Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра ПМ и К

Курсовой проект

на тему: «Оценка производительности реализаций SAXPY с помощью различных API»

Выполнил:
студент гр. МГ-165
Мосин И.А.

Проверил:
д.ф-м.н. профессор
Малков Е.А.

Новосибирск 2017 г.

Оглавление

Введение

На сегодняшний день существует множество задач, которые требуют высоких вычислительных затрат и имеют параллельные алгоритмы решения. При этом современные графические процессоры (GPU — Graphics Processing Units) обладают параллельной архитектурой и высокой производительностью, поэтому могут эффективно применяться для решения подобных задач. Однако достигнуть высокой скорости решения задач можно лишь после значительной оптимизации алгоритма под параллельную архитектуру и конкретный GPU. Научное направление по созданию, реализации и оптимизации различных алгоритмов, напрямую не связанных с компьютерной графикой, получило названием GPGPU (General Purpose computations on Graphics Processing Units — вычисления общего назначения на графических процессорах).

Компания NVIDIA предложила свое решение, предназначенное для разработки приложений для массивно-параллельных вычислительных устройств, и назвала его CUDA (Compute Unified Device Architecture — унифицированная архитектура компьютерных вычислений). CUDA ориентирована на графические процессоры NVIDIA GeForce 8-й серии и новее, а также специализированные процессоры NVIDIA Tesla. Технология активно развивается и поддерживается разработчиками ПО. Для CUDA разработаны вспомогательные библиотеки: CUBLAS, CUDA Performance Primitives, thrust. В состав пакета CUDA Toolkit входит компилятор PTX (Parallel Thread eXecution), позволяющий работать с ассемблерным кодом программ, выполняемым на графическом процессоре.

Целью данной курсовой работы является оценка производительности различных методологий реализации программ для графических процессоров. Для этого реализовано решение простейшей задачи линейной алгебры SAXPY (Scalar Alpha X Plus Y) — задачи скалярного умножения и векторного сложения [1]. Она заключается в вычислении результата двух векторных операций: умножения на скаляр и сложения векторов. Необходимо вычислить новое значение вектора y по формуле

$$y = a \cdot x + y,$$

где a — число, x и y - векторы большой размерности.

Скалярное умножение и векторного сложения реализовывалось при помощи библиотеки thrust, CUBLAS, а также без использования библиотечных реализаций.

Описание технологии CUDA

Программы для CUDA (соответствующие файлы обычно имеют расширение .cu) пишутся на расширении языка C и компилируются при

помощи компилятора nvcc, предоставляемого бесплатно компанией NVIDIA через Интернет.

Ниже перечислены основные расширения языка C, которые введены в CUDA:

1) спецификаторы функций, показывающих, где будет выполняться функция (host или device) и откуда она может быть вызвана;
2) спецификаторы переменных, задающие тип памяти, используемый для данной переменной;
3) директива, предназначенная для запуска ядра программы на GPU, задающая как данные, так и иерархию потоков выполнения;
4) встроенные переменные, содержащие информацию о выполняемом потоке;
5) библиотека, включающая в себя дополнительные типы данных и функции работы с памятью, управления устройством и другое.

Функция решения задачи SAXPY на CUDA входит в библиотеку CUBLAS. Однако мы представляем собственную реализацию для оценки возможностей разработки с использованием инструмента CUDA.

Обзор CUBLAS

CUBLAS — реализация интерфейса программирования приложений для создания библиотек, выполняющих основные операции линейной алгебры BLAS (*Basic Linear Algebra Subprograms*) для CUDA. Он позволяет получить доступ к вычислительным ресурсам графических процессоров NVIDIA. Библиотека является самодостаточной на уровне API, то есть, прямого взаимодействия с драйвером CUDA не происходит. CUBLAS прикрепляется к одному GPU и автоматически не распараллеливается между несколькими GPU.

Основные функции библиотеки CUBLAS: создание матриц и векторных объектов в пространстве памяти GPU, заполнение их данными, вызов последовательных функций CUBLAS, и загрузка результатов из области памяти GPU обратно к хосту. Чтобы достичь этого, CUBLAS предоставляет вспомогательные функции для создания и уничтожения объектов в памяти GPU, и для записи данных и извлечения информации из этих объектов.

Из-за того, что основные функции CUBLAS (в отличие от вспомогательных функций), не возвращают статус ошибки напрямую (из соображений совместимости с существующими библиотеками BLAS), CUBLAS предоставляет отдельную функцию, чтобы помочь в отладке, которая возвращает последнюю записанную ошибку.

Описание библиотеки thrust

Одной из самых красивых и гибких библиотек для CUDA является библиотека thrust. Основным отличием данной библиотеки от других

распространенных библиотек для CUDA является то, что thrust - это библиотека, основанная на использовании шаблонов (template) языка C++.

Все классы и функции в этой библиотеки - шаблонные, все, что вам нужно для работы с этой библиотекой - это подключить соответствующие заголовочные файлы - в thrust нет никаких стандартных библиотечных файлов (.lib, .a, .dll, .so).

Ряд понятий и подходов thrust заимствовала из STL. thrust предоставляет в ваше распоряжение набор различных параллельных примитивов, таких как различные преобразования, сортировка, операции reduce и scan. Применяя thrust, многие действия могут быть записаны просто и понятно с использованием минимального объема кода. Все последнии версии CUDA включают в себя thrust, так что для работы с thrust никаких дополнительных установок вам не понадобится.

Все, что вводит thrust, помещается в пространстве имен thrust. В частности, это удобно тем, что позволяет избежать путаницы с рядом операций, имеющих те же самые имена (и параметры), что и операции, вводимые в STL (например, sort). Обычно, если thrust вводит операцию, уже существующую в STL, то для нее используется то же имя и аналогичный синтаксис.

Реализация

Ниже приведен листинг разных реализаций SAXPY.

SAXPY наивная реализация

```
__global__ void saxpy(int n, float a, float *x, float *y)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n)
    {
        y[i] = a * x[i] + y[i];
    }
}
```

SAXPY CUBLAS

```
cublasInit();

cublasSetVector(N, sizeof(x[0]), x, 1, d_x, 1);
cublasSetVector(N, sizeof(y[0]), y, 1, d_y, 1);

cublasSaxpy(handle, N, &alpha, d_x, 1, d_y, 1);
```

SAXPY Thrust

```
struct saxpy_functor : public
thrust::binary_function<float,float,float>
{
    const float a;

    saxpy_functor(float _a) : a(_a) {}

    __host__ __device__
    float operator()(const float& x, const float& y) const
    {
        return a * x + y;
    }
};

void saxpy_fast(float A, thrust::device_vector<float>& X,
thrust::device_vector<float>& Y)
{
    // Y <- A * X + Y
    thrust::transform(X.begin(), X.end(), Y.begin(), Y.begin(),
saxpy_functor(A));
}
```

Результаты эксперимента

Для оценки производительности реализации на CUDA были проведены эксперименты. Определялось время выполнения задачи SAXPY на CUDA (сырой код языка СИ), а также с использованием CUBLAS и Thrust. Измерения производились следующим образом: сначала подготавливаются все данные для ядра, затем запускается таймер, после чего выполняется функция-ядро. Далее выполняется синхронизация с CPU любым доступным способом, и таймером замеряется результат.

Конфигурация тестовой системы: процессор – Intel Core i7 2,8 ГГц, оперативная намять объемом 4ГБ, видеокарта па основе видеопроцессора GeForce GT 630M с 2ГБ видеопамяти.

График, представленный на рисунке 1 показывает время выполнения операции SAXPY различных реализаций на векторах размером от 2^{16} до 2^{22} элементов. Видно, что реализация на сыром коде демонстрирует самую высокую скорость выполнения. Thrust и CUBLAS отстают в скорости. Возможно, это связано с лишними затратами времени па установку состояния API и синхронизацию с GPU при запуске каждой итерации расчета. Вызовы функций thurst и cublas осуществляются через CUDA, поэтому па выполнение требуется дополнительное время.

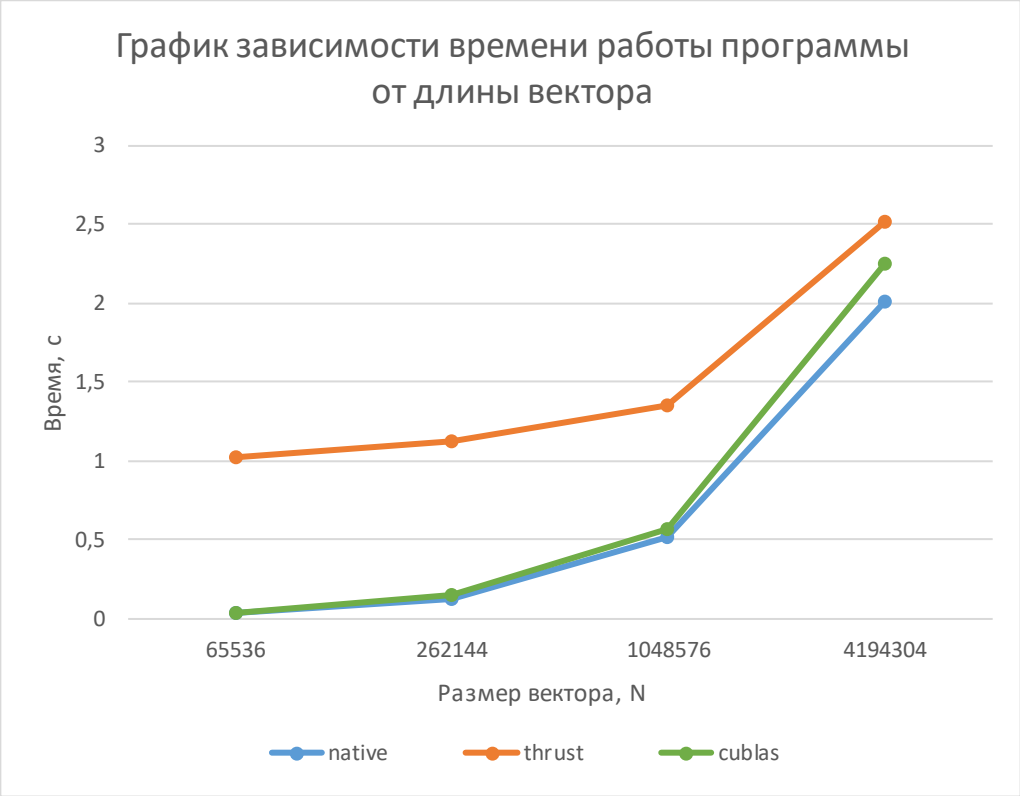

Рис 1 – Сравнение времени вычисления SAXPY различных реализаций

Выводы

С использованием технологии CUDA было реализовано решение задачи линейной алгебры SAXPY. По результатам теста были сделаны выводы касательно производительности реализаций.

Литература

1. Описание алгебраической функции SAXPY // Википедия. URL: http://en.wikipedia.org/wiki/SAXPY (дата обращения: 09.12.2017).

2. Боресков А. Основы CUDA. URL: http://steps3d.narod.ru/tutorials/cuda-tutorial.html (дата обращения: 09.12.2017).

3. Zibula A. General Purpose Computation on Graphics Processing Units (GPGPU) using CUDA // Parallel Programming and Parallel Algorithms Seminar. Munster: Westfalische Wilhelms-Universität, 2009.

4. Библиотека CUBLAS для CUDA. URL: http://developer.download.nvidia.com/ compute/cuda/2_ 3/toolkit/ docs/CUBLAS_Librarv_2.3.pdf (дата обращения: 09.12.2017).

5. Страницы загрузок средств разработки CUDA 2.3. URL: http://developer.nvidia.com/object/cuda_2_3_downloads.html (дата обращения: 28.02.2010).

6. CUDA 3.0 beta. URL: http://forums.nvidia.com/index.php?showtopic= 149959 (дата обращения: 09.12.2017).

7. NVIDIA CUDA Programming Guide. URL: http://developer.download.nvidia.com/ compute/cuda/2_ 3/toolkit/docs/ NVIDIA_CUDA_Programming_Guide_2.3.pdf (дата обращения: 09.12.2017).