

CUDA. API

ЯЗЫК CUDA

Романенко А.А.

arom@ccfit.nsu.ru

Новосибирский государственный университет

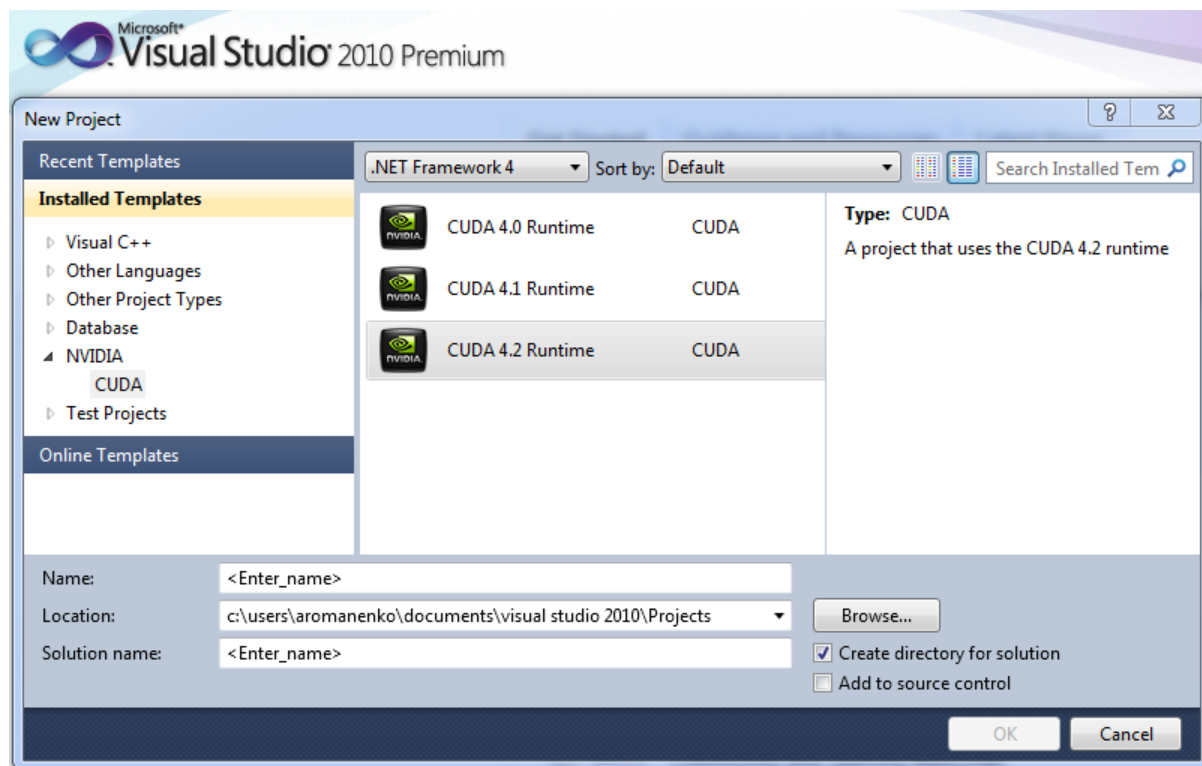
CUDA, компоненты

- * Драйвер (обязательный компонент)
 - * `/lib/modules/...`
- * Toolkit (компилятор, библиотеки CUBALS, CUFFT...)
 - * `/usr/local/cuda`
- * SDK (документация и примеры)
 - * `/usr/local/cudasdk`

Сборка программы

- * Компилятор — nvcc
- * Исходные коды - *.cu или *.cuh
- * Рекомендуется собирать с помощью make
 - * Скопировать к себе из примеров Makefile и common.mk
 - * Поправить пути для выходных файлов
- * Сборка
 - * make emu=1 — в режиме эмуляции определен макрос `__DEVICE_EMULATION__`
 - * make dbg=1 — с отладочной информацией
 - * make — финальной версии программы

После установки NSight 2.0



Для CUDA 3.2 + NSight 1.5

<http://blog.cuvilib.com/2011/02/24/how-to-run-cuda-in-visual-studio-2010/>

Модификаторы функций

- * `__device__`
 - * Исполняется на GPU
 - * Запускается только из GPU
- * `__host__`
 - * Исполняется на CPU
 - * Запускается только с CPU
- * `__global__`
 - * Исполняется на GPU
 - * Запускается только с CPU

Модификаторы функций

Ограничения

- * `__device__` и `__global__` не поддерживают рекурсию
- * В теле `__device__` и `__global__` не должны объявлять статические переменные
- * В `__device__` и `__global__` не может быть переменное число параметров
- * `__global__` и `__host__` не могут использоваться одновременно
- * `__global__` должна возвращать `void` и суммарный объем параметров должен быть не больше 256 байт

Модификаторы типов

- * `__device__`
 - * Располагается в глобальной памяти устройства
 - * Имеет время жизни приложения
 - * Доступна всем потокам в сети и через библиотеки для CPU
- * `__constant__`
 - * Располагается в константной памяти устройства
 - * Имеет время жизни приложения
 - * Доступна всем потокам в сети и через библиотеки для CPU
- * `__shared__`
 - * Располагается в разделяемой памяти потокового блока
 - * Имеет время жизни потокового блока
 - * Доступна только потокам внутри потокового блока

Модификаторы типов. Ограничения

- * `__shared__` переменная не может быть инициализирована при объявлении
- * `__constant__` переменная может быть инициализирована только со стороны CPU
- * Область видимости переменных `__device__` и `__constant__` - файл
- * `__shared__` и `__constant__` переменные неявно имеют статическое хранилище
- * Модификаторы не могут применяться к полям типов `struct` и `union`

Конфигурация времени выполнения

- * Определяется при запуске ядра (`__global__` функции)
 - * `__global__ void Func(float* data);`
 - * `Func<<<Dg, Db, Ns, S>>>(data);`
- * `Dg` — размер сети. Тип `dim3`
 - * `Dg.x`, `Dg.y` задают размер. `Dg.z` не используется
- * `Db` — размер блока. Тип `dim3`
 - * `Db.x * Db.y * Db.z` — количество потоков в блоке
- * `Ns` — размер дополнительной разделяемой памяти на блок. Тип `size_t`. Опциональный. По-умолчанию - 0
- * `S` — номер потока. Тип `cudaStream_t`. Опциональный. По-умолчанию — 0.
- * Выполнение ядра асинхронно

Встроенные переменные

- * **gridDim** — размер сети. Тип dim3.
- * **blockIdx** — индекс блока в сети. Тип uint3.
- * **blockDim** — размерность блока. Тип dim3.
- * **threadIdx** — индекс потока в блоке. Тип uint3.
- * uint3 и dim3 - структуры из трех полей: x, y, z
- * Встроенные переменные нельзя модифицировать
- * Нельзя получить адрес встроенной переменной

Пример

```
__global__ void my_sum(float* a, float* b, float* c, int len){  
    unsigned int index;  
    index = blockIdx.x * blockDim.x + threadIdx.x;  
    if(index < len){  
        c[index] = a[index] + b[index];  
    }  
}  
  
dim3 GS(100);  
dim3 BS(512);  
my_sum<<<GS, BS>>>(a, b, c, 5000);
```

Встроенные векторные типы данных

- * 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
- * Поля: x,y,z,w

Инициализация устройства

- * **cudaGetDeviceCount** — количество устройств GPU
- * **cudaGetDeviceProperties(cudaDeviceProp*, uint)** — получить параметры устройства
- * **cudaSetDevice(uint)** — сделать устройство активным
- * Перед первым вызовом любого ядра или функции из runtime API
- * Если требуется работать с несколькими устройствами, необходимо несколько потоков (threads) в программе
 - * CUDA 3.2 и более ранние версии

Параметры устройства

```
struct cudaDeviceProp{
    char        name[256];
    size_t      totalGlobalMem;
    size_t      sharedMemPerBlock;
    int         regsPerBlock;
    int         warpSize;
    size_t      memPitch;
    int         maxThreadsPerBlock;
    int         maxThreadsDim[3];
    int         maxGridSize[3];
    int         clockRate;
    size_t      totalConstMem;
    int         major;
    int         minor;
    size_t      textureAlignment;
    int         deviceOverlap;
    int         multiProcessorCount;
    int         __cudaReserved[40];
};
```

Выделение памяти

- * CPU

malloc, calloc, free, cudaMallocHost, cudaFreeHost

- * GPU

cudaMalloc, cudaMallocPitch, cudaFree

```
float* ptr;
```

```
cudaMalloc((void**)ptr, 256*sizeof(float));
```

```
....
```

```
cudaFree(ptr);
```

```
....
```

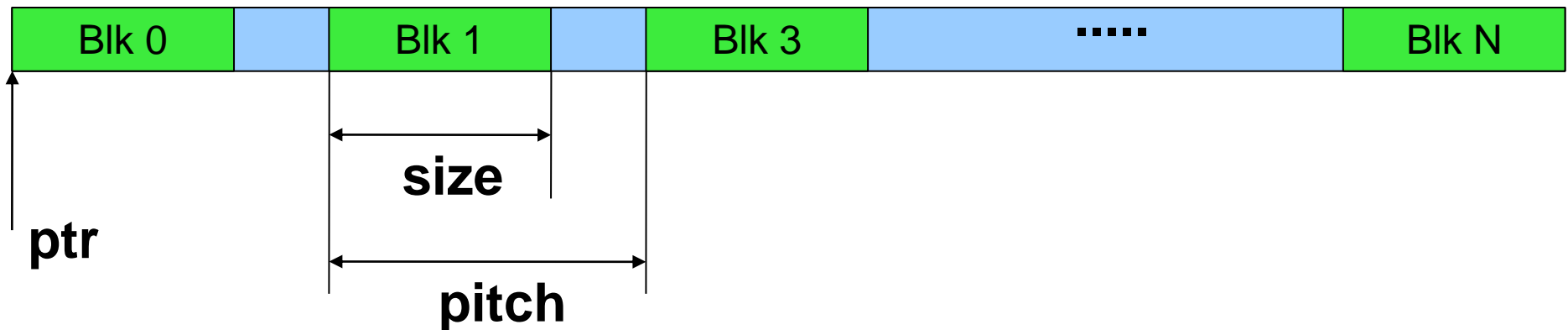
```
cudaMallocPitch((void**)ptr, &pitch, 256*sizeof(float), 256);
```

```
....
```

```
cudaFree(ptr);
```

Выделение памяти

* `cudaMallocPitch((void**)ptr, &pitch, size, blocks);`



- * `cudaMallocArray(struct cudaArray **array, const struct cudaChannelFormatDesc* desc, size_t width, size_t height);`
- * `cudaFreeArray(struct cudaArray *array);`
- * `cudaCreateChannelDesc(int x, int y, int z, int w, enum cudaChannelFormatKind f);`

Копирование данных в/из GPU

- * `cudaMemcpy(void* dst, void* src, size_t size, direction)`
- * `direction`:
 - * `cudaMemcpyHostToDevice`
 - * `cudaMemcpyDeviceToHost`
 - * `cudaMemcpyDeviceToDevice`
- * `cudaMemcpy2D(void* dst, size_t dpitch, const void* src, size_t spitch, size_t width, size_t height, direction)`
- * И т.д.

Атомарные операции

- * Только знаковые и беззнаковые целые
- * Операции над 32-битными словами в глобальной памяти
- * `atomicAdd`, `atomicSub`, `atomicExch`, `atomicMin`, `atomicInc`, `atomicDec`, `atomicCAS`, `atomicOr`, `atomicAnd`, `atomicXor`

Математические функции

- * Есть функции, которые исполняются как на GPU так и на CPU; есть те, которые выполняются только на GPU
- * Вычисление может идти с погрешностью (см. документацию)
- * Точность указана в ULP -Unit in the Last Place или Unit of Least Precision
- * Время вычисления функций различно
- * Существуют быстрые аналоги функций, но с ограничениями на диапазон/точность

CUDA Utilities library

- * `#include <cutil.h>`
- * Не является частью CUDA
- * Назначение
 - * Разбор командной строки
 - * Чтение/запись бинарных файлов и изображений (PPM)
 - * Сравнение массивов данных
 - * Таймеры
 - * Макросы проверки ошибок/инициализации
 - * Проверка конфликтов банков разделяемой памяти

CUDA Utilities library

- * `CUT_DEVICE_INIT(ARGC, ARGV)`
- * `CUT_EXIT(ARGC, ARGV)`
- * `CUDA_SAFE_CALL(call)` — в режиме отладки
- * `CUT_BANK_CHECKER(array, index)` — в режиме эмуляции + отладки
- * `cutCreateTimer(unsigned int* name);`
- * `cutDeleteTimer(unsigned int name);`
- * `cutStartTimer(const unsigned int name);`
- * `cutStopTimer(const unsigned int name);`
- * `cutGetTimerValue(const unsigned int name);`

Сценарий работы с GPU

1. Произвести выбор GPU
2. Выделить память под входные данные и результат
3. Произвести инициализацию памяти
4. Запустить ядро
5. Скопировать результат из памяти GPU в память центрального процессора.
6. При необходимости повторить п.п. 2-5