УДК 004.921

CUDA РЕАЛИЗАЦИЯ АЛГОРИТМОВ ВОКСЕЛЬНОГО ПРЕДСТАВЛЕНИЯ ОТРЕЗКОВ ПРЯМЫХ ДЛЯ ОБЪЕМНЫХ 3D ДИСПЛЕЕВ

2

Хлопов Д.И., Авксентьева О.А.
Донецкий национальный технический университет

Рассматривается прогрессивная технология NVIDIA CUDA, предназначенная для реализации высокопроизводительных алгоритмов. Предложены методы использования данной технологии для реализации задачи генерации отрезков прямых для трехмерных дисплеев. Представлены результаты компьютерных экспериментов и оценки ошибок и времени реализации алгоритма генерации с помощью технологии CUDA.

Актуальность

Одним из наиболее эффективных средств информативности человека в различных областях науки и техники является визуализация на базе 3-х мерных дисплеев. К таким системам относятся параллаксные, голографические и объемные технологии [1]. Построение 3D дисплеев на базе объемных технологий требует разработки как аппаратуры дисплея, так и его программного обеспечения, причем важную роль при этом играет требование реального времени. Одним из способов увеличения производительности графических устройств является применение технологии CUDA. CUDA (Compute Unified Device Architecture) - это технология от компании NVIDIA, предназначенная для разработки приложений для массивно-параллельных вычислительных устройств (в первую очередь для GPU (графических процессоров) начиная с серии G80) [2]. Технология CUDA работает на видеокартах NVIDIA начиная с 8400GS и выше. Видеокарты разных серий имеют различные вычислительные возможности. В

целом, если в видеокарте, например, 128 SP(Streaming Processor) — это значит, что она содержит 8 SIMD MP (multiprocessor), каждый из которых делает одновременно 16 операций. Для написания программ с использованием технологии CUDA необходима одна из вышеперечисленных видеокарт, а так же распространяемая свободно библиотека соответствующей версии [3].

Вычислительная модель GPU

Внутренняя модель Nvidia GPU – ключевой момент в понимании GPGPU с использованием CUDA [4]. Верхний уровень ядра GPU состоит из блоков, которые группируются в сетку или грид (grid) размерностью N1 * N2 * N3. В большинстве видеокарт максимальное количество блоков составляет 65536. При этом N1 и N2 может быть равно 65536, а значение N3 равно 1. Любой блок в свою очередь состоит из потоков (threads), которые являются непосредственными исполнителями вычислений. Потоки в блоке сформированы в виде трехмерного массива. Максимальное количество потоков в блоке зависит от типа обрабатываемых данных. Для чисел с одинарной точностью в блок может содержать 512 потоков, а для чисел с двойной точностью – 256. При вычислениях у каждого потока есть свой уникальный идентификатор (трехмерный вектор, имеющий три измерения x, y и z). По x и y максимальное значение составляет 512(256), а по z – 64. При использовании GPU можно задействовать сетку необходимого размера и сконфигурировать блоки под нужды решаемой задачи [4].

Модель памяти в технологии CUDA

При использовании GPU разработчику доступно несколько видов памяти: регистровая, локальная, глобальная, разделяемая, константная и текстурная память. Каждая из этих типов памяти имеет определенное назначение, которое обуславливается её техническими характеристиками и параметрами (скорость работы, уровень доступа на чтение и запись). Иерархия типов памяти представлена на рисунке 1.

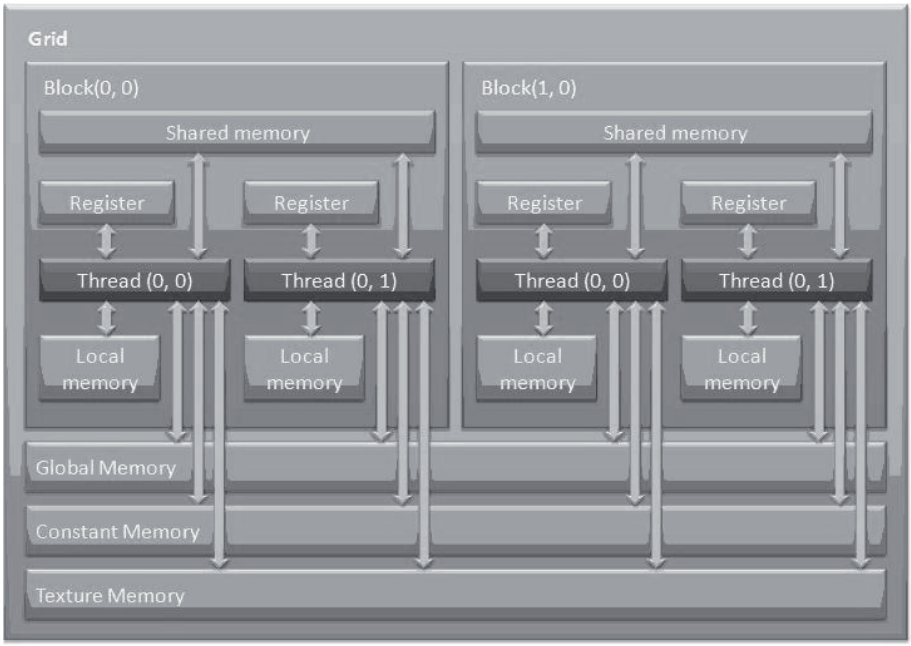

Рисунок 1 – Модель памяти GPU

Регистровая память (register) является самой быстрой из всех видов. Для того чтобы определить количество регистров, доступных одной нити GPU, необходимо разделить общее число регистров на произведение количества потоков в блоке и количества блоков в гриде. Все регистры GPU 32 разрядные. В CUDA нет явных способов использования регистровой памяти, всю работу по размещению данных в регистрах берет на себя компилятор [4]. Локальная память (local memory) может быть использована компилятором при большом количестве локальных переменных в какой-либо функции. По скоростным характеристикам локальная память значительно медленнее, чем регистровая. В документации от NVIDIA рекомендуется использовать локальную память только в самых необходимых случаях [4]. Глобальная память (global memory) – самый медленный тип памяти. Глобальная память в основном служит для хранения больших объемов данных, поступивших с хоста. В алгоритмах, тре-

бующих высокой производительности, количество операций с глобальной памятью рекомендуется свести к минимуму. Разделяемая память (shared memory) относится к быстрому типу памяти. Разделяемую память рекомендуется использовать для минимизации обращения к глобальной памяти, а так же для хранения локальных переменных функций. Адресация разделяемой памяти между потоками одинакова в пределах одного блока, что может быть использовано для обмена данными между ними в пределах одного блока. Константная память (constant memory) является достаточно быстрой. Если необходимо использовать массив в константной памяти, то его размер необходимо указать заранее, так как динамическое выделение в отличие от глобальной памяти в константной не поддерживается. Текстурная память (texture memory), как и следует из названия, предназначена главным образом для работы с текстурами.

Оптимизация приложений

При оптимизации приложения для CUDA используются три основные стратегии:

- увеличение параллельного выполнения;
- оптимизация размера сетки;
- оптимизация использования памяти для достижения максимальной пропускной способности [2].

Увеличение параллельности программы начинается с того, что алгоритм структурируется таким образом, чтобы как можно больше вычислений выполнялось параллельно. После оптимизации алгоритма, его необходимо отобразить в соответствии с архитектурой и оборудованием NVIDIA CUDA наиболее эффективным способом. Эта проблема решается путем выбора оптимальных для данной задачи размера сетки и размера блока. Язык C в технологии CUDA расширяет стандартный, позволяя программисту определять свои функции, называемые ядром. Ядро – это функция, которая выполняется N раз различными N потоками. При вызове функции необходимо сконфигурировать количество потоков, которые будут выполняться. Данная переменная является 3х мерным массивом,

и адресация потоков может быть одномерной двумерной или трехмерной. Количество потоков конфигурируется в зависимости от выполняемой задачи. Для выполнения параллельных операций над массивами, структурами, а так же текстурами необходимо, чтобы данные находились в памяти видеокарты, и обрабатывались с помощью ядра. Вне функции ядра доступа к отдельному элементу такого массива нет, возможно только копирование уже инициализированных данных с хоста в память видеокарты. При вызове функции ядра необходимо сконфигурировать количество блоков и потоков. Эти показатели сильно влияют на скорость выполнения вычислений. Количество блоков и потоков не определяют количество и порядок использования процессоров, эту функцию берет на себя компилятор. Но, тем не менее, рекомендуется, чтобы количество блоков было не менее количества мультипроцессоров в видеокарте. Разрабатываемое приложение должно обрабатывать данные большого размера, поэтому использование разделяемой, регистровой и константной памяти ограничено. Все данные хранятся в глобальной памяти, и при вычислениях происходит неоднократное чтение и запись. Для увеличения скорости работы приложения целесообразно, где возможно, заменить использование глобальной памяти на регистровую.

Использование технологии CUDA для реализации задачи генерации отрезков прямых для трехмерных дисплеев

Задача генерации отрезков прямых 3D дисплеев и последовательные алгоритмы ее решения рассмотрены в [5]. Входными данными рассматриваемой задачи является множество координат, которые являются начальными и конечными точками множества генерируемых отрезков в 3D пространстве. Выходными данными является множество сгенерированных векселей.. Сначала данные читаются из файла. Далее необходимо выделить память для хранения массивов с данными в глобальной памяти видеокарты. Затем данные, прочитанные из файла, копируются в глобальную память видеокарты. В соответствии с полученным порядком

входных значений параллельно генерируются все отрезки по алгоритму из [5], рис. 2. При запуске ядра все потоки выполняются асинхронно. Поэтому после параллельного выполнения ядра необходимо их синхронизировать. Данная программа использует тысячу блоков (для каждого отрезка прямой) и в каждом блоке по семь потоков, которые соответствуют вокселям-претендентам в растровом разложении отрезка. Работа ядра заключается в следующем. Ядро запускает сразу все используемые блоки. Во всех блоках параллельно для всех прямых вычисляются направляющие векторы и матрицы для вокселей- претендентов. Далее в каждом блоке генерируются параллельно все прямые, причем на каждом шаге генерации вокселей параллельно вычисляются расстояния от прямых до соответствующих вокселей-претендентов. Это достигается благодаря использованию нитей внутри каждого блока, вычисления в которых тоже осуществляются параллельно.

Экспериментальное исследование предложенного метода заключалось в генерации тысячи различных отрезков в пространстве 1 мегавоксель. Обобщенные результаты экспериментов приведены в таблице 1.

Анализ результатов экспериментов показывает, что использование технологии NVIDIA CUDA существенно повышает быстро-

Рисунок 2 – Алгоритм работы программы

действие (примерно в 16 раз), что может быть использовано для генерации сложных сцен с учетом требований реального времени.

Таблица 1 – Результаты генерации тысячи отрезков

	Кол-во вокселей	Время генерации тысячи отрезков (млсек)	Производительность (тыс. вокс. в сек)	Относительное ускорение
Выполнение на CPU	553869	3099	179	1
Выполнение на GPU	553869	192	2 885	16

Литература

[1] Favalora G.E. et al., “100 million-voxel volumetric display”, in Proc. SPIE Cockpit-Displays IX: Displays for Defense Appl, 2002, vol. 4712.

[2] Nvidia Cuda Programming Guide 2.2. Электронный ресурс. Режим лоступа: http://developer.download.nvidia.com/compute/cuda/2_2/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.2.pdf

[3] CUDA Toolkit 3.2 RC 2 (October 2010) Электронный ресурс. Режим доступа: http://developer.nvidia.com/object/cuda_3_2_toolkit_rc.html

[4] CUDA: Как работает GPU. Компьютерный блог habrahabr. Электронный ресурс. Режим доступа: habrahabr.ru/blogs/CUDA/54707

[5] Башков Е.А., Авксентьева О.А., Аль-Орайкат Анас М. К построению генератора графических примитивов для трехмерных дисплеев. В сб. Наукові праці Донецького національного технічного університету, серія «Проблеми моделювання та автоматизації проектування динамічних систем». Вип. 7 (150). – Донецьк, ДонНТУ. – 2008. - С. 203-214.