ОПЫТ РАЗРАБОТКИ СЕРВИСА ДЛЯ HDR-ОБРАБОТКИ ВИДЕО-ДАННЫХ В GPGPU-ОБЛАКЕ

М.Н. Притула, М.А. Кривов, С.Г. Елизаров

Стремительное развитие архитектуры графических ускорителей, а в частности появление L1/L2-кэшей, увеличение числа регистров, добавление новых возможностей на уровне компилятора и программных интерфейсов, а также многократный рост пиковой производительности и энергоэффективности самих ускорителей, позволило ставить новые задачи, решение которых ещё пару лет назад казались если не невозможным, то экономически нецелесообразным в силу дороговизны аренды или покупки соответствующих вычислительных ресурсов.

Одной из таких задач является обработка видео-данных с помощью «тяжёлых» составных алгоритмов, в большинстве случаев являющихся некоторой композицией относительно простых, но специально настроенных фильтров, обеспечивающей существенное улучшение обрабатываемого видео-потока. Подобные алгоритмы находят применение в достаточно разных областях — от фильтрации видео-данных с камер внешнего наблюдения до наложения спецэффектов в фильмах и даже компьютерных играх.

Основной проблемой данных алгоритмов является большой объём вычислений, требуемых для фильтрации одного кадра, в результате чего временные затраты на обработку видео продолжительностью несколько минут могут составить несколько часов. Как следствие, до появления графических ускорителей приходилось либо упрощать сами алгоритмы, тем самым уменьшая время работы и понижая качество обработки, либо закупать дополнительные вычислительные мощности. Более того, достаточно часто обработка должна идти в режиме реального времени (25 кадров в секунду, или же не более 40 миллисекунд на кадр), что лишь обостряет озвученную проблему.

Авторами данной статьи была создана эффективная GPGPU-версия одного из подобных алгоритмов, заключающегося в повышения реалистичности видео-данных за счёт изменения локального контраста каждого кадра (так называемая HDR-обработка). Планируется оформить проект виде облачного интернет-сервиса, позволяющего проводить требуемую обработку практически в режиме реального времени, в то время как выполнение аналогичного преобразования доступными на данный момент решениями может потребовать время, превышающее продолжительность самого видео в десятки и даже сотни раз.

Описание алгоритма

Практически все современные фотокамеры могут сохранять, а современные мониторы — отображать не более 256 значений для каждого цветового канала, в то время как человеческий глаз может воспринимать намного большее количество оттенков. В результате этого при недостаточном освещении объекты, которые в реальности для глаза отлично различимы, на фотографиях смотрятся слишком затемнёнными и неразборчивыми. И наоборот — при избытке освещения появляются пересвеченные области, хотя с точки зрения восприятия человека на фотографируемом объекте они отсутствовали.

Для решения данной проблемы используются так называемые алгоритмы HDR-обработки фотографий, заключающиеся в трёх шагах. Сначала делаются несколько обычных LDR-фотографий (256 градаций для каждого цвета) одного и того же объекта, но с разной выдержкой. Далее производится «склейка» всех снимков в одну HDR-фотографию, содержащую порядка 2^{32} оттенков для каждого цвета. Последний шаг заключается в отображении полученного HDR-снимка обратно в LDR-фотографию, в которой на один цветовой канал опять отводится не более 256 значений и который уже можно отображать на обычных мониторах.

Легко заметить, что ключевым моментом является именно алгоритм отображения из HDR-фотографии в LDR-снимок, так как его задача заключается в выборе для каждого цвета из 2^{32} только 256 таких оттенков, которые обеспечивают максимально реалистичное восприятие фотографии. Данные алгоритмы известны под названием Tone Mapping, и в общем случае изменяют локальный контраст изображения, тем самым затемняя наиболее яркие объекты и подсвечивая наиболее тёмные. В результате полученная фотография с точки зрения человеческого глаза становится более правдоподобной, лучше соответствует реальности.

В предыдущей работе авторов [1] была предпринята попытка портирования одного популярного алгоритма Tone Mapping'a [2] на графические ускорители, в результате чего время обработки одной 12-мегапиксельной фотографии удалось уменьшить более чем в 60 раз - от примерно одной минуты до одной секунды. Однако созданная реализация крайне плохо подходила для работы с видео-данными, так как, во-первых, в отличие от фотографий практически не имеется возможности снять несколько полностью одинаковых видео, но с разным выдержками, а, во-вторых, исходный алгоритм крайне чувствительно реагировал на общую освещённость кадра, в результате чего при движении любого объекта яркость всех его соседей изменялась непредсказуемым образом.

Чтобы избежать озвученных проблем, был выбран проприетарный алгоритм, позволяющий даже по одному видео-потоку качественно построить его HDR-версию, а при переводе в LDR-видео сохранить общий уровень освещённости для соседних кадров. Идея выбранного алгоритма заключалась в разбиении исходного

изображения на группы объектов в зависимости от содержания кадра и последующем независимом наложении разнообразных простых фильтров на каждую группу объектов. Подобная схема позволяет достичь аналогичных результатов, что и при «честном» HDR-преобразовании, но она применима не только для фотографий, но и для видео-потоков, так как позволяет избежать вышеперечисленных проблем.

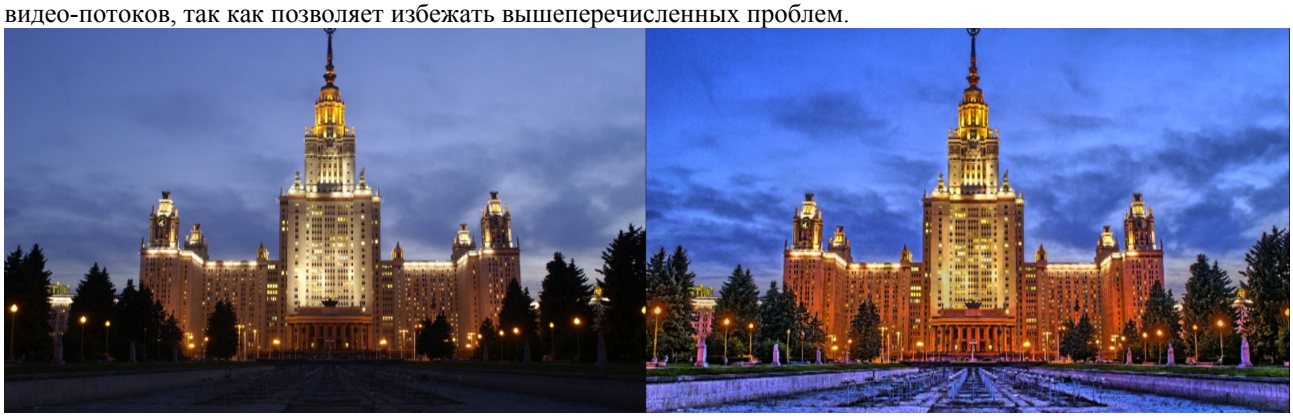

Рис. 1. Результат работы HDR-преобразования

Недостатком же данного подхода оказалась крайне низкая скорость работы. В алгоритме используется более чем 60 независимых фильтров, каждый из которых необходимо применять на одно или даже несколько промежуточных изображений, размер которых полностью соответствует размеру исходной фотографии. Как следствие, время обработки одной фотографии в несколько мегапикселей с помощью прототипа сервиса, реализованного через одну из стандартных библиотек фильтров, занимало порядка нескольких минут.

Разработка GPU-версии алгоритма

В целях повышения скорости работы было решено «с нуля» разработать отдельно оптимизированную версию для многоядерного процессора и версию для графических ускорителей от компании NVidia с использованием технологии NVidia CUDA, в дальнейшем которые будут использоваться одновременно с целью загрузить все доступные вычислительные устройства.

При разработке GPU-реализации было решено изначально ориентироваться только на архитектуру Fermi, что на данной задаче означало сознательный отказ от использования разделяемой памяти. Так, практически во всех фильтрах паттерн доступа к памяти был весьма специфический — в большинстве случаев для вычисления в новом изображении цвета пикселя в исходном требовалось порядка прочитать порядка 16-25 цветов точек-соседей. Более того, в ряде фильтров координаты требуемых соседей на момент компиляции были неизвестны, в результате чего при попытке «в ручную» закешировать некоторую область требовалось использование динамической разделяемой памяти, а также выполнение ряда достаточно сложных преобразований координат. В предыдущей архитектуре GT200, представителем которой является ускоритель NVidia Tesla C1060, подобные модификации действительно немного повысили бы скорость работы, в то время как на ускорителях поколения Fermi из-за подобных «улучшений» следует ожидать лишь замедления работы, так как встроенный аппаратный L1-кэш, уже расположенный в разделяемой памяти, прекрасно выполняет свои функции, а все программные преобразования координат лишь окажутся лишней вычислительной нагрузкой.

Другим типом оптимизации было расположение заранее известных ядер, применяемых в функции свёртки, в константной памяти, что позволило не только отказаться от излишних обращений к глобальной памяти, но и для ряда функций понизить количество требуемых регистров.

В результате проведённых оптимизаций на тестовой системе на базе GPU NVidia GeForce 580GTX (пиковая производительность — 1.5 ТФлопс) и CPU Intel Xeon E3-1230 (пиковая производительность — 100 ГФлопс) для GPU-версии удалось получить более чем 100-кратное ускорение относительно одного ядра центрального процессора. Зависимость скорости обработки от разрешения изображений приведена ниже на графике 2.

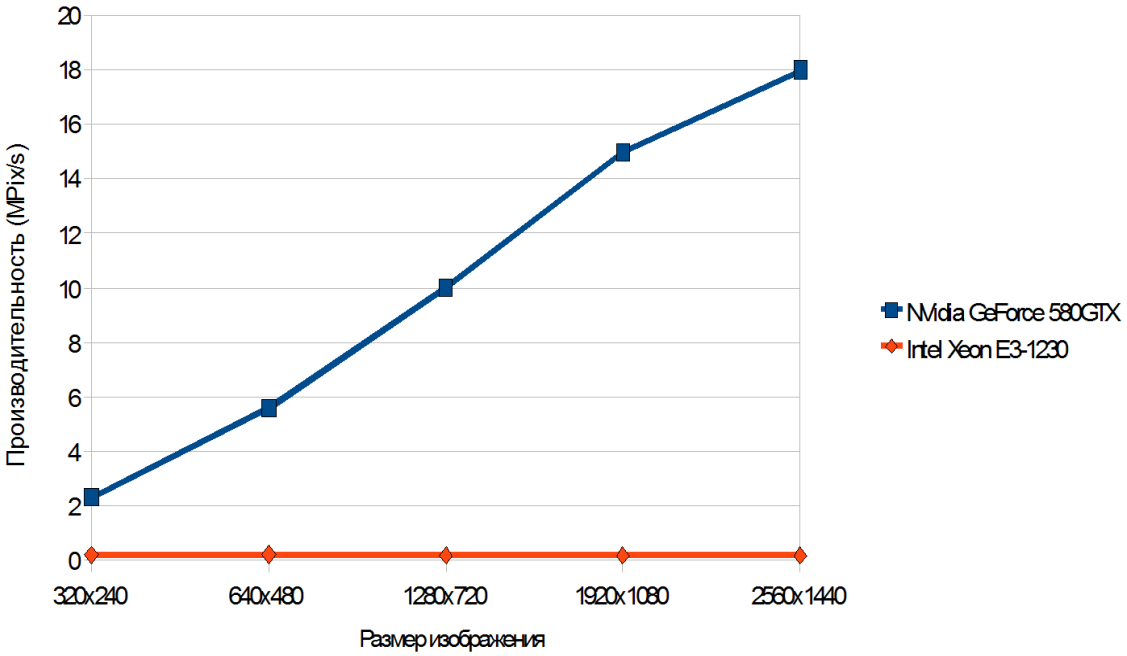

Рис. 2. Скорость обработки в зависимости от размера изображения

Легко заметить, что производительность графического ускорителя резко растёт с увеличением размера кадра (что объясняется лучшей загрузкой всех вычислительных блоков GPU), в то время как производительность центрального процессора остаётся инвариантной. Как следствие, при обработке видеоданных имеет смысл «склеивать» несколько штук или даже несколько десятков кадров в один с целью повышения общей производительности системы. К сожалению, в данном конкретном случае данная идея труднореализуема, так как в исходном алгоритме отсутствует возможность асинхронной обработки кадров — перед тем, как начать обрабатывать N-ый кадр, необходимо обработать N-1 кадр. Данную проблему удалось частично решить за счёт разбиения алгоритма на несколько этапов, в наиболее вычислительноёмких из которых кадры обрабатывались асинхронно, а в остальных — последовательно. В результате при обработке видеоданных производительность практически не зависит от размера кадра.

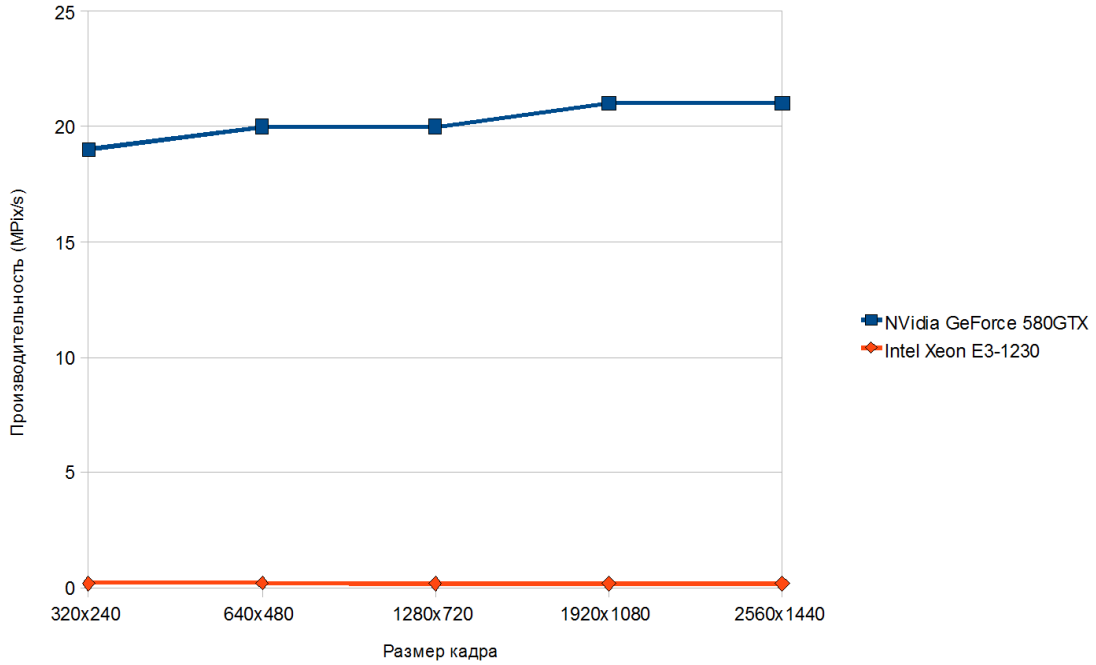

Рис. 3. Скорость обработки в зависимости от размера кадра

Стоит отметить, что порога для размера «склеенного» изображения, превышение которого больше не будет повышать суммарную производительность, для данной задачи не наблюдалось. Поэтому при выборе размера в первую очередь учитывались такие ограничения как объём доступной памяти (который на тестовом ускорителе GeForce 580 GTX равнялся 1.5 гигабайтам) и специфические особенности архитектуры CUDA,

определяющие максимальный размер решетки блоков. Опытным путём было установлено, что алгоритму для работы требуется не менее 45 байт на каждый пиксель обрабатываемого изображения (т. е. при 1.5 гигабайте видео-памяти пределом будет 33-мегапиксельное изображение), в то время как архитектура CUDA 2.0 дополнительно накладывает ограничение на обрабатываемое изображение в 17.3 мегапикселя.

Разработка облачного сервиса

Следующим этапом было доведение разработанной CUDA-реализации алгоритма до полноценного и устойчивого к ошибкам сервиса, позволяющего производить одновременную обработку произвольных данных на мини-кластере из нескольких разнородных узлов. Данный сервис предполагается использовать в рамках облачной модели вычислений, поэтому он должен удовлетворять следующим свойствам:

- *Динамическое подключение/отключение узлов.* Очевидно, что количество пользователей будет изменяться не только по мере развития проекта, но и в зависимости от времени суток. В случае аренды вычислительных мощностей, например, в облаке Amazon EC2 [3-4] учёт подобных колебаний в виде отказа от неиспользуемых ресурсов может существенно сократить затраты на аренду. Более того, даже при использовании собственного мини-кластера подобный механизм позволяет в моменты пиковых нагрузок подключать резервные (или используемые для других целей) ресурсы, тем самым снижая среднее время обработки одного видео-файла.
- *Равномерное распределение нагрузки.* В выбранной модели обработка всех поступающих запросов производится одновременно и независимо, в результате чего производительность каждого узла равномерно «размазывается» по всем видео-потокам, им обрабатываемым. Так как при этом все запросы являются неоднородными и имеют различную вычислительную сложность, было необходимо реализовать механизм, обеспечивающий балансировку нагрузки как на уровне мини-кластера, так и на уровне узла, имеющего несколько графических ускорителей и многоядерный центральный процессор, с целью обеспечения гарантированной минимальной скорости обработки каждого видео-файла.
- *Возможность использования неоднородных узлов.* Появление новых ускорителей поколения Kepler на данной задаче по приблизительным оценкам позволит повысить производительность узла с одним GPU с 25-30 МПикс/с до 40-60 МПикс/с. При этом имеет смысл не «выбрасывать» существующие ресурсы, а использовать их совместно с более новыми, в результате чего разрабатываемый сервис должен уметь распределять вычислительную нагрузку между узлами как с различным количество ускорителей, так и с разными поколениями GPU.
- *Универсальный интерфейс для запуска задач.* Последним требованием является механизм универсального запуска обработки видео-данных на узлах, которые в общем случае могут находиться как в разных подсетях, так и в физически удалённых ЦОД'ах. В частности, это накладывает ограничения на пересылку несжатых данных между узлами, а также требует по возможности абстрагироваться от топологии сети.
- *Отказоустойчивость*. Достаточно частым случаем является попытка обработать некорректный файл, что может быть вызвано как ошибкой при его передаче по сети, так и новым типом сжатия (кодеком), ещё не поддерживаемым разрабатываемым сервисом. Подобные ошибки не должны приводить к «поломке» сервиса, также как и сказываться на обработке других файлов. Более того, сервис должен быть устойчив к появлению проблем на отдельном вычислителе и даже к выходу из строя отдельного узла, по возможности минимизировав количество «запорченных» видео-потоков.

Легко заметить, что для реализации всех перечисленных выше требований совершенно не подходит стандарт MPI, в результате чего для осуществления взаимодействия между узлами было решено использовать такие стандартные средства операционной системы Linux, как файловая система NFS и протокол для удалённого запуска команд SSH, а также ряд «самописных» утилит и скриптов для реализации балансировщика загрузки и прочих частей сервиса.

В результате запуск на обработку любого файла реализован в соответствии со следующей схемой:

- Обрабатываемый файл начинает загружаться в общую NFS-директорию.
- Клиент через универсальный скрипт производится запуск задания на обработку загружаемой фотографии или видео-файла. Данный скрипт считывает из конфигурационных файлов текущую топологию облака и параметры требуемой обработки, запрашивает у доступных узлов метрики загруженности и производительности, после чего по SSH запускает на выбранном узле следующий скрипт для непосредственной обработки.
- На узле-обработчике запущенный скрипт опознаёт тип запроса (обработка фотографии или видео, тип формата и т.д.), после чего через библиотеку FFMPEG или вспомогательный загрузчик начинает декодировать исходный файл, укрупнять кадры в соответствии с

возможностями данного узла и через сокеты отправлять их в процесс-обработчик, независимо получая от него уже обработанные кадры и кодируя их в требуемый формат.

- Процесс-обработчик получает готовые порции несжатых изображений в формате R8G8B8, после чего выбирает наиболее подходящий из доступных вычислителей (для небольших кадров — центральный процессор, для больших — графический ускоритель) и отправляет его на непосредственную обработку, независимо обновляя метрики загруженности и производительности.
- При окончании обработки начальный скрипт завершает свою работу, возвращая обработанный файл или код ошибки.

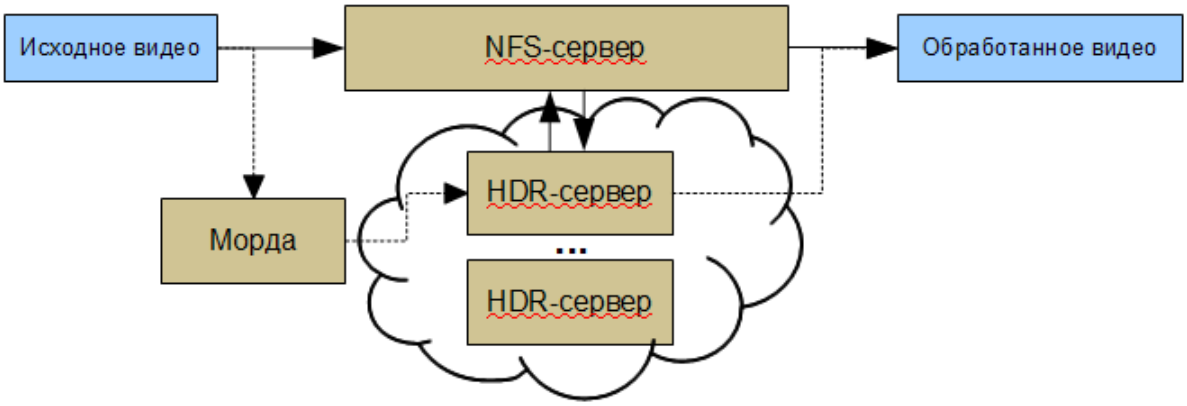

Рис. 4. Схема работы облачного сервиса

Стоит отдельно отметить, что в данной модели под клиентом понимается не конкретный пользователь сервиса, а некоторая программная сущность, имеющая права на запуск обработки. В частности, клиентом может быть как web-сервер, предоставляющий соответствующую функциональность непосредственным пользователям через браузер, так и независимое ПО для HDR-обработки видео-файлов.

Заключение

Результатом выполненных авторами работ стала GPGPU-реализация алгоритма HDR-обработки фотографий и видео-потоков, продемонстрировавшая на тестовой системе 100-кратное ускорение относительно последовательной версии, что практически позволяет проводить конвертирование в режиме реального времени. Отдельно была разработана начальная реализация интернет-сервиса для выполнения соответствующего преобразования на GPU-кластере в полном соответствии с парадигмой облачных вычислений.

На момент написания статьи рассматриваемый облачный сервис для HDR-обработки видео-данных был успешно протестирован на системе их нескольких узлов, каждый из которых оснащён несколькими графическими ускорителями. Текущим направлением работ является проведение нагрузочного тестирования сервиса как на собственном мини-кластере, так и в облаке Amazon EC2 с целью оценки максимальной производительности и масштабируемости, а также себестоимости проведения HDR-обработки для видео различных форматов.

ЛИТЕРАТУРА:

1. Кривов М.А., Притула М.Н., Елизаров С.Г. Опыт портирования среды для HDR-обработки изображений на GPU и APU // Параллельные вычислительные технологии (ПаВТ’2012): труды международной научной конференции (Новосибирск, 26 – 30 марта 2012 г.). Челябинск: Издательский центр ЮУрГУ, 2012, с. 537–546.
2. Mantiuk R., Myszkowski K., Seidel H.-P., A Perceptual Framework for Contrast Processing of High Dynamic Range Images, ACM, 2006.
3. Кривов М.А. Научные облака в рублях // Журнал «Суперкомпьютеры», Зима 2011, с. 51-53.
4. The Magellan Report on Cloud Computing for Science, December, 2011