Использование средств nVidia CUDA для эффективной реализации алгоритмов построения карт диспаратности

А.Н. Волкович

Объединенный институт проблем информатики НАН Беларуси, Беларусь, Минск

Введение

Построение объемной модели на основе стереоизображений традиционно является одним из наиболее актуальных направлений в развитии компьютерного зрения. Последние исследования в этой области значительно продвинули область знания в вопросах качества и точности построений. Однако, на современном этапе исследований требования к производительности значительно превышают возможности элементной базы – алгоритмы стереовосстановления обычно требуют от нескольких секунд до нескольких минут машинного времени, для построения карты диспаратности.

На современном этапе развитие современных графических ускорителей или Graphic Processor Unit (в дальнейшем GPU) привело к появлению устройств с программируемым конвейером. В современных системах организована возможность программируемой обработки вершин. Столь коренное изменение архитектуры GPU привело к возможности использования GPU не только для целей компьютерной графики, но и для задач, которые раннее решались на CPU, таким образом, появилась технология вычислений общего назначения на графическом ускорителе или General Purpose Computation On Graphic Processor Unit (в дальнейшем просто GPGPU).

Практика показывает, что правильный перенос алгоритма на платформу графического процессора зачастую позволяет в 5-20 раз увеличить производительность по сравнению с выполнением на современном процессоре общего назначения. Некоторые алгоритмы, особенно удачно перерабатываемые для архитектуры графического процессора, позволяют добиться стократного роста производительности.

Отсюда можно сделать вывод о том, что перенос наиболее трудоемких частей алгоритма на средства GPGPU может дать значительный прирост эффективности их выполнения

Адаптация алгоритмов построения карт диспаратности

Специализированная архитектура графических процессоров не подходит для реализации произвольного алгоритма. Многие алгоритмы ориентированы на последовательную обработку. Вместе с тем многие важные задачи требуют значительных вычислительных ресурсов и хорошо укладываются в характерную для графического процессора схему интенсивной многоядерной арифметической обработки.

Как известно для реализации алгоритма на параллельном вычислителе необходимо произвести его адаптацию, для чего следует предварительно провести разбор принципов выполнения алгоритмов.

Простейшим алгоритмом построения карт диспаратности является – блочный. Данный алгоритм относится к классу локальных (обрабатывающих некоторую часть изображения), что изначально дает возможность судить о его приспособленности к распараллеливанию.

Блочный алгоритм определяет диспаратность, сравнивая небольшой регион (блок) вокруг точки первого изображения с последовательностью таких же регионов на втором изображении в некоторой области поиска. В качестве меры сходства блоков изображений часто используются сумма квадратов разностей интенсивностей

$$\sum_{u,v} (I_1(u,v) - I_2(u+d,v))^2 \qquad (1)$$

сумма абсолютных разностей

$$\sum_{u,v} \left| I_1(u,v) - I_2(u+d,v) \right| \tag{2}$$

и нормированная кросс-корреляция. В приведенных формулах используются следующие обозначения: *Ik(u,v)* – интенсивность пикселя *(u,v)* на *k*-м изображении, *d* – оцениваемое значение диспаратности, суммирование ведётся по некоторому прямоугольному окну.

Таким образом оптимальным рассматривается возможность непосредственного деления изображения на блоки и их параллельной обработки на вычислительных ядрах GPU. В свою очередь этап суммирования формул реализуемый обычно в виде циклов так же обладает не плохим коэффициентом распараллеливания, что может быть использовано средой CUDA с целью повышения эффективности в случае наличия незагруженных вычислительных узлов. Данная функция реализована в среде разработки как средство автоматической балансировки загрузки.

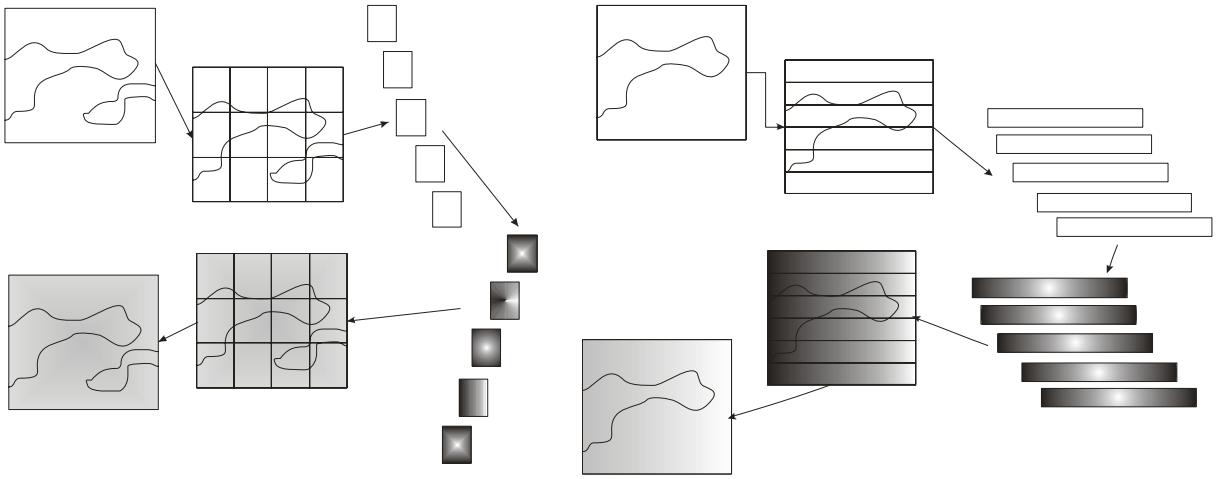

Рисунок 1. Условные схемы выполнения алгоритмов.

Рассмотрим далее алгоритм, использующий метод динамического программирования для построения плотной карты диспаратности. Этот алгоритм работает с парами соответствующих строк на изображении, рассматривая их независимо от остальных строк. Поскольку этот алгоритм обрабатывает всю строку целиком, то он, как правило, позволяет получить лучшие по сравнению с блочным алгоритмом результаты для областей со слабовыраженной текстурой.

В данном методе вводится понятие изображения пространства диспаратности *(DSI)* – явное представление пространства, в котором ведётся поиск соответствий. Обозначим через *IR(x, i)* и *IL(x, i)* интенсивность точки с координатами *(x, i)* на правом и левом изображениях соответственно, а через *d* – диспаратность. Тогда в общем виде DSI можно представить функцией

$$DSI_i^L(x, d) = |I_L(x, i) - I_R(x-d, i)|, \tag{3}$$

где $-d_{max} \le d \le d_{max}$, $0 \le x+d < N$, N – ширина изображения..

Здесь очевидна возможность разделения изображения на блоки обрабатываемых строк с дальнейшим вычислением диспаратности на ядрах GPU.

В принципе аналогичный подход в декомпозиции данных может быть использован и для метода основанного на задаче о назначениях, т.к. в нем также производится обработка строк изображения.

Одним их глобальных методов построения карт диспаратности относится метод максимального потока, который строит по отображению пространства диспаратности граф, в котором затем находится максимальный поток, используемый для построения плотного отображения диспаратности.

Идея данного метода состоит в расширении метода динамического программирования путем добавления межстрочных связей. При этом каждая внутренняя точка становится шестисвязной и вместо пути минимальной стоимости через отображение пространства диспаратности строится поверхность минимальной стоимости.

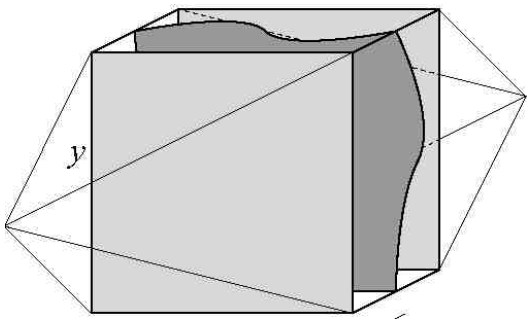

Рисунок 2. Граф, построенный по методу максимального потока.

Каждая внутренняя вершина графа является шестисвязной. Все ребра, за исключением ребер источника и стока, можно разделить на две группы: ребра диспаратности, соединяющие вершины с разными значениями диспаратности, и ребра затенения, соединяющие вершины с одинаковой диспаратностью.

Помимо большой вычислительной сложности самого метода здесь производится обработка изображения целиком, что теоретически говорит о не возможности распараллеливания данного подхода. Однако, здесь возможно произвести декомпозицию по принципу создания группы линейных задач, т.е. произвести разделение исходных изображений на несколько достаточно крупных, но при этом быстро обрабатываемых секторов. Далее, данные сектора следует рассматривать как самостоятельные изображения, обрабатываемые, затем, линейно в вычислительных узлах. Таким образом, будет получено несколько завершенных карт диспаратности которые потом могут быть обработаны на CPU и объединены в одну.

Рассматривая приведенные выше подходы и модель вычислений можно предложить создание программной библиотеки функционирующей по принципу т.н. «черного ящика». Используя данный подход на вход алгоритма подаются указатели на исходные изображения и массив настроечных данных, а так же атрибуты указывающие на выбор обрабатывающего алгоритма. В свою очередь на выходе алгоритма будет получен указатель на рассчитанную карту диспаратности. Кроме того необходимо позаботиться о реализации средств мониторинга выполнения алгоритма, для создания возможности реализации интерфейсно-диалоговых средств отображающих состояние вычислительного процесса.

Реализация прототипной системы

В целях оптимальной реализации алгоритма необходимо разобрать архитектуру системы.

Вычислительная архитектура CUDA основана на концепции одна команда на множество данных (Single Instruction Multiple Data, SIMD). Концепция SIMD подразу-

мевает, что одна инструкция позволяет одновременно обработать множество данных. Среда CUDA представляет собой, своего рода, «черный ящик» на входе которого имеются данные и алгоритм, а на выходе – полученный результат. Система практически все функции коммуникации и управления берет на себя. Однако существует несколько приемов оптимизации программ для их более быстрого исполнения.

Одним из основных шагов является минимизация перемещений данных *Host* <=> *Device* (компьютер ⇔ ускоритель). Поскольку скорость обмена с памятью компьютера через системную шину намного ниже чем скорость в пределах GPU следует как можно реже передавать промежуточные результаты на *host* для обработки с помощью CPU. В идеальном случае математическая основную часть задачи должна быть реализована и выполняться только на GPU, оставляя CPU лишь управляющие задачи.

Далее следует произвести правильный выбор типа используемой памяти. Рекомендуется размещать данные в текстурную или константную память, если все задачи одного блока обращаются к одному и тому же участку памяти или к близко расположенным участкам. Следует использовать глобальную память в сочетании с разделяемой памятью, если все задачи обращаются бессистемно к разным, далеко расположенным друг от друга участкам памяти.

Наиболее оптимальным является минимизирование использования регистров и разделяемой памяти т.к. чем больше ядро использует регистров или разделяемой памяти, тем меньше потоков одновременно могут выполняться на мультипроцессоре, в связи с ограниченностью ресурсов мультипроцессора. Поэтому небольшое увеличение занятости регистров или разделяемой памяти может приводить, в некоторых случаях, к падению производительности в два раза из-за сокращения одновременно исполняемых на мультипроцессоре потоков.

Таким образом анализируя задачу построения плотной карты диспаратности видится оптимальным разделение частей программы между GPU и CPU следующим образом:

- CPU: инициализация программы, загрузка данных из файлов, запуск вычислительной среды, запись результатов в файл;
- GPU: инициализация CUDA, загрузка данных в CUDA, вычисление диспаратности, возврат результатов.

Распределение задачи таким образом позволяет исключить многократные обмены данными между оперативной памятью компьютера и памятью графического ускорителя, что могло значительно замедлить скорость выполнения алгоритма.

Инициализация приложения производится компилятором автоматически. Во время выполнения инициализации производится выделение оперативной памяти компьютера и производится запуск необходимых драйверов и библиотек необходимых для выполнения программы.

Следующим этапом выполнения является получение исходных данных. Данная задача может быть решена несколькими способами в зависимости от конкретной реализации системы: чтение из файлов, получение из памяти устройств. Поскольку данная задача является хорошо проработанной на практике видится возможным использование сторонних библиотек для получения данных из файлов различных графических форматов или из фото- (видео-) устройств. При отладке алгоритма наиболее оптимальным видится использование графических файлов как контейнера исходных данных, как наиболее стабильного и простого в реализации.

Следующим этапом выполнения приложения является построение плотной карты диспаратности. Исходя из специфики архитектуры среды необходимо произвести дробление задачи таким образом, чтобы вычисления производились параллельно. Оптимальным является деление задачи согласно сетки блоков графического процессора.

Таким образом оптимальным рассматривается возможность непосредственного деления изображения на блоки и их параллельной обработки на вычислительных ядрах GPU. В свою очередь этап суммирования формул реализуемый обычно в виде циклов так же обладает не плохим коэффициентом распараллеливания, что может быть использовано средой CUDA с целью повышения эффективности в случае наличия незагруженных вычислительных узлов. Данная функция реализована в среде разработки как средство автоматической балансировки загрузки.

После проведения всех исследований с целью оптимизации получен достаточно эффективный процесс построения плотной карты диспаратности с использованием средств NVIDIA CUDA как высокопроизводительного вычислителя.

Тестирование системы

Во время реализации системы проводилось тестирование алгоритма, для чего использовалась стереопара, представленная на рис.3.

Рисунок 3. Тестовая стереопара и карта диспаратности.

Изображения данной стереопары обладают размерами 450/375 точек со стандартным бифокальным смещением приближенным к человеческому зрению. На этапе отладки были произведены расчеты 8-уровневой диспаратности и достигнута средняя скорость исполнения в 0.008-0.012 секунды, что значительно превосходит эффективность линейного исполнения алгоритма.

Выводы

В результате можно говорить о том, что представленная компанией NVIDIA программно-аппаратная архитектура для расчётов на графических процессорах CUDA хорошо подходит для решения широкого круга задач с высоким параллелизмом. CUDA работает на большом количестве продуктов NVIDIA, и улучшает модель программирования GPU, значительно упрощая её и добавляя большое количество возможностей, таких как разделяемая память, возможность синхронизации потоков, вычисления с двойной точностью и целочисленные операции.

Задача вычисления плотной карты диспаратности достигла приемлемой скорости для ее дальнейшего развития в систему восстановления объема по видеопоследовательности в реальном времени.