Программные возможности CUDA

Основные понятия

- Хост (Host) —
- Устройство (Device) —
- Тред (Thread, нить) —
- Блок (Block) —

- Грид (Grid) —
- Ядро (Kernel) —
- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) —
- Тред (Thread, нить) —
- Блок (Block) —
- Грид (Grid) —
- Ядро (Kernel) —
- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) —
- Блок (Block) —
- Грид (Grid) —
- Ядро (Kernel) —
- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) — единица выполнения программы. Имеет свой уникальный идентификатор внутри блока.
- Блок (Block) —

- Грид (Grid) —

- Ядро (Kernel) —

- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) — единица выполнения программы. Имеет свой уникальный идентификатор внутри блока.
- Блок (Block) — объединение тредов, которое выполняется целиком на одном SM. Имеет свой уникальный идентификатор внутри грида.
- Грид (Grid) —

- Ядро (Kernel) —

- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) — единица выполнения программы. Имеет свой уникальный идентификатор внутри блока.
- Блок (Block) — объединение тредов, которое выполняется целиком на одном SM. Имеет свой уникальный идентификатор внутри грида.
- Грид (Grid) — объединение блоков, которые выполняются на одном устройстве.
- Ядро (Kernel) —

- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) — единица выполнения программы. Имеет свой уникальный идентификатор внутри блока.
- Блок (Block) — объединение тредов, которое выполняется целиком на одном SM. Имеет свой уникальный идентификатор внутри грида.
- Грид (Grid) — объединение блоков, которые выполняются на одном устройстве.
- Ядро (Kernel) — параллельная часть алгоритма, выполняется на гриде.
- Варп (Warp) —

Основные понятия

- Хост (Host) — центральный процессор, управляющий выполнением программы.
- Устройство (Device) — видеоадаптер, выступающий в роли сопроцессора центрального процессора.
- Тред (Thread, нить) — единица выполнения программы. Имеет свой уникальный идентификатор внутри блока.
- Блок (Block) — объединение тредов, которое выполняется целиком на одном SM. Имеет свой уникальный идентификатор внутри грида.
- Грид (Grid) — объединение блоков, которые выполняются на одном устройстве.
- Ядро (Kernel) — параллельная часть алгоритма, выполняется на гриде.
- Варп (Warp) — 32 последовательно идущих треда, выполняется физически одновременно.

Отличие нити от потока

Отличие нити от потока

- Нить GPU чрезвычайно легковесна

  контекст минимален

  регистры распределены заранее
- Необходимо использовать тысячи отдельных нитей для эффективного использования ресурсов GPU

  - для CPU?

SIMD -> SIMT

Single Instruction – Multiple Threads

SIMD -> SIMT

Single Instruction – Multiple Threads

Одновременно выполняются нити в одном варпе (warp) – 32 нити для Fermi

Нити разных варпов находятся на разных стадиях выполнения программы

Асинхронное выполнение

Управление в вызывающую функцию возвращается до завершения требуемой операции

Асинхронное выполнение

- запуск ядра
- Функции копирования памяти, которые заканчиваются на Async
- Функции копирования памяти device < = > device внутри устройства и между устройствами
- Функции инициализации памяти

CUDA Streams (потоки исполнения)

Позволяют группировать последовательности операций, которые необходимо выполнять в строго определенном порядке

Порядок выполнения между разными потоками не определен и может изменяться

По умолчанию все операции выполняются в нулевом потоке

Синхронизация CUDA Stream

- cudaDeviceSynchronize

(cudaThreadSynchronize до версии 4.0)

- Использование CUDA Events
- При копировании результатов с GPU по адресу, ранее использованному при запуске ядра

Обработка ошибок

Функции CUDA runtime API (кроме запуска ядра) возвращают значение типа *cudaError_t*

Успешное выполнение = > *cudaSuccess*

«неуспешное» выполнение = > код ошибки

cudaGetErrorString возвращает текстовое описание ошибки

Обработка ошибок: *cudaGetLastError*

Устранимая ошибка:

вызов приводит к сбросу хранимого значения, далее возвращает *CudaSuccess*

Неустранимая ошибка:

возвращает тот же код, устройство находится в некорректном состоянии. Повторная инициализация *cudaDeviceReset*

CUDA Events

Тип *cudaEvent_t*

Функции управления эвентами:

- cudaEventCreate
- cudaEventCreateWithFlags
- cudaEventDestroy
- cudaEventElapsedTime
- cudaEventQuery
- cudaEventRecord
- cudaEventSynchronize

```
// Объявление переменных-событий начала и окончания
// выполнения ядра.
cudaEvent_t start, stop;
float gpuTime = 0.0f;
// Инициализация переменных-событий.
cudaEventCreate ( &start );
cudaEventCreate ( &stop );
// Привязка события start к данной позиции в коде
// программы (начало выполнения ядра).
cudaEventRecord ( start, 0 );
// Запуск GPU-ядра.
myKernel<<<blocks, threads>>> ( adev, bdev, cdev, N);
// Привязка события stop к данной позиции в коде
// программы (окончание выполнения ядра).
cudaEventRecord ( stop, 0 );
// Ожидание окончания выполнения ядра,
// синхронизация по событию stop.
cudaEventSynchronize ( stop );
// Получение времени, прошедшего между событиями start и stop.
cudaEventElapsedTime ( &gpuTime, start, stop );
printf("time spent executing by the GPU: %.2f millseconds\n", gpuTime );
// Уничтожение событий.
cudaEventDestroy ( start );
cudaEventDestroy ( stop );
```

Compute Capability

Пара чисел: 1.*; 2.*; 3.*; … major.minor

major – архитектурная версия

minor – небольшие изменения внутри архитектуры

Встроенные типы

- 1/2/3/4-мерные векторные типы на основе char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, longlong, float, double

char1, char2, char3,char4, uchar1, uchar2, uchar3, uchar4, short1, short2, short3, short4, ushort1, ushort2, ushort3, ushort4, int1, int2, int3, int4, uint1, uint2, uint3, uint4, long1, long2, long3, long4, ulong1, ulong2, ulong3, ulong4, float1, float2, float3, float4, longlong1, longlong2, double1, double2.

Встроенные типы

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

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

Компоненты имеют имена x, y, z, w

a.x = 1, u.z = 3.4f

Встроенные типы

В отличие от шейдерных языков не поддерживаются векторные покомпонентные операции

int3 v1 = make_int3(1, 2, 3)

int3 v2 = make_int3(10, 20, 30)

~~v1 + v2~~ v1.x + v2.x; v1.y + v2.y; v1.z + v2.z

dim3 (основан на *uint3*, нормальный конструктор)

dim3 blocks(16, 16) //эквивалентно blocks(16, 16, 1)

dim3 grid(512) // эквивалентно grid(512, 1, 1)

Встроенные функции

Математические функции

fminf, fmaxf, fabsf

корни: rsqrtf, sqrtf, cbrtf

степени: expf, exp2f, exp10f, expm1f, logf, log2f, log10f, logp1f, powf

округления: roundf (8 инструкций!), rintf, truncf, ceilf, floorf

Аналоги для GPU

__fadd_[rn|rz], __fmul_[rn|rz], __fdividef

__expf, __exp10f, __logf, __log10f, __powf

__sinf, __cosf, __sincosf, __tanf

__saturate

Атомарные операции

Операции для обеспечения корректного доступа к разделяемому ресурсу в параллельной программе

В случае CUDA разделяемый ресурс – переменная, доступная множеству нитей

Запись не происходит одновременно с чтением или еще одной записью

Атомарные операции

Compute Capability > 1.1

атомарные операции с глобальной памятью

Compute Capability > 1.2

Поддержка 64 битных значений и операций с разделяемой памятью

Compute Capability > 2.0

64 битные целые числа в разделяемой памяти

Атомарные операции

(!) Атомарные операции работают только с целыми числами

Кроме *AtomicAdd*, *AtomicSub* – увеличение или уменьшение значения переменной на заданную величину.

могут работать с 32-битные числами с плавающей точкой (ComputeCapability > 2.0)

int atomicAdd (int * addr, int value);

(!) возвращает исходное значение

Атомарные операции

atomicExch – атомарный обмен значениями. Новое значение записывается по адресу, предыдущее возвращается как результат.

```
int atomicExch ( int * addr, int value );
```

atomicMin, atomicMax – сравнение значение по адресу с переданным и запись минимума/максимума по адресу и возврат предыдущего значения по адресу.

```
int atomicMin ( int * addr, int value );
```

atomicInc, atomicDec, atomicCAS

Атомарные операции

Атомарные побитовые операции

Читают слово по адресу, применяют к нему побитовую операцию с заданным параметром и записывают результат обратно. Результатом возвращают исходное значение, находившееся по заданному адресу до начала операции.

```
int atomicAnd ( int * addr, int value );
unsigned int atomicAnd ( unsigned int * addr, unsigned int value );
int atomicOr ( int * addr, int value );
unsigned int atomicOr ( unsigned int * addr, unsigned int value );
int atomicXor ( int * addr, int value );
unsigned int atomicXor ( unsigned int * addr, unsigned int value );
```