Казанский федеральный университет

ВЫСШАЯ ШКОЛА информационных технологий и информационных систем

Программная модель CUDA

Требования

- Совместимая с Nvidia CUDA видеокарта
- Драйвер Nvidia
- Установленный CUDA Toolkit
- Установленный компилятор C/C++

Видеокарта и драйвера

- Список поддерживаемых видеокарт
  - https://developer.nvidia.com/cuda-gpus
- Каждая версия CUDA Toolkit имеет свой список актуальных драйверов
- Драйвера GPU Nvidia
  - http://www.nvidia.com/drivers

CUDA Toolkit

- Документация
  - http://docs.nvidia.com/cuda/index.html
- Поддержка ОС: Windows, Linux, Mac
- Каждая версия CUDA поддерживает определенный диапазон архитектур
- Если GPU устарел, можно установить старую версию CUDA Toolkit
  - https://developer.nvidia.com/cuda-toolkit-archive

CUDA Toolkit

- Алгоритм
  - Определить модель GPU
  - Определить последнюю версию CUDA для данного GPU
  - Поставить последний драйвер поддерживающий данную версию CUDA
  - Найти соответствующую документацию CUDA и следовать инструкциям

Компиляция

- nvcc – строковый компилятор для CUDA
- Исходные файлы – *.cu
- Основные опции командной строки – в документации CUDA Toolkit
- Работа с компилятором nvcc идентична gcc
- Пример компиляции исходного файла:
  - ***$nvcc -arch=sm_37 –O3 test.cu -o test***

Visual Studio и CUDA

- CUDA версии 6.0 и старше интегрируются в Visual Studio
- Существуют ограничения по совместимости версий CUDA/Visual Studio (RTFM)

Nvidia CUDA

- Программная модель CUDA является гетерогенной – использует и GPU и CPU
- В CUDA выделяют
  - Host (хост) – обозначает CPU и оперативную память
  - Device – обозначает GPU и графическую память
  - Kernels (ядра) – функции исполняемые на GPU

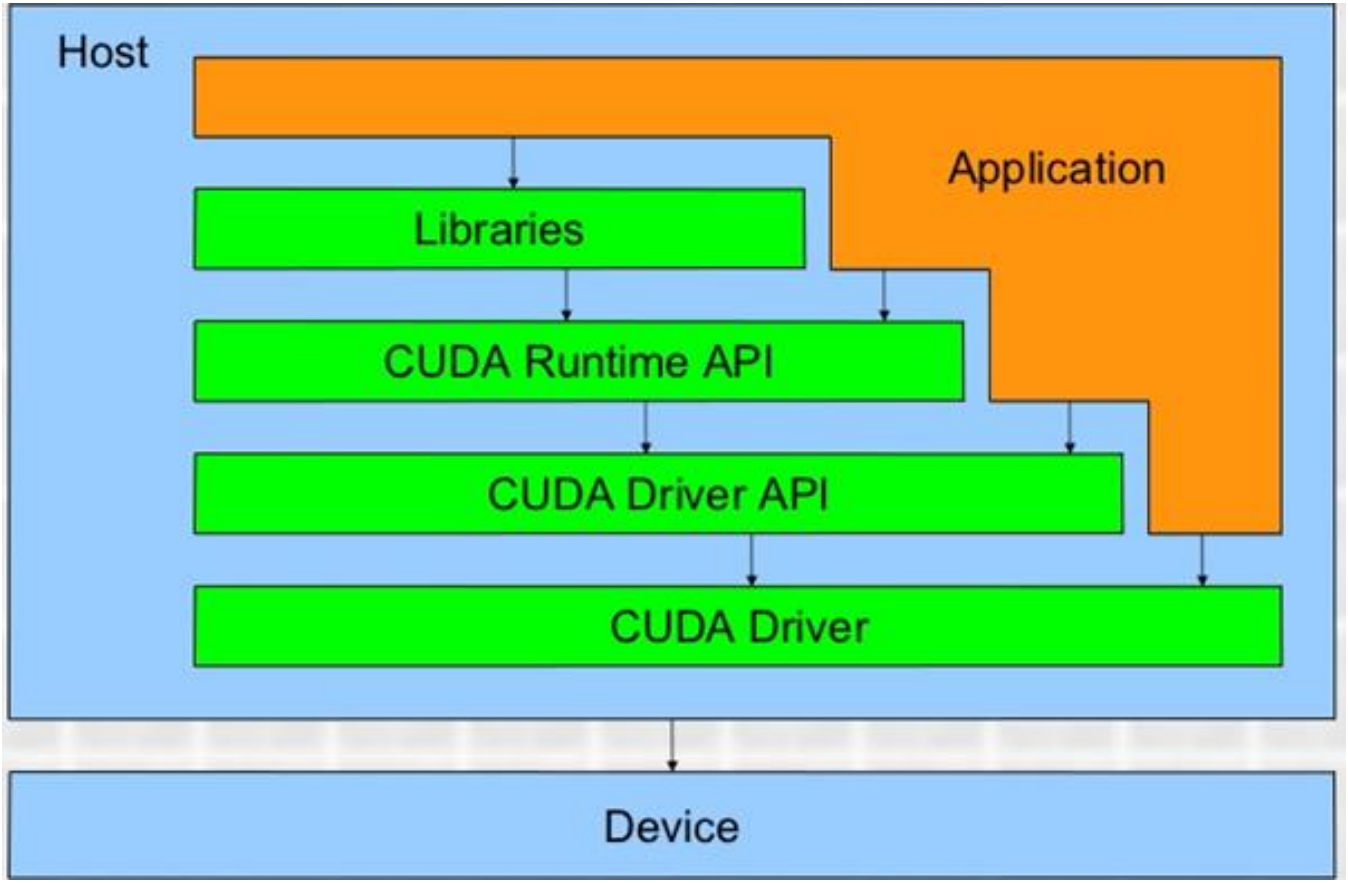
Nvidia CUDA

- Типичная последовательность команд программы на CUDA
  - Объявить и выделить память на хосте и девайсе
  - Инициализировать данные на хосте
  - Скопировать данные с хоста на девайс
  - Запустить ядро(а) на исполнение
  - Скопировать память обратно на хост
  - Освободить выделенную память

Nvidia CUDA

- Код на хосте управляет памятью и хоста и девайса, а также запускает ядра для выполнения на девайсе
- Ядра исполняются на девайсе параллельно множеством нитей (threads)
- Каждая нить выполняет один и тот же код функции-ядра
- Как определить, какую часть данных должна обрабатывать каждая нить?

Программный стек CUDA



Сетки, блоки и нити

- Нити организованы в иерархию
  - Нити (threads)
  - Блоки нитей (blocks of threads)
  - Сетка блоков (grid of blocks)
- При запуске ядра обязательно указывается два параметра
  - Количество нитей в блоке
  - Количество блоков

Сетки, блоки и нити

- Нити в блоке и блоки в сетке могут быть организованы в 1-, 2- и 3-х мерные массивы
- Каждая нить хранит в специальных переменных следующие значения
  - Свой номер в блоке
  - Номер своего блока
  - Количество блоков в сетке
- Таким образом нить может вычислить свой уникальный порядковый номер

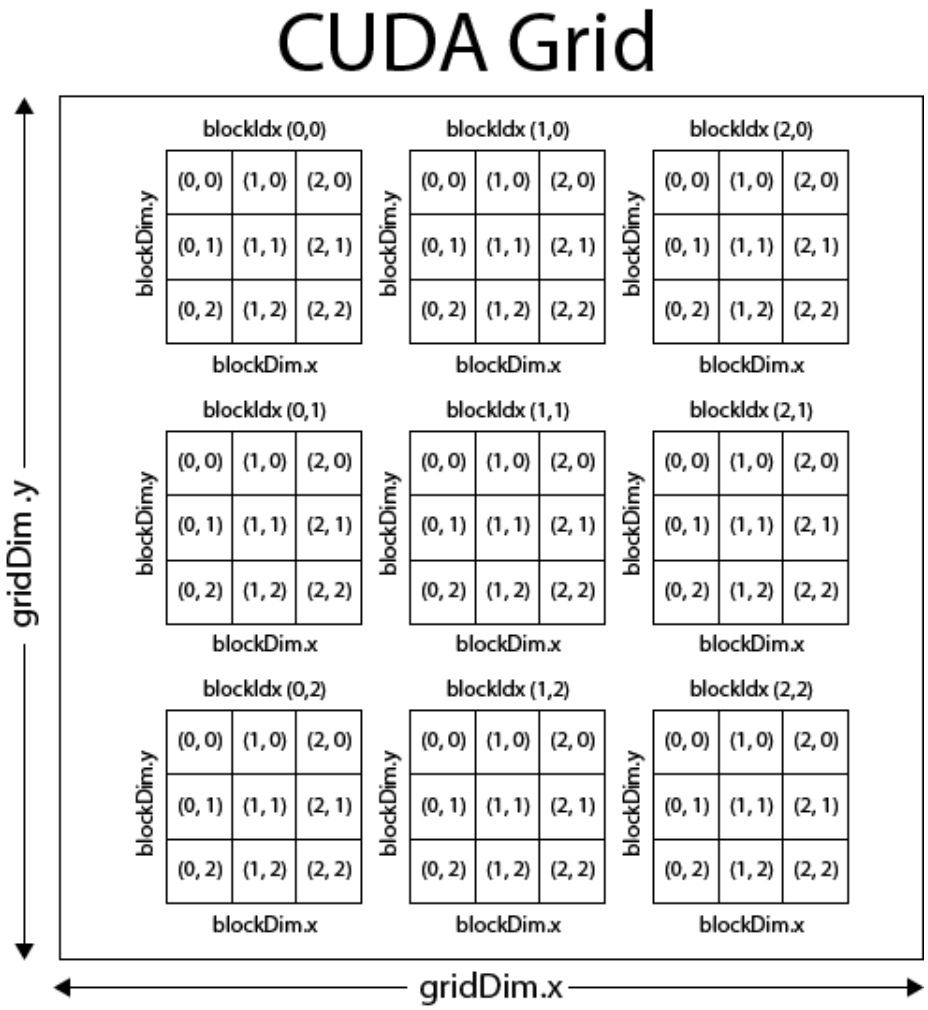
Сетки, блоки и нити

- Каждый из блоков сетки исполняется на одном SM
- В каждый момент времени на SM может исполняться лишь один блок
- Распределение блоков по SM происходит динамически, управляется логикой GPU
- Коммуникация между нитями возможна лишь в пределах одного блока

Сетки, блоки и нити

- Нити блока делятся на логические группы по 32 нити – варпы (warps)
- Только нити одного варпа исполняются физически одновременно
- Распределение нитей по варпам происходит автоматически без участия программиста
- Если $n_{threads\_in\_block}$ % 32 != 0 один из варпов будет содержать неактивные нити
- $MAX(n_{threads\_in\_block})$ = 1024

Сетки, блоки и нити



Hello, world: CUDA-style

- Классический пример hello, world для CUDA – вычисление SAXPY
- SAXPY – Single precision A*X Plus Y
- Поэлементное сложение векторов, один из которых умножен на скаляр

$$\vec{z} = \alpha\vec{x} + \vec{y}$$

Расширения языка

- Атрибуты функций
- Атрибуты переменных
- Встроенные переменные
- Дополнительные типы данных
- Оператор запуска ядра

Атрибуты функций

- Атрибут *__global__*
  - Обозначает функцию-ядро - вызывается с хоста исполняется на девайсе
  - Ядро должно возвращать значение типа *void*
- Атрибут *__device__*
  - Обозначает девайс-функцию – вызывается из ядра или другой девайс-функции
  - Исполняется на девайсе
- Атрибут *__host__*
  - Обычная функция C++

Атрибуты функций

Атрибут	Выполняется на	Вызывается из
__device__	device	device
__global__	device	host
__host__	host	host

Атрибуты переменных

- Атрибуты переменных определяют в какой памяти GPU будет размещаться переменная
- Скорость и способ доступа к переменной сильно варьируется в зависимости от типа размещения
- Подробнее о памяти GPU – в соответствующей лекции

Атрибуты переменных

- CUDA не поддерживает модульную сборку – каждая __*global*__ функция должна находиться в одном исходном файле вместе со всеми __*device*__ функциями и переменными, которые она использует

Встроенные типы

- 1/2/3/4-мерные векторные типы на основе *char, short, int, long, float* и *double*
- Компоненты векторных типов имеют имена x, y, z и w

```
int2 a = make_int2(1, 7); //Создает вектор (1, 7)

float3 u = make_float3(1, 2, 3.4f); //Создает вектор (1.0f, 2.0f, 3.4f )
```

Встроенные типы

- Для этих типов не поддерживаются автоматические векторные операции
- Тип *dim3*, используемый для задания размерностей блоков потоков и сеток блоков
- Этот тип основан на *uint3*, элементы по умолчанию инициализируются 1

```
dim3 blocks ( 16, 16 ); // то же что blocks ( 16, 16, 1 )

dim3 grid ( 256 ); // то же что grid (256, 1, 1)
```

Встроенные переменные

- *gridDim* – количество блоков в сетке по каждому из измерений (тип *dim3*)
- *blockDim* – количество нитей в блоке по каждому из измерений (тип *dim3*)
- *blockIdx* – индекс текущего блока в сетке по каждому из измерений (тип *uint3*)
- *threadIdx* – индекс текущей нити в блоке по каждому из измерений (тип *uint3*)
- *warpSize* – размер варпа (тип *int*)

Оператор вызова ядра

```
kernel_name<<<
              dim3 Nb,
              dim3 Nt,
              size_t Ns = 0,
              cudaStream_t S = 0>>>(args)
```

Оператор вызова ядра

- kernel_name – это имя или адрес ядра
- Обязательный параметр Nb типа задает число блоков в сетке блоков
- Обязательный параметр Ns задает число нитей в блоке
- Параметр Ns задает дополнительный объем разделяемой памяти в для блоков сетки
- Параметр S ставит вызов ядра в заданную очередь команд (CUDA Stream)

Обработка ошибок

- Каждая функция CUDA (кроме запуска ядра) возвращает значение типа $cudaError\_t$
- При успешном выполнении функции возвращается значение $cudaSuccess$, иначе возвращается код ошибки

```
char* cudaGetErrorString(cudaError_t code);//описание ошибки

cudaError_t cudaGetLastError();//получить последнюю ошибку
```

Асинхронность в CUDA

- Некоторые функции CUDA являются асинхронными
  - Запуск ядра
  - Асинхронные версии функций копирования и инициализации памяти
  - Функции копирования памяти device ↔ device внутри устройства и между устройствами
- Для синхронизации устройства используется функция *cudaDeviceSynchronize()*

CUDA Streams

- Очередь команд для запуска ядер и обмена данными между GPU и хостом
- По умолчанию все команды работают в очереди номер 0
- Каждый stream определяет идентификатор – аргумент для команд CUDA
- Команды с одним значением stream выполняются последовательно, с разными – независимо друг от друга

CUDA Streams

- Streams могут использоваться для оптимизации доступа в память
- Если копирование данных хост-девайс, девайс-хост и работа ядра занимают примерно одинаковое время, можно получить выигрыш в производительности

CUDA Streams

```
cudaStream_t streams[nStreams];

for (int i = 0; i < nStreams; ++i)
{
   cudaStreamCreate(&streams[i]);
   int of = i * streamSize;
   cudaMemcpyAsync(&d_a[of], &a[of], nB, cudaMemcpyHostToDevice, stream[i]);
   kernel<<<streamSize/blockSize, blockSize, 0, stream[i]>>>(d_a, offset);
   cudaMemcpyAsync(&a[of], &d_a[of], nB, cudaMemcpyDeviceToHost, stream[i]);
}
```

CUDA events

- Событие – объект типа cudaEvent_t для измерения времени выполнения операций CUDA
- Функции CUDA API позволяют
  - Создавать и уничтожать события
  - Выделять начало и конец профилируемого кода
  - Проверять/ожидать наступления события
  - Замерять время выполнения

Профилировщик CUDA

- Профилирование осуществляется через NVIDIA Visual Profiler
- Определение гонок данных, бутылочных горлышек и т.д.
- Кроме того, можно воспользоваться командой nvprof:
  - $nvprof [options] [application] [application args]

Казанский федеральный
УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Вопросы

ekhramch@kpfu.ru