая память. Являясь одной из самых быстрых, доступных разработчику, она обладает очень маленьким объемом, что не позволяет использовать ее для хранения всех необходимых данных, в том числе промежуточных результатов. Для достижения максимальной пропускной способности разделяемая память делится на блоки – специальные ячейки одинакового размера. Треду предоставляется возможность читать данные из любого блока, но не рекомендуется, когда 2 и более треда читают один и тот же блок, т.к. это приводит к конфликтам между тредами за блок разделяемой памяти. Пример без конфликтного обращения показан на рисунке 1. Единственное исключение – когда все потоки, которые исполняются в текущий момент времени, читают один и тот же блок.

Константная память доступна только для чтения, поэтому эффективна только при размещении констант. Ее объем ограничен, но она кэшируется.

Глобальный тип памяти является самым большим по объему, однако, и самым медленным. К тому же отсутствует кэширование данных, что еще больше ухудшает производительности. Технология программирования CUDA предоставляет два основных подхода к работе с глобальной памятью видеопроцессора: непосредственное обращение и использование текстур.

При непосредственном обращении к памяти существует способ ускорить обмен данными – использовать механизм транзакций [1]. Транзакция позволяет объединить 16 обращений от соседних тредов в одно обращение, которое обрабатывается эффективнее. Для формирования транзакций требуется выполнение ряда условий: обращения всех 16 тредов попадают в один 64 или 128 байтный сегмент, который выровнен в памяти; тип используемы

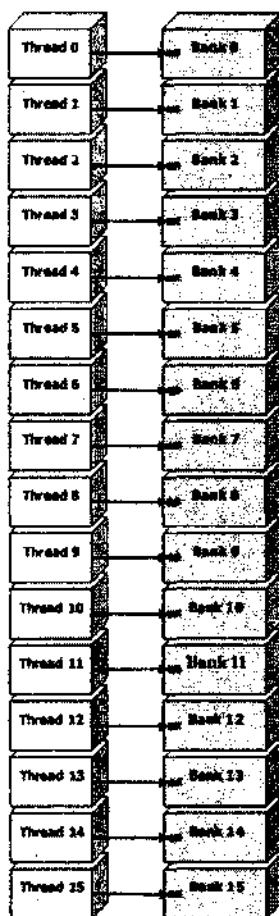


Рис.1. Примеры без конфликтного обращения к разделяемой памяти.

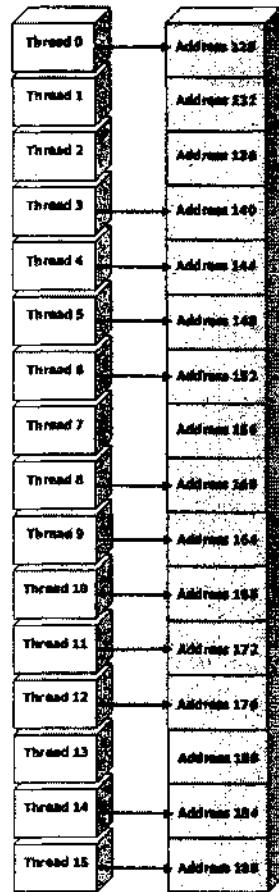
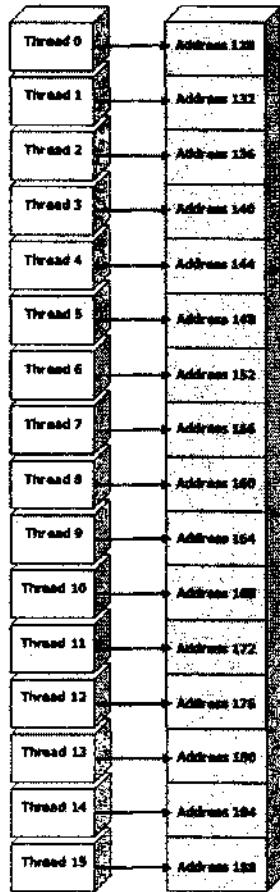
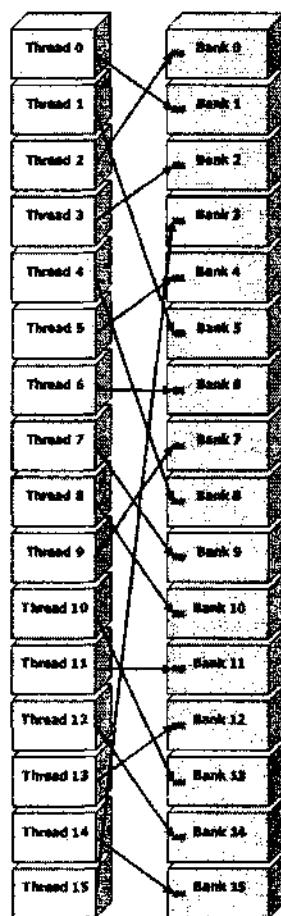


Рис.2. Примеры формирования транзакций.

при обращении должен занимать 32, 64 или 128 бит, k-ый поток должен читать соответствующее ему k-ое слово из сегмента данных в памяти, что проиллюстрировано на рисунке 2 (в качестве примера используется тип данных float).

Как видно из рисунка, допускается пропуск операции чтения потоком соответствующего адреса. Любые другие варианты не приводят к формированию транзакций.

При использовании текстур данные хранятся в той же глобальной памяти, только для ее выборки используются специальные текстурные блоки. Тектурные блоки обладают собственным кэшем, что дает заметный выигрыш в скорости. Тексель обозначает одну выборку данных из текстуры. Тексель может не совпадать с пикселям текстуры, а быть их интерполяцией. Недостатком текстур является не возможность их модификации. Эта особенность связана с изначальным предназначением видеoadаптера.

Для проведения тестов были выбраны следующие час-

то используемые типы данных: int, float, byte. На основе полученных результатов сделан вывод об эффективности применения различных типов памяти для алгоритмов обработки изображений.

Исследования проводились с использованием видеoadаптера NVidia GeForce 9800 GTE. Основные характеристики карты: тип шины памяти – GDDR3, ширина шины – 256 бит, частота памяти – 1800 MHz, частота шейдерного домена – 1375 MHz, частота текстурных блоков 550 MHz. Теоретическая пропускная способность шины памяти рассчитана по формуле, представленной в [2], составляет 53,6 Гб/с.

### 3 РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЯ

С целью определения оптимального типа данных для обработки изображения было определено время, необходимое для чтения всего объема данных. Для равномерного распределения нагрузки между мультипроцессорами выбран объем данных, эквивалентный миллиону пикс-

ТАБЛИЦА 1

#### РЕЗУЛЬТАТЫ ВРЕМЕНИ ЧТЕНИЯ

Тип данных	Время чтения с формированием транзакции, мкс	Время чтения без формирования транзакции, мкс	Время чтения текстур, мкс
int	105,2	927,6	36,8
float	103	920,4	37,5
byte	-	898,8	36,9

лей. Для минимизации влияния различных негативных факторов измерение осуществлялось 1000 раз и в качестве результата бралось среднее значение.

За основу взяты следующие типы данных: int, byte, float. При этом предполагалось, что 1 пиксель изображения соответствует указанному типу данных. Чтение осуществлялось: с формированием транзакций, без транзакций, с использованием текстур. Полученные результаты сведены в таблицу 1.

На основе полученных результатов рассчитана практическая пропускная способность шины по формуле:

$$P_1 = \frac{1}{1000 \cdot t_{\text{ср}}} . \quad (4.1)$$

Здесь  $P_1$  измеряется в Гпикселях/с. Результаты расчетов сведены в таблицу 2.

вом случае она составляет приблизительно 38 Гбайт/с во втором 26,4 Гбайт/с. Это значит что теоретически можно добиться скорости чтения 38 Гпикселей/с для типа byte.

Для того чтобы обойти ограничения невозможности формирования транзакции при использовании типа byte, предлагается объединить четыре соседних пикселя в 32-х битный блок и читать из памяти его, а не одиночный пиксель. Такое ядро было разработано и в ходе замера показало время исполнения 26,3 мкс, что соответствует 38 Гпикселей/с.

Использование упакованных пикселей требует их распаковки для дальнейшей обработки. Для этих целей используется разделяемая память. Исполнение любого алгоритма обработки изображения будет состоять из трех основных этапов: чтения данных из глобальной памяти с использованием упакованных пикселей и запись в разделяемую память, выполнение основного алгоритма, запись

ТАБЛИЦА 2

ПРОПУСКНАЯ СПОСОБНОСТЬ ШИНЫ			
Тип данных	Пропускная способность при формировании транзакций, Гпикселях/с	Пропускная способность без формирования транзакций, Гпикселях/с	Пропускная способность при чтении текстур, Гпикселях/с
int	9,5	1,1	27,1
float	9,7	1,1	26
byte	38	1,1	26,4

Как видно из таблиц 1 и 2, операция чтения без формирования транзакции имеет наихудшие показатели времени выполнения. При этом, независимо от применяемого типа данных, результат получается приблизительно одинаковый.

Использование транзакций позволяет повысить скорость чтения данных почти в 9 раз, для типов данных float и int. Формирование транзакций при использовании типа byte для обращения к памяти не происходит. Поэтому в таблице 1 отсутствуют данные в соответствующей ячейке.

Использование текстур позволяет добиться наилучшего результата по скорости чтения для типов данных float и int. Это прежде всего становится возможным за счет использования кеш памяти. Если перевести полученные значения скорости выборки в Гбайт/с, то скорость чтения в два раза превысит теоретическую пропускную способность памяти. Но вместе с тем скорость чтения при использовании типа данных byte осталась той же. Это объясняется тем, что для данного видеопроцессора скорость выборки текстур ограничена теоретически возможным порогом в 31 Гтекслей/с и для всех трех случаев мы в плотную подходим к этому пределу.

Существует еще один интересный факт, который становится очевидным, если сравнить скорости чтения из памяти в Гбайт/с для случая использования транзакций с типом int и использования текстур с типом byte. В первом

в глобальную память упакованных пикселей. Обработка четверки упакованных пикселей осуществляется последовательно тем же treadom, который их прочитал,. Это позволяет избежать конфликтов при обращении к разделяемой памяти.

Использование упакованных пикселей при записи также позволяет сформировать транзакцию. Формирование транзакций на запись также является очень важным. Использование текстур помогает не думать о формирование транзакции только при чтении. Поэтому использование упакованных пикселей позволит добиться максимальной скорости и при записи.

Для демонстрации преимущества использования упакованных пикселей был выбран классический алгоритм фильтрации изображений фильтром 3x3. Были написаны четыре реализации фильтра: реализация, которая формирует транзакции на основе упакованных пикселей как для чтения, так и для записи; реализация без транзакций для чтения и записи; реализация, которая использует текстуры для чтения и формирует транзакции на запись; реализация, которая использует текстуры для чтения и не формирует транзакции на запись. При использовании текстур для чтения из памяти разделяемая память для хранения обрабатываемого блока не использовалась. Тестирование проводилось на изображении 1000x1000 пикселей. Во всех случаях тип пикселя являлся байтом.

байт/с во  
ки можно  
на byte.  
можности  
типа byte,  
пеля в 32-  
биточный  
де замера  
ветствует

ет их рас-  
целей ис-  
юбого ал-  
ь из трех  
и памяти с

ГАБЛИЦА 2

ость при  
кселей /с

Обработка  
ется после-  
читал., Это  
ии к разде-

записи так-  
рмированием  
ажным. Ис-  
рмирование  
пользование  
аксимальной

ювания упа-  
ий алгоритм  
ли написаны  
орая форми-  
селей как для  
анзакций для  
ызует тексту-  
апись; реали-  
ния и не фор-  
зации текстур  
для хранения  
Тестирование  
пикселей. Во

Полученные результаты сведены в таблицу 3.

Как видно из таблицы, наилучшие результаты достигнуты с использованием разделяемой памяти и транзакций для чтения и для записи. Относительно не высокий результат для третьей реализации объясняется тем, что не

ТАБЛИЦА 3  
ВРЕМЯ ВЫПОЛНЕНИЯ ФИЛЬТРАЦИИ

Реализация	Время выполнения, мкс
Транзакции при чтении записи	294
Без транзакций	879
Текстуры при чтении, транзакции при записи	578
Текстуры при чтении, без транзакций при записи	764

используется разделяемая память. Скорость выборки тек-  
стур даже из кэша, значительно уступает скорости чтения  
из разделяемой памяти.

#### 4. ВЫВОДЫ

По результатам проведенных экспериментов можно сделать несколько выводов. Во-первых, в качестве типа для хранения пикселя нужно использовать байт, что при использовании упакованных пикселей для формирования транзакций и разделяемой памяти для хранения обрабатываемого блока позволяет добиться наивысшей производительности. Во-вторых, использование кэша текстурных блоков для замены разделяемой памяти является не эффективным из-за ограниченной скорости фильтрации. В третьих, при использовании текстур для обращения к глобальной памяти достигается меньшая пропускная способность для типа байт, чем при использовании упакованных пикселей и формировании транзакций.

#### ЛИТЕРАТУРА

- [1] NVIDIA CUDA™. Programming Guide. Version 2.3.1 // NVidia Corporation. – 2009.
- [2] NVIDIA CUDA C Programming. Best Practices Guide. CUDA Toolkit 2.3 // NVidia Corporation. – 2009.