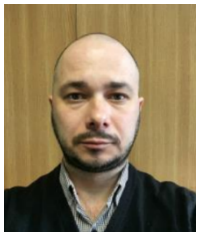
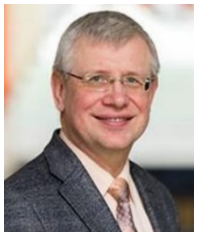
УДК 004.021:004.75

МАТЕМАТИЧЕСКИЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ ПРОЦЕССОРОВ ПО ТЕХНОЛОГИИ CUDA



Е.И. Лещевич
аспирант БГУИР, инженер кафедры электронной техники и технологии БГУИР
e.leshchevich@bsuir.by

П.В. Камлач
заместитель декана факультета компьютерного проектирования, кандидат технических наук, доцент
kamlachpv@bsuir.by



В.М. Бондарик
декан факультета доуниверситетской подготовки и профессиональной ориентации, кандидат технических наук,доцент

А.В. Чураков
доцент кафедры электронной техники и технологии, кан.мед.наук, врач анестезиолог-реаниматолог

Г.Д. Ситник
доцент кафедры общей врачебной практики БелМАПО, кандидат медицинских наук, доцент, врач высшей категории по неврологии

И.И. Ревинская
аспирант БГУИР, ассистент кафедры электронной техники и технологии БГУИР

Е.И. Лещевич

Окончил Белорусский государственный университет информатики и радиоэлектроники. Магистрант кафедры инженерной и компьютерной графики БГУИР. Область научного интереса – влияние инфразвука на биологические ткани.

П.В. Камлач

Окончил Белорусский государственный университет информатики и радиоэлектроники. Доцент, кандидат технических наук, заместитель декана факультета компьютерного проектирования, доцент кафедры электронной техники и технологии БГУИР. Область научного интереса – проектирование медицинских электронных систем.

В.М. Бондарик

Образование: 1983-1988 Минский радиотехнический институт, специальность «Конструирование и производство радиоаппаратуры», квалификация - инженер-конструктор-технолог. Область научного интереса – проектирование медицинских электронных систем, внедрение дистанционных образовательных технологий.

Г.Д. Ситник

Кандидат медицинских наук, доцент, врач высшей категории по неврологии. Область научного интереса – лечение больных с неврологическими проявлениями поясничного остеохондроза,

И.И. Ревинская

Окончила Белорусский государственный университет информатики и радиоэлектроники. Аспирант кафедры электронной техники и технологии. Область научного интереса – медицинская электроника и обработка медицинских сигналов.

Аннотация. Рассмотрена технология вычислений больших массивов данных на базе графических процессоров с использованием технологии CUDA.

Ключевые слова: CUDA, GPU, GPGPU, ALU, видеокарта, графический адаптер

Введение.

Устройства для превращения персональных компьютеров в маленькие суперкомпьютеры известны довольно давно. Ещё в 80-х годах прошлого века на рынке предлагались так называемые транспьютеры, которые вставлялись в распространенные тогда слоты расширения ISA. Первое время их производительность в соответствующих задачах впечатляла, но затем рост быстродействия универсальных процессоров ускорился, они усилили свои позиции в параллельных вычислениях, и смысла в транспьютерах не осталось.

Ещё несколько лет назад появились первые технологии неграфических расчётов общего назначения GPGPU (General-Purpose computation on GPUs). Ведь современные видеочипы содержат сотни математических исполнительных блоков, и эта мощь может использоваться для значительного ускорения множества вычислительно интенсивных приложений. И нынешние поколения GPU обладают достаточно гибкой архитектурой, что вместе с высокоуровневыми языками программирования и программно-аппаратными архитектурами, раскрывает эти возможности и делает их значительно более доступными.

Актуальность.

На создание GPCPU разработчиков побудило появление достаточно быстрых и гибких шейдерных программ, которые способны исполнять современные видеочипы. Разработчики задумали сделать так, чтобы GPU рассчитывали не только изображение в 3D приложениях, но и применялись в других параллельных расчётах. В GPGPU для этого использовались графические API: OpenGL и Direct3D, когда данные к видеочипу передавались в виде текстур, а расчётные программы загружались в виде шейдеров. Вычисления на GPU развивались и развиваются очень быстро. И в дальнейшем, два основных производителя видеочипов, Nvidia и AMD, разработали и анонсировали соответствующие платформы под названием CUDA (Compute Unified Device Architecture) и CTM (Close To Metal или AMD Stream Computing)

Разница между CPU и GPU в параллельных расчётах.

Рост частот универсальных процессоров упёрся в физические ограничения и высокое энергопотребление, и увеличение их производительности всё чаще происходит за счёт размещения нескольких ядер в одном чипе. Продаваемые сейчас процессоры содержат лишь до четырёх ядер (дальнейший рост не будет быстрым) и они предназначены для обычных приложений, используют MIMD — множественный поток команд и данных. Каждое ядро работает отдельно от остальных, исполняя разные инструкции для разных процессов.

Специализированные векторные возможности (SSE2 и SSE3) для четырехкомпонентных (одинарная точность вычислений с плавающей точкой) и двухкомпонентных (двойная точность) векторов появились в универсальных процессорах из-за возросших требований графических приложений, в первую очередь. Именно поэтому для определённых задач применение GPU выгоднее, ведь они изначально сделаны для них.

Например, в видеочипах Nvidia основной блок — это мультипроцессор с восемью-десятью ядрами и сотнями ALU в целом, несколькими тысячами регистров и небольшим количеством разделяемой общей памяти. Кроме того, видеокарта содержит быструю

глобальную память с доступом к ней всех мультипроцессоров, локальную память в каждом мультипроцессоре, а также специальную память для констант.

Самое главное — эти несколько ядер мультипроцессора в GPU являются SIMD (одиночный поток команд, множество потоков данных) ядрами. И эти ядра исполняют одни и те же инструкции одновременно, такой стиль программирования является обычным для графических алгоритмов и многих научных задач, но требует специфического программирования. Зато такой подход позволяет увеличить количество исполнительных блоков за счёт их упрощения.

Итак, перечислим основные различия между архитектурами CPU и GPU. Ядра CPU созданы для исполнения одного потока последовательных инструкций с максимальной производительностью, а GPU проектируются для быстрого исполнения большого числа параллельно выполняемых потоков инструкций. Универсальные процессоры оптимизированы для достижения высокой производительности единственного потока команд, обрабатывающего и целые числа и числа с плавающей точкой. При этом доступ к памяти случайный.

Разработчики CPU стараются добиться выполнения как можно большего числа инструкций параллельно, для увеличения производительности. Для этого, начиная с процессоров Intel Pentium, появилось суперскалярное выполнение, обеспечивающее выполнение двух инструкций за такт, а Pentium Pro отличился внеочередным выполнением инструкций. Но у параллельного выполнения последовательного потока инструкций есть определённые базовые ограничения и увеличением количества исполнительных блоков кратного увеличения скорости не добиться.

У видеочипов работа простая и распараллеленная изначально. Видеочип принимает на входе группу полигонов, проводит все необходимые операции, и на выходе выдаёт пиксели. Обработка полигонов и пикселей независима, их можно обрабатывать параллельно, отдельно друг от друга. Поэтому, из-за изначально параллельной организации работы в GPU используется большое количество исполнительных блоков, которые легко загрузить, в отличие от последовательного потока инструкций для CPU. Кроме того, современные GPU также могут исполнять больше одной инструкции за такт (dual issue). Так, архитектура Tesla в некоторых условиях запускает на исполнение операции MAD+MUL или MAD+SFU одновременно.

GPU отличается от CPU ещё и по принципам доступа к памяти. В GPU он связанный и легко предсказуемый — если из памяти читается тексель текстуры, то через некоторое время придёт время и для соседних текселей. Да и при записи то же — пиксель записывается во фреймбуфер, и через несколько тактов будет записываться расположенный рядом с ним. Поэтому организация памяти отличается от той, что используется в CPU. И видеочипу, в отличие от универсальных процессоров, просто не нужна кэш-память большого размера, а для текстур требуются лишь несколько (до 128-256 в нынешних GPU) килобайт.

Да и сама по себе работа с памятью у GPU и CPU несколько отличается. Так, не все центральные процессоры имеют встроенные контроллеры памяти, а у всех GPU обычно есть по несколько контроллеров, вплоть до восьми 64-битных каналов в чипе Nvidia GT200. Кроме того, на видеокартах применяется более быстрая память, и в результате видеочипам доступна в разы большая пропускная способность памяти, что также весьма важно для параллельных расчётов, оперирующих с огромными потоками данных.

В универсальных процессорах большие количества транзисторов и площадь чипа идут на буферы команд, аппаратное предсказание ветвления и огромные объёмы начиповой кэш-памяти. Все эти аппаратные блоки нужны для ускорения исполнения немногочисленных потоков команд. Видеочипы тратят транзисторы на массивы исполнительных блоков, управляющие потоками блоки, разделяемую память небольшого

объёма и контроллеры памяти на несколько каналов. Вышеперечисленное не ускоряет выполнение отдельных потоков, оно позволяет чипу обрабатывать нескольких тысяч потоков, одновременно исполняющихся чипом и требующих высокой пропускной способности памяти.

Про отличия в кэшировании. Универсальные центральные процессоры используют кэш-память для увеличения производительности за счёт снижения задержек доступа к памяти, а GPU используют кэш или общую память для увеличения полосы пропускания. CPU снижают задержки доступа к памяти при помощи кэш-памяти большого размера, а также предсказания ветвлений кода. Эти аппаратные части занимают большую часть площади чипа и потребляют много энергии. Видеочипы обходят проблему задержек доступа к памяти при помощи одновременного исполнения тысяч потоков — в то время, когда один из потоков ожидает данных из памяти, видеочип может выполнять вычисления другого потока без ожидания и задержек.

Есть множество различий и в поддержке многопоточности. CPU исполняет 1-2 потока вычислений на одно процессорное ядро, а видеочипы могут поддерживать до 1024 потоков на каждый мультипроцессор, которых в чипе несколько штук. И если переключение с одного потока на другой для CPU стоит сотни тактов, то GPU переключает несколько потоков за один такт.

Кроме того, центральные процессоры используют SIMD (одна инструкция выполняется над многочисленными данными) блоки для векторных вычислений, а видеочипы применяют SIMT (одна инструкция и несколько потоков) для скалярной обработки потоков. SIMT не требует, чтобы разработчик преобразовывал данные в векторы, и допускает произвольные ветвления в потоках.

Области применения параллельных расчётов на GPU.

Чтобы понять, какие преимущества приносит перенос расчётов на видеочипы, приведём усреднённые цифры, полученные исследователями по всему миру. В среднем, при переносе вычислений на GPU, во многих задачах достигается ускорение в 5-30 раз, по сравнению с быстрыми универсальными процессорами. Самые большие цифры (порядка 100-кратного ускорения и даже более!) достигаются на коде, который не очень хорошо подходит для расчётов при помощи блоков SSE, но вполне удобен для GPU.

Это лишь некоторые примеры ускорений синтетического кода на GPU против SSE-векторизованного кода на CPU (по данным Nvidia):

• Флуоресцентная микроскопия: 12x;

• Молекулярная динамика (non-bonded force calc): 8-16x;

• Электростатика (прямое и многоуровневое суммирование Кулона): 40-120x и 7x.

Как видите, цифры весьма привлекательные, особенно впечатляют 100-150-кратные приросты. В следующей статье, посвящённой CUDA, мы подробно разберём некоторые из этих цифр. А сейчас перечислим основные приложения, в которых сейчас применяются вычисления на GPU: анализ и обработка изображений и сигналов, симуляция физики, вычислительная математика, вычислительная биология, финансовые расчёты, базы данных, динамика газов и жидкостей, криптография, адаптивная лучевая терапия, астрономия, обработка звука, биоинформатика, биологические симуляции, компьютерное зрение, анализ данных (data mining), цифровое кино и телевидение, электромагнитные симуляции, геоинформационные системы, военные применения, горное планирование, молекулярная динамика, магнитно-резонансная томография (MRI), нейросети, океанографические исследования, физика частиц, симуляция свёртывания молекул белка, квантовая химия, трассировка лучей, визуализация, радары, гидродинамическое моделирование (reservoir simulation), искусственный интеллект, анализ спутниковых данных, сейсмическая разведка, хирургия, ультразвук, видеоконференции.

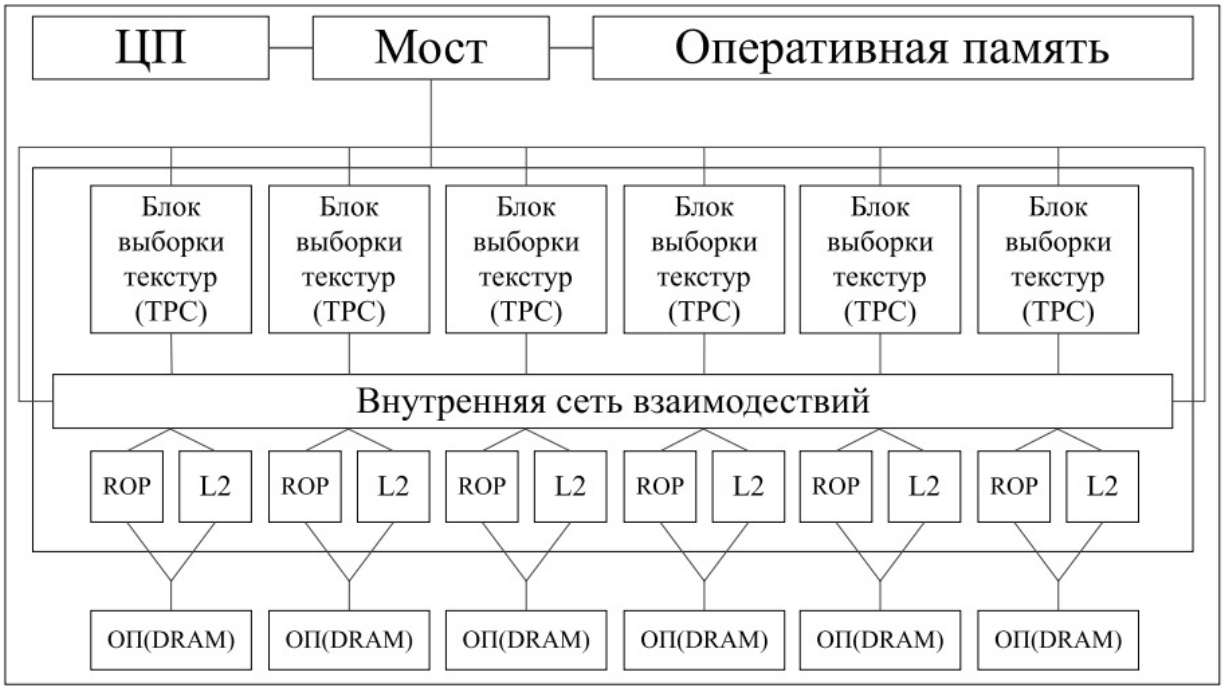


Рисунок 1. Схематическое изображение устройства графического адаптера

Модель памяти CUDA

Модель памяти в CUDA отличается возможностью побайтной адресации, поддержкой как gather, так и scatter. Доступно довольно большое количество регистров на каждый потоковый процессор, до 1024 штук. Доступ к ним очень быстрый, хранить в них можно 32-битные целые или числа с плавающей точкой.

Каждый поток имеет доступ к следующим типам памяти (Рис. 2): глобальная память — самый большой объём памяти, доступный для всех мультипроцессоров на видеочипе, размер составляет от 256 мегабайт до 1.5 гигабайт на текущих решениях (и до 4 Гбайт на Tesla). Обладает высокой пропускной способностью, более 100 гигабайт/с для топовых решений Nvidia, но очень большими задержками в несколько сот тактов. Не кэшируется, поддерживает обобщённые инструкции load и store, и обычные указатели на память.

Локальная память — это небольшой объём памяти, к которому имеет доступ только один потоковый процессор. Она относительно медленная — такая же, как и глобальная.

Разделяемая память — это 16-килобайтный (в видеочипах нынешней архитектуры) блок памяти с общим доступом для всех потоковых процессоров в мультипроцессоре. Эта память весьма быстрая, такая же, как регистры. Она обеспечивает взаимодействие потоков, управляется разработчиком напрямую и имеет низкие задержки. Преимущества разделяемой памяти: использование в виде управляемого программистом кэша первого уровня, снижение задержек при доступе исполнительных блоков (ALU) к данным, сокращение количества обращений к глобальной памяти.

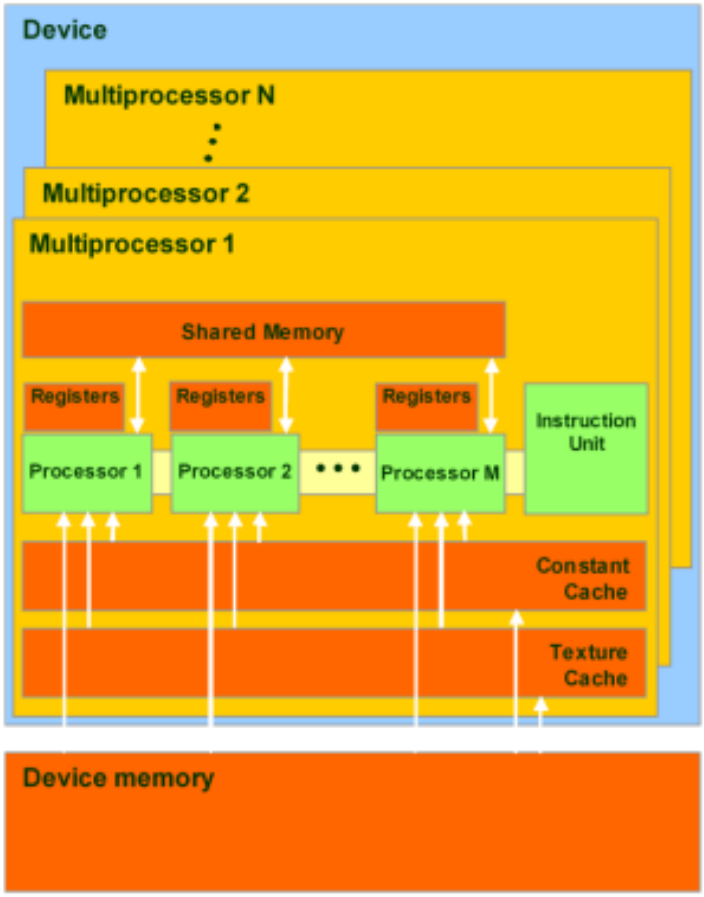


Рисунок 2. Модель доступа к различным типам памяти

Память констант — область памяти объемом 64 килобайта (то же — для нынешних GPU), доступная только для чтения всеми мультипроцессорами. Она кэшируется по 8 килобайт на каждый мультипроцессор. Довольно медленная — задержка в несколько сот тактов при отсутствии нужных данных в кэше.

Текстурная память — блок памяти, доступный для чтения всеми мультипроцессорами. Выборка данных осуществляется при помощи текстурных блоков видеочипа, поэтому предоставляются возможности линейной интерполяции данных без дополнительных затрат. Кэшируется по 8 килобайт на каждый мультипроцессор. Медленная, как глобальная — сотни тактов задержки при отсутствии данных в кэше.

Естественно, что глобальная, локальная, текстурная и память констант — это физически одна и та же память, известная как локальная видеопамять видеокарты. Их отличия в различных алгоритмах кэширования и моделях доступа. Центральный процессор может обновлять и запрашивать только внешнюю память: глобальную, константную и текстурную.

Из написанного выше понятно, что CUDA предполагает специальный подход к разработке, не совсем такой, как принят в программах для CPU. Нужно помнить о разных типах памяти, о том, что локальная и глобальная память не кэшируется и задержки при доступе к ней гораздо выше, чем у регистровой памяти, так как она физически находится в отдельных микросхемах.

Типичный, но не обязательный шаблон решения задач:

• задача разбивается на подзадачи;

• входные данные делятся на блоки, которые вмещаются в разделяемую память;

• каждый блок обрабатывается блоком потоков;
• подблок подгружается в разделяемую память из глобальной;
• над данными в разделяемой памяти проводятся соответствующие вычисления;
• результаты копируются из разделяемой памяти обратно в глобальную.

Среда программирования

В состав CUDA входят runtime библиотеки:

• общая часть, предоставляющая встроенные векторные типы и подмножества вызовов RTL, поддерживаемые на CPU и GPU;
• CPU-компонента, для управления одним или несколькими GPU;
• GPU-компонента, предоставляющая специфические функции для GPU.

Основной процесс приложения CUDA работает на универсальном процессоре (host), он запускает несколько копий процессов kernel на видеокарте. Код для CPU делает следующее: инициализирует GPU, распределяет память на видеокарте и системе, копирует константы в память видеокарты, запускает несколько копий процессов kernel на видеокарте, копирует полученный результат из видеопамяти, освобождает память и завершает работу.

В качестве примера для понимания приведем CPU код для сложения векторов, представленный в CUDA:

```
//Размер вектора в элементах
const int N = 1048576;
//размер вектора в байтах
const int dataSize = N * sizeof(float);

//Выделение памяти CPU
float *h_A = (float *)malloc(dataSize);
float *h_B = (float *)malloc(dataSize);
float *h_C = (float *)malloc(dataSize);

//Выделение памяти GPU
float *d_A, *d_B, *d_C;
cudaMalloc((void **)&d_A, dataSize));
cudaMalloc((void **)&d_B, dataSize));
cudaMalloc((void **)&d_C, dataSize));

//Инициализировать h_A[], h_B[]...

//Скопировать входные данные в GPU для обработки
cudaMemcpy(d_A, h_A, dataSize, cudaMemcpyHostToDevice) );
cudaMemcpy(d_B, h_B, dataSize, cudaMemcpyHostToDevice) );

//Запустить ядро из N / 256 блоков по 256 потоков
//Предполагая, что N кратно 256
vectorAdd<<<N / 256, 256>>>(d_C, d_A, d_B);

//Считать результаты GPU
cudaMemcpy(h_C, d_C, dataSize, cudaMemcpyDeviceToHost) );
```

Рисунок 3. Пример кода сложения векторов

Функции, исполняемые видеочипом, имеют следующие ограничения: отсутствует рекурсия, нет статических переменных внутри функций и переменного числа аргументов.

Поддерживается два вида управления памятью: линейная память с доступом по 32-битным указателям, и CUDA-массивы с доступом только через функции текстурной выборки.

Программы на CUDA могут взаимодействовать с графическими API: для рендеринга данных, сгенерированных в программе, для считывания результатов рендеринга и их обработки средствами CUDA (например, при реализации фильтров постобработки). Для этого ресурсы графических API могут быть отображены (с получением адреса ресурса) в пространство глобальной памяти CUDA. Поддерживаются следующие типы ресурсов графических API: Buffer Objects (PBO / VBO) в OpenGL, вершинные буферы и текстуры (2D, 3D и кубические карты) Direct3D9.

Функции, исполняемые видеочипом, имеют следующие ограничения: отсутствует рекурсия, нет статических переменных внутри функций и переменного числа аргументов. Поддерживается два вида управления памятью: линейная память с доступом по 32-битнымуказателям, и CUDA-массивы с доступом только через функции текстурной выборки.

Программы на CUDA могут взаимодействовать с графическими API: для рендеринга данных, сгенерированных в программе, для считывания результатов рендеринга и их обработки средствами CUDA (например, при реализации фильтров постобработки). Для этого ресурсы графических API могут быть отображены (с получением адреса ресурса) в пространство глобальной памяти CUDA. Поддерживаются следующие типы ресурсов графических API: Buffer Objects (PBO / VBO) в OpenGL, вершинные буферы и текстуры (2D, 3D и кубические карты) Direct3D9.

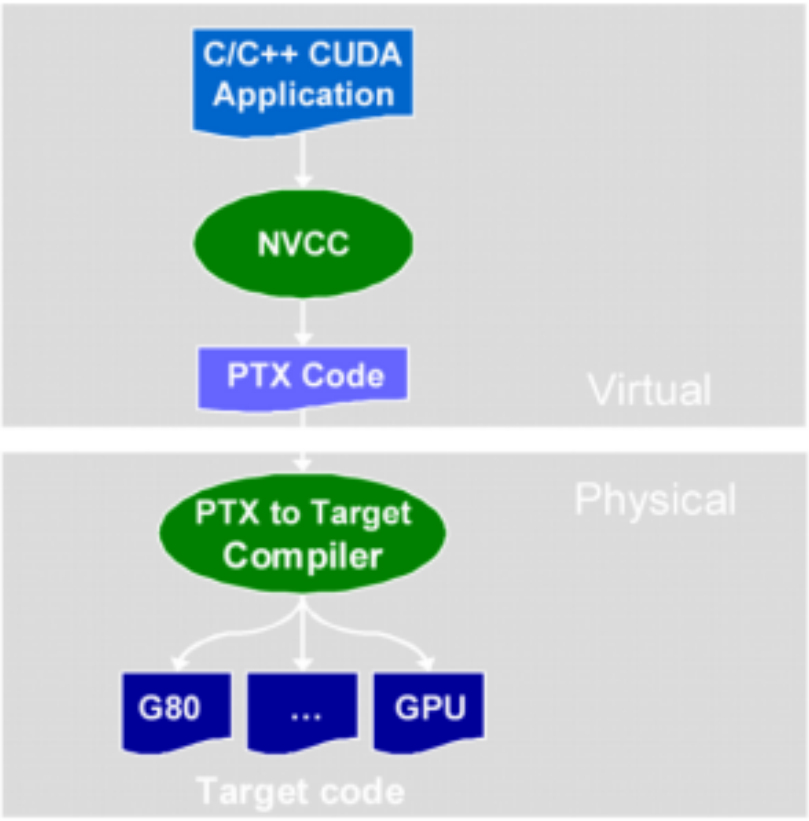


Рисунок 4. Стадии компиляции CUDA-приложения

Файлы исходного кода на CUDA C компилируются при помощи программы NVCC, которая является оболочкой над другими инструментами, и вызывает их: cudacc, g++, cl и

др. NVCC генерирует: код для центрального процессора, который компилируется вместе с остальными частями приложения, написанными на чистом Си, и объектный код PTX для видеочипа. Исполнимые файлы с кодом на CUDA в обязательном порядке требуют наличия библиотек CUDA runtime library (cudart) и CUDA core library (cuda).

Заключение.

Представленная компанией Nvidia программно-аппаратная архитектура для расчётов на видеочипах CUDA хорошо подходит для решения широкого круга задач с высоким параллелизмом. CUDA работает на большом количестве видеочипов 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.

Список литературы

[1] Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учебное пособие. А. В. Боресков и др. Предисл.: В. А. Садовничий Издательство Московского университета, 2012

[2] Технология CUDA в примерах. Введение в программирование графических процессоров Джейсон Сандерс, Эдвард Кэндрот ДМК Пресс, 2011 г.

[3] Боресков А. В., Харламов А. А., Основы работы с технологией CUDA, Москва: ДМК Пресс, 2010.

[4] Краткий курс обработки изображений. -URL: https://hub.exponenta.ru/post/kratkiy-kurs-teorii-obrabotkiizobrazheniy734

MATHEMATICAL CALCULATIONS USING GRAPHIC PROCESSORS USING CUDA TECHNOLOGY

E.I. Leshchevich
Postgraduate student of the BSUIR, assistant of the Department of Electronic Engineering and Technology of BSUIR

P.V. Kamlach
Deputy Dean of of Computer-Aided Design, PhD, Associate Professor at the Department of Electronic Technique and Technology

V. M. Bandarik
Dean of the Faculty of Pre-University Preparation and Occupational Guidance, PhD, Associate Professor

A.V. Churakov
PhD, Associate Professor at the Department of Electronic Technique and Technology

G.D. Sitnik
PhD, Associate Professor at the Department of Belarusian Medical Academy of Postgraduate Education, doctor of the highest category in neurology

I.I. Revinskaya
Postgraduate at the Department of Electronic Technique and Technology

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
Email: e.leshchevich@bsuir.by, kamlachpv@bsuir.by

Abstract. The technology of computing large data arrays based on graphic processors using CUDA technology is considered.

Keywords: CUDA, GPU, GPGPU, ALU, graphics card, graphics adapter