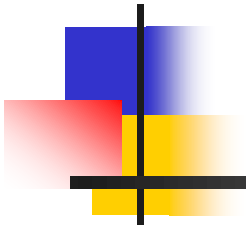
GRAPHICS & MEDIA LAB
VIDEO GROUP



Обзор OpenCL

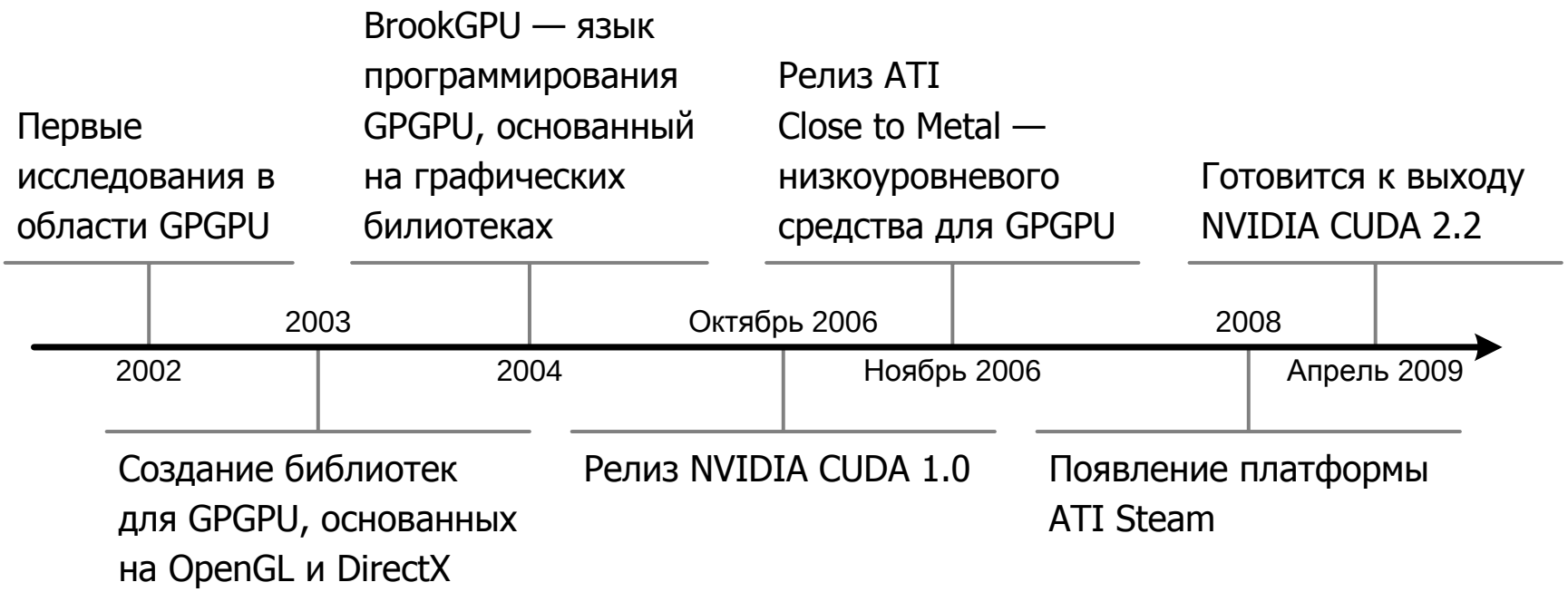
Илья Цветков

Video Group
CS MSU Graphics & Media Lab

Содержание

- **Введение**
- Модель OpenCL
  - Платформа OpenCL
  - Модель памяти
  - Модель вычислений
- Возможности OpenCL
  - Использование локальной памяти
  - Использование изображений
  - Средства языка
- Расширения OpenCL

Развитие GPGPU



Open Computing Language

- OpenCL предложен компанией Apple
- Сотрудничество с другими компаниями (AMD, Intel, NVIDIA)
- Стандартизацией занимается группа Khronos
- В декабре 2008 года выпущен стандарт OpenCL 1.0
  - http://www.khronos.org/
  - http://www.khronos.org/registry/cl/
- На данный момент реализации нет

Особенности OpenCL

- Использование всех вычислительных ресурсов системы
  - GPU, CPU и другие процессоры
  - Параллелизм на уровне данных и на уровне задач
- Эффективная модель параллельных вычислений
  - Абстрагирование от оборудования
  - Основан на языке C
- Специфицирует точность вычислений
  - Режимы округления IEEE 754
  - Определена точность встроенных функций
- Возможность добавления расширений

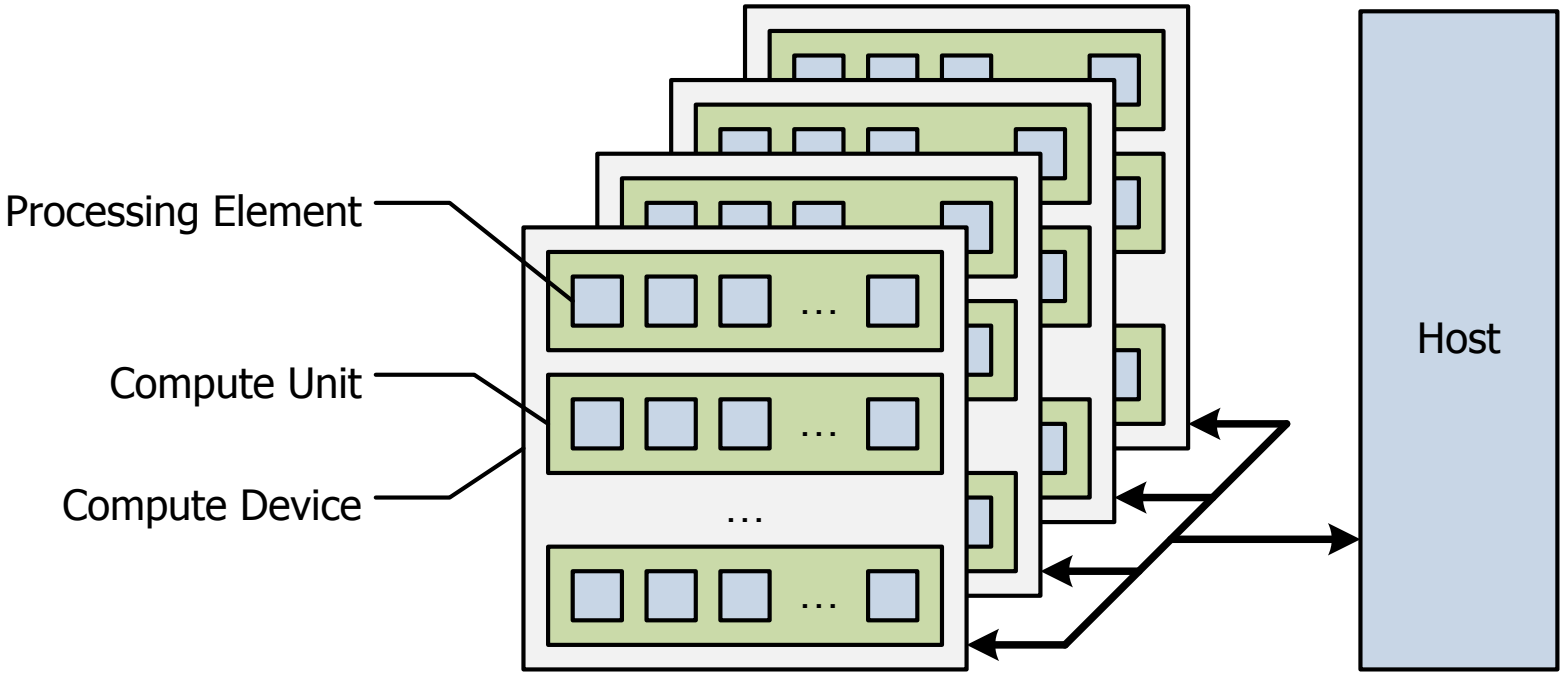
Содержание

- Введение
- **Модель OpenCL**
  - Платформа OpenCL
  - Модель памяти
  - Модель вычислений
- Возможности OpenCL
- Расширения OpenCL

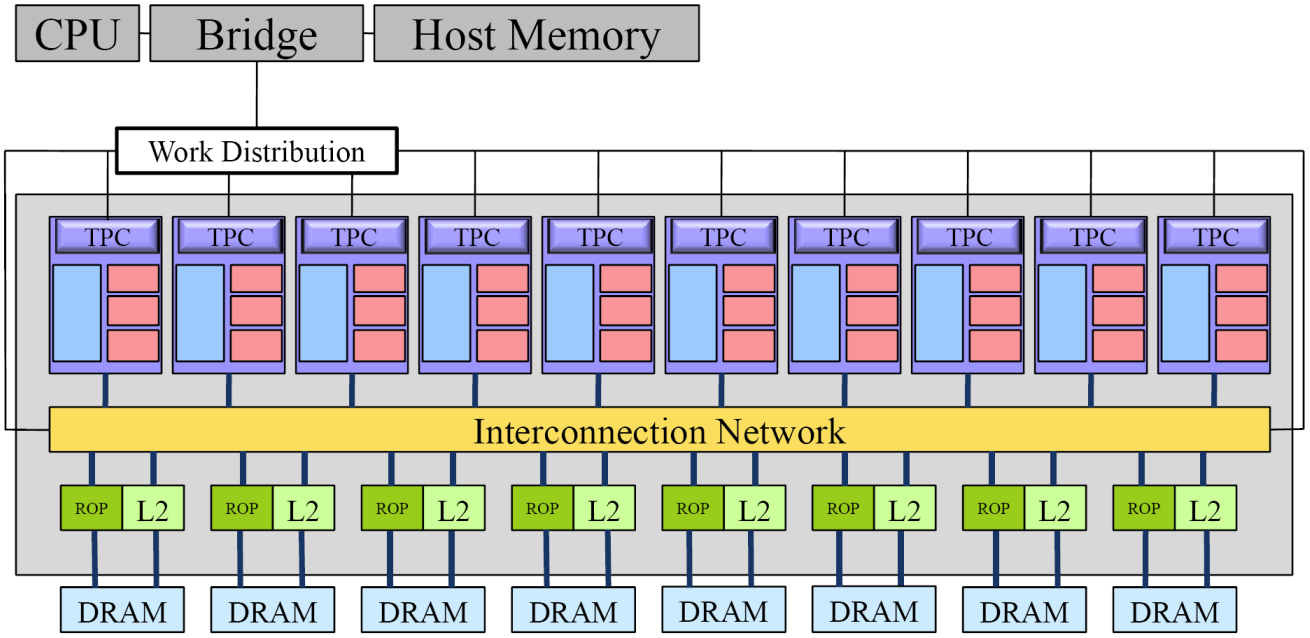
Программный стек

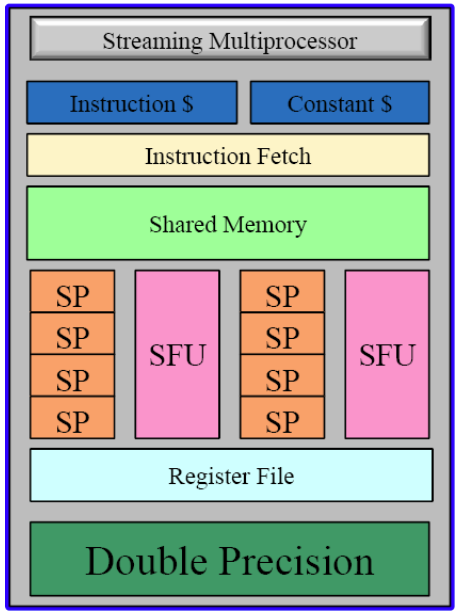
- Platform
  - Определение устройств в системе
  - Инициализация устройств
  - Создание контекста и очередей задач
- Runtime
  - Управление ресурсами
  - Исполнение вычислительных ядер
- Compiler
  - Подмножество языка C с расширениями
  - Компиляция вычислительных ядер

Платформа OpenCL



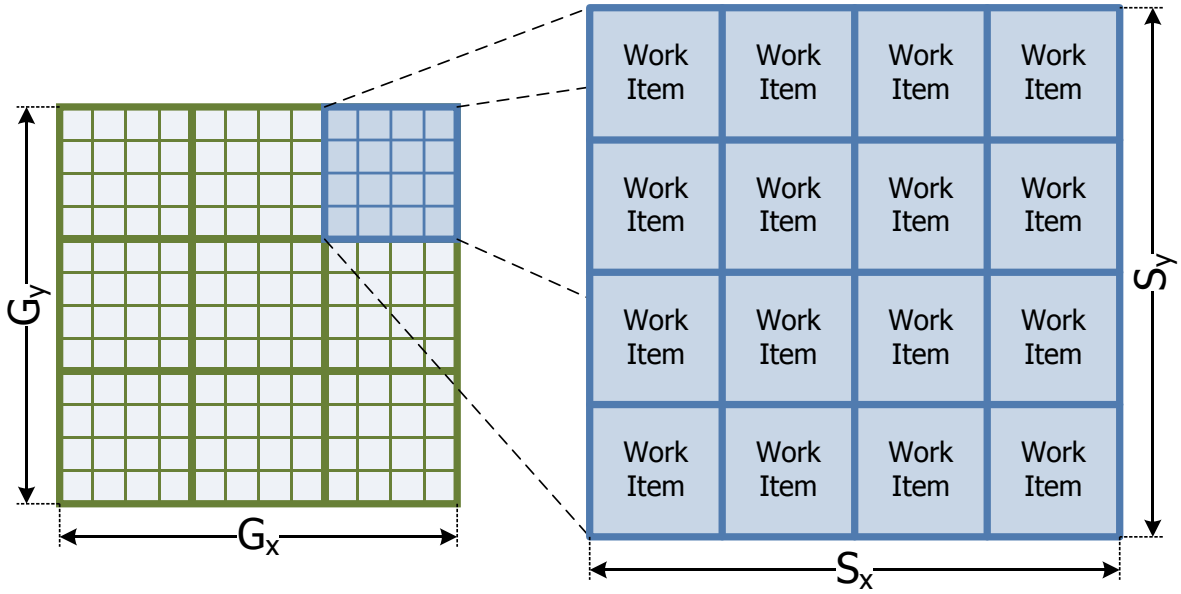
Пример: NVIDIA GT200





- 1 compute device
- 30 compute units
- 240 processing elements

Модель вычислений




```
for (int x = 0; x < gx; ++x)
    for (int y = 0; y < gy; ++y)
        runKernel(x, y);
```

Простейший пример

- Код на C:

```
int nIndex;
for (nIndex = 0; nIndex < gx; ++nIndex)
{
    c[nIndex] = a[nIndex] + b[nIndex];
}
```

- Ядро OpenCL:

```
__kernel void
vectorAdd(__global const float * a,
          __global const float * b,
          __global float * c)
{
    int nIndex = get_global_id(0);
    c[nIndex] = a[nIndex] + b[nIndex];
}
```

Выполнение ядра

```
cl_int clEnqueueNDRangeKernel(
    cl_command_queue command_queue,    // Очередь команд
    cl_kernel kernel,                  // Ядро
    cl_uint work_dim,                  // Размерность пространства
    const size_t *global_work_offset,
    const size_t *global_work_size,    // Размер индексного пространства
    const size_t *local_work_size,     // Размер группы
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,   // Ожидаемые события
    cl_event *event)
```

Для приведенного примера:

```
clEnqueueNDRangeKernel(hQueue, hKernel, 1, NULL,
                       &gx, NULL, NULL,
                       NULL, NULL);
```

Ядро

```
__kernel void
vectorAdd(__global const float * a,
          __global const float * b,
          __global float * c)
{
    int nIndex = get_global_id(0);
    c[nIndex] = a[nIndex] + b[nIndex];
}
```

Инициализация контекста

```
// Создание контекста OpenCL
cl_context hContext;
hContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU,
                                   0, 0, 0);

// Получение списка доступных в контексте устройств
size_t nContextDescriptorSize;
clGetContextInfo(hContext, CL_CONTEXT_DEVICES,
                 0, 0, &nContextDescriptorSize);
cl_device_id * aDevices = malloc(nContextDescriptorSize);
clGetContextInfo(hContext, CL_CONTEXT_DEVICES,
                 nContextDescriptorSize, aDevices, 0);

// Создание очереди команд для первого из устройств
cl_command_queue hQueue;
hQueue = clCreateCommandQueue(hContext, aDevices[0], 0, 0);
```

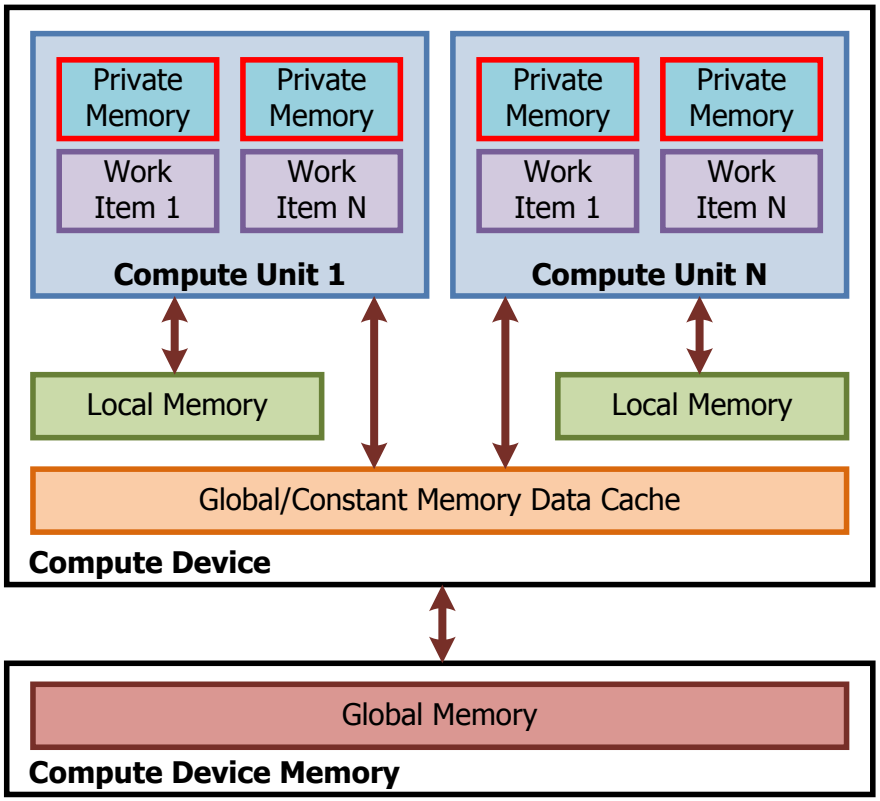
Инициализация памяти

```
// Выделение памяти на устройстве
cl_mem hDevMemA, hDevMemB, hDevMemC;
hDevMemA = clCreateBuffer(hContext,
                          CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                          dataSize * sizeof(cl_float), pA, 0);
hDevMemB = clCreateBuffer(hContext,
                          CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                          dataSize * sizeof(cl_float), pB, 0);
hDevMemC = clCreateBuffer(hContext,
                          CL_MEM_WRITE_ONLY,
                          dataSize * sizeof(cl_float), 0, 0);
```

Выполнение ядра

```
// Компиляция OpenCL-программы
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, 1, sProgramSource, 0, 0);
clBuildProgram(hProgram, 0, 0, 0, 0, 0);
// Создание ядра
cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, "vectorAdd", 0);
// Установка параметров ядра
clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&hDevMemA);
clSetKernelArg(hKernel, 1, sizeof(cl_mem), (void *)&hDevMemB);
clSetKernelArg(hKernel, 2, sizeof(cl_mem), (void *)&hDevMemC);
// Выполнение ядра
clEnqueueNDRangeKernel(hQueue, hKernel, 1, 0, &dataSize, 0, 0, 0, 0);
// Чтение результатов в оперативную память
clEnqueueReadBuffer(hContext, hDevMemC, CL_TRUE, 0,
                    dataSize * sizeof(cl_float), pC, 0, 0, 0);
```

Организация памяти



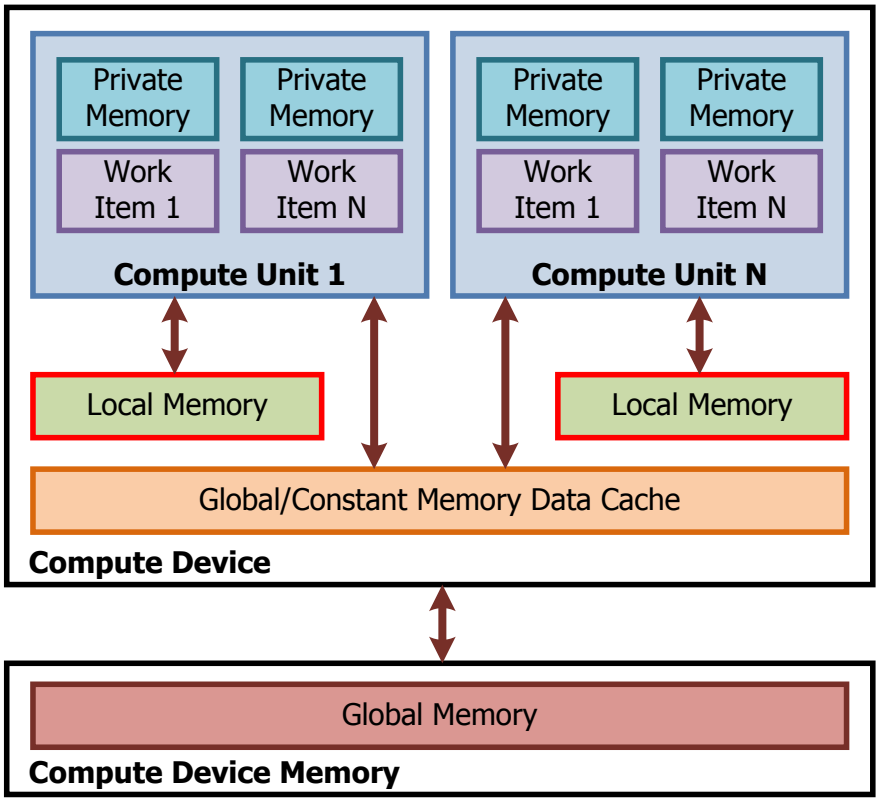
Private Memory

- Выделяется компилятором
- Чтение и запись для work-item
- Высокая скорость доступа

Для платформы CUDA:

- Аналог — регистровый файл
- Около 32 регистров на work-item

Организация памяти



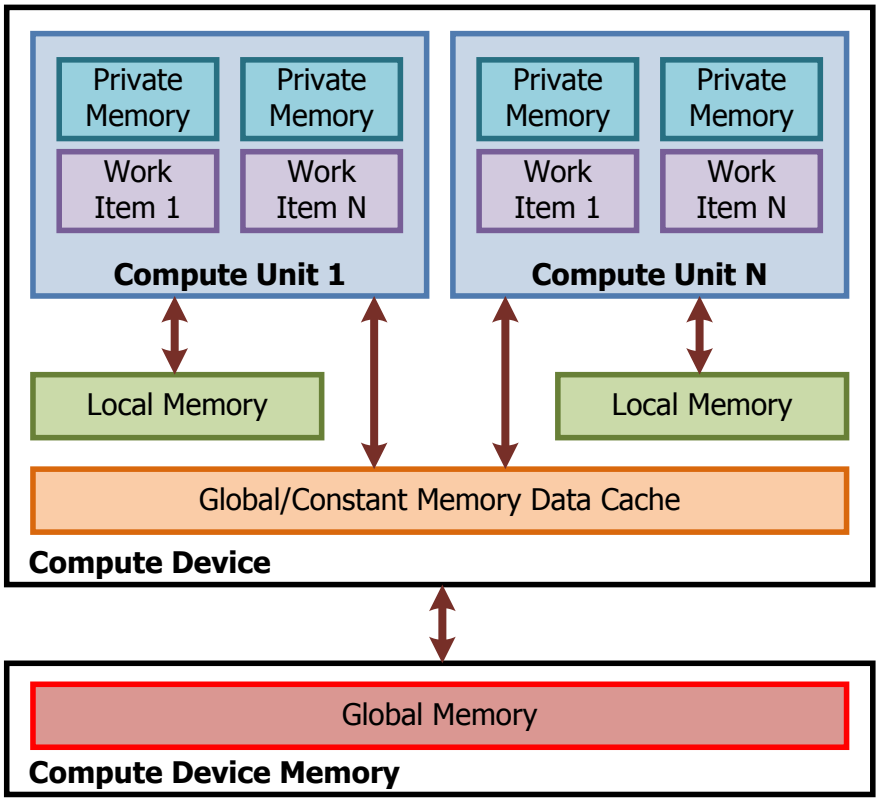
Local Memory

- Чтение и запись для всех work-item одной группы
- Высокая скорость доступа
- Необходима синхронизация

Для платформы CUDA:

- Аналог — разделяемая память
- Около 8 КБ на work-group (на compute unit)

Организация памяти



Global Memory

- Чтение и запись
- Низкая скорость доступа
- Необходима синхронизация

Для платформы CUDA:

- Аналог — глобальная память
- Типичный объём — 1 ГБ

Работа с памятью

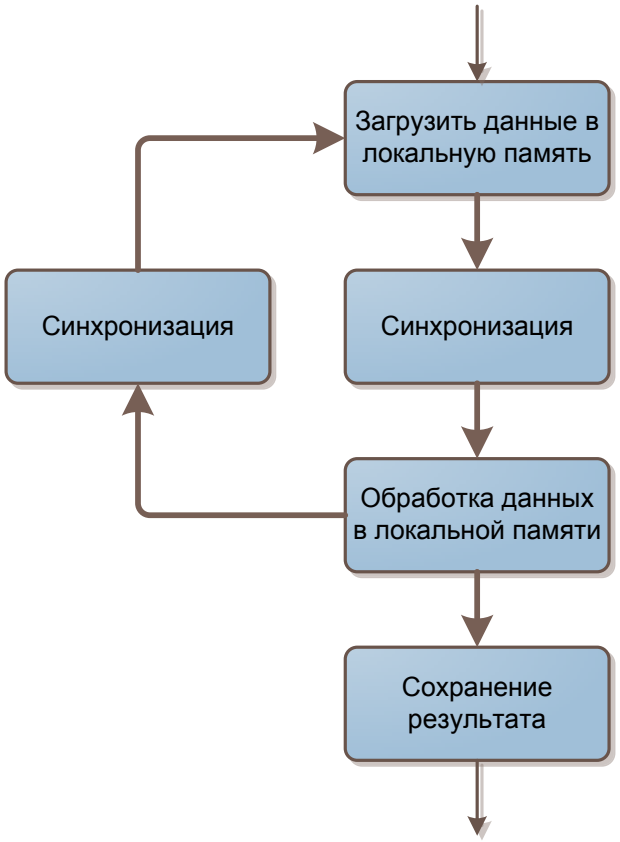
	Global	Constant	Local	Private
Host	Динамическое выделение	Динамическое выделение	Динамическое выделение	—
	Чтение и запись	Чтение и запись	Нет доступа	Нет доступа
Kernel	—	Статическое выделение	Статическое выделение	Статическое выделение
	Чтение и запись	Только чтение	Чтение и запись	Чтение и запись

Содержание

- Введение
- Модель OpenCL
- **Возможности OpenCL**
  - Использование локальной памяти
  - Использование изображений
  - Средства языка
- Расширения OpenCL

Использование локальной памяти

- Проблемы глобальной памяти
  - Большая латентность
  - Отсутствие кэша
- Решение
  1. Предварительная загрузка данных в локальную память
  2. Обработка
  3. Сохранение результата



Синхронизация

Точка синхронизации:

```
void barrier(cl_mem_fence_flags flags)
```

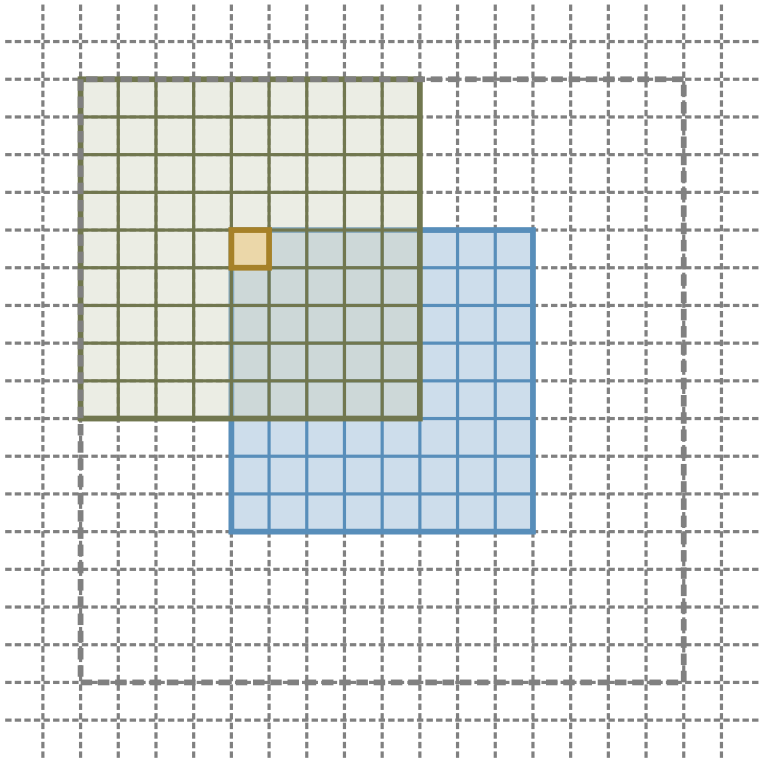
Особенности:

- Точка синхронизации должна быть достигнута либо всеми потоками, либо ни одним из потоков в группе
- Флаг является подсказкой компилятору:
  - CLK_LOCAL_MEM_FENCE
  - CLK_GLOBAL_MEM_FENCE

Пример: свёртка

Свёртка с ядром 9×9

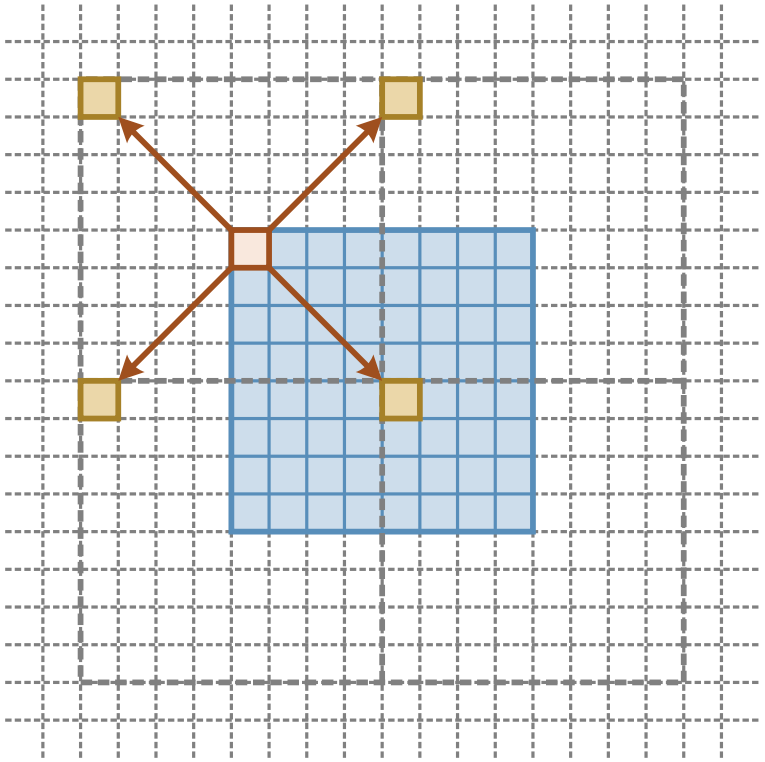
- Сложности:
  - Много обращений к глобальной памяти
  - Мало вычислений
- Реализация:
  - Разбиение на группы 8×8
  - Каждая группа работает в локальной памяти



Пример: свёртка

Свёртка с ядром 9×9

- Реализация:
  - В локальную память загружается блок 16×16
  - Каждый поток загружает по 4 значения



```
#define BRD_SIZE  4
#define BLK_SIZE 16
#define GRP_SIZE  8
```

Реализация

```
__kernel void convolution(__global float *dst, __global const float *src,
                          __local float *block)
{
    int gx = get_global_id(0); int gy = get_global_id(1);
    int lx = get_local_id(0);  int ly = get_local_id(1);
    int bx = gx - BRD_SIZE;    int by = gy - BRD_SIZE;

    int sid = by * SRC_STRIDE + bx; int bid = ly * BLK_SIZE + lx;
    block[bid] = src[sid];
    block[bid + GRP_SIZE] = src[sid + GRP_SIZE];

    sid += GRP_SIZE * SRC_STRIDE; bid += GRP_SIZE * BLK_SIZE;
    block[bid] = src[sid];
    block[bid + GRP_SIZE] = src[sid + GRP_SIZE];

    barrier(CLK_LOCAL_MEM_FENCE);
    // Вычисления в локальной памяти
    dst[gy * SRC_STRIDE + gx] = result;
}
```

Функции для работы с памятью

GRAPHICS & MEDIA LAB
VIDEO GROUP

Функция	Описание

`event_t async_work_group_copy(`
`__local gentype *dst,`
`const __global gentype *src,`
`size_t num_elements,`
`event_t event)`
`event_t async_work_group_copy(`
`__global gentype *dst,`
`const __local gentype *src,`
`size_t num_elements,`
`event_t event)`
	Асинхронное копирование данных между локальной и глобальной памятью. Возвращает объект для синхронизации.

`void wait_group_events(`
`int num_events,`
`event_t *event_list)`
	Синхронизация по определённому событию

`void prefetch(`
`const __global gentype *p,`
`size_t num_elements)`
	Упреждающая загрузка данных в кэш

Текстуры в GPGPU

- Достоинства
  - Многомерное кэширование
  - Обработка обращений за границы изображения
  - Интерполяция на лету
- Недостатки
  - Нормализованные данные
  - Нормализованные координаты

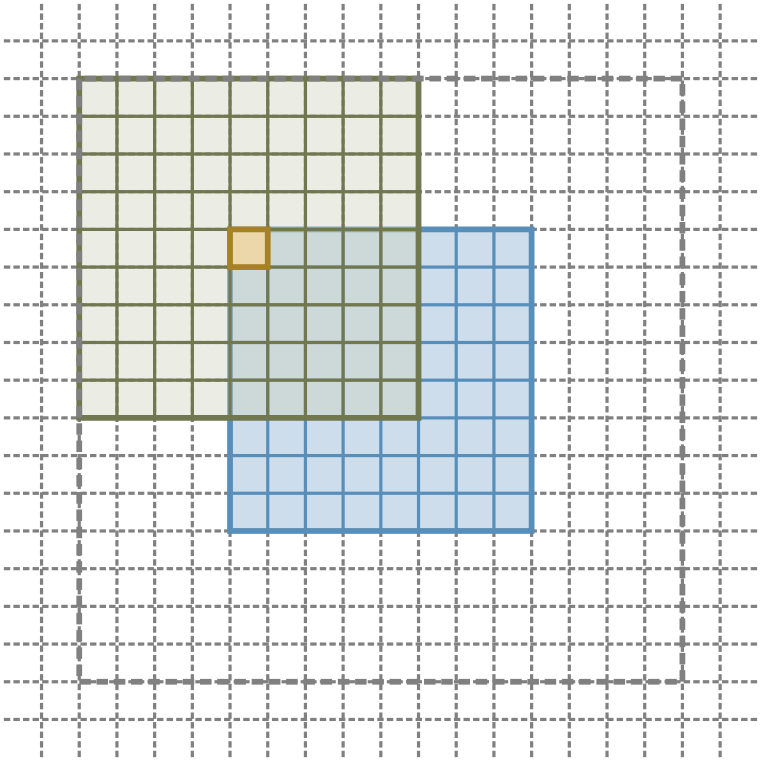
Изображения в OpenCL

- Обладают достоинствами текстур
- Нормализация координат и данных опциональна
- Доступные форматы — от 1 до 4 компонент на пиксель:
  - CL_SIGNED_INT8
  - CL_SIGNED_INT16
  - CL_SIGNED_INT32
  - CL_HALF_FLOAT
  - CL_FLOAT
- Функции для работы с изображениями:
  - `float4 read_imagef(image2d_t image, sampler_t sampler, int2 coord)`
  - `void write_imagef(image2d_t image, int2 coord, float4 color)`
- Возможна работа с 3D-изображениями

Использование изображений

Свёртка 9×9

- Проблемы:
  - Обработка краёв
  - Перекрытие блоков
- Решение — изображения
  - Кэширование
  - Обработка адресации вне изображения



Пример

```
__kernel void convolution(__write_only image2d_t *dst, __read_only image2d_t *src,
                          __local float *block)
{
    const sampler_t smplr = CLK_NORMALIZED_COORDS_FALSE |
                            CLK_ADDRESS_CLAMP_TO_EDGE | CLK_FILTER_NEAREST;
    int gx = get_global_id(0); int gy = get_global_id(1);
    int lx = get_local_id(0);  int ly = get_local_id(1);
    int bx = gx - BRD_SIZE;    int by = gy - BRD_SIZE;
    int bid = ly * BLK_SIZE + lx;
    block[bid] = read_imagef(src, smplr, (int2)(bx, by)).x;
    block[bid + GRP_SIZE] = read_imagef(src, smplr, (int2)(bx + GRP_SIZE, by)).x;
    bid += GRP_SIZE * BLK_SIZE;
    block[bid] = read_imagef(src, smplr, (int2)(bx, by + GRP_SIZE)).x;
    block[bid + GRP_SIZE] = read_imagef(src, smplr, (int2)(bx + GRP_SIZE,
                                                           by + GRP_SIZE)).x;
    barrier(CLK_LOCAL_MEM_FENCE);
    // Вычисления в локальной памяти
    write_imagef(dst, (int2)(gx, gy), result);
}
```

Типы данных

- Скалярные
  - `bool`
  - `char`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`
  - `float`
  - `size_t` и другие вспомогательные типы
- Векторные
  - Состоят из 2, 4, 8 или 16 элементов
  - Именуются `char`n, `uchar`n, `short`n, `ushort`n, `int`n, `uint`n, `long`n, `ulong`n, `float`n, где n — количество элементов
  - Доступ к элементам с помощью `.xyzw`, `.odd`, `.even`, `.lo`, `.hi`, `.s0123456789ABCDEF`
  - Пример

```
uchar16 a; uchar4 b = a.s6789;
float4 c; float4 d = c.zyxw;
```

Функции

- Математические функции из `math.h` языка C
- Целочисленные вычисления
  - `ugentype abs(gentype x)`
  - `ugentype abs_diff(gentype x, gentype y)`
  - `gentype hadd(gentype x, gentype y)` — $(x + y) \gg 1$
  - `gentype rhadd(gentype x, gentype y)` — $(x + y + 1) \gg 1$
- Вспомогательные функции для работы с `float`
  - `gentype clamp(gentype x, gentype minval, gentype maxval)`
  - `gentype {min|max}(gentype x, gentype y)`
  - `gentype mix(gentype x, gentype y, gentype a)` — $x + (y - x) \cdot a$
  - `gentype step(gentype edge, gentype x)`
- Геометрические функции
- Функции чтения и записи изображений
- Определение индексов work-item

Определение индексов

Функция	Описание
`uint get_work_dim()`	Размерность индексного пространства
`size_t get_global_size(uint ind)`	Количество work-item в измерении `ind`
`size_t get_global_id(uint ind)`	Индекс work-item в измерении `ind`
`size_t get_local_size(uint ind)`	Размер work-group в измерении `ind`
`size_t get_local_id(uint ind)`	Локальный индекс work-item внутри work-group в измерении `ind`
`size_t get_num_groups(uint ind)`	Количество work-group в измерении `ind`
`size_t get_group_id(uint ind)`	Индекс work-group в измерении `ind`

Содержание

- Введение
- Модель OpenCL
- Возможности OpenCL
- **Расширения OpenCL**

Ограничения в OpenCL 1.0

- Не поддерживается рекурсия
- Минимальный объём записываемых данных — 32 бита
- Отсутствуют вычисления с двойной точностью
- Не поддерживается запись в 3D-изображения
- Отсутствуют указатели на функции
- Указатели на указатели не могут быть параметрами
- Битовые поля не поддерживаются
- Не поддерживаются динамические структуры данных

Расширения OpenCL

Название	Описание
cl_khr_fp64	Вычисления с двойной точностью
cl_khr_select_fprounding_mode	Выбор режимов округления
cl_khr_global_int32_base_atomics, cl_khr_global_int32_extended_atomics	Атомарные функции для работы с 32-битыми целыми в глобальной памяти
cl_khr_local_int32_base_atomics, cl_khr_local_int32_extended_atomics	Атомарные функции для работы с 32-битыми целыми в локальной памяти
cl_khr_int64_base_atomics, cl_khr_int64_extended_atomics	Атомарные функции для работы с 64-битыми целыми в глобальной и локальной памяти
cl_khr_3d_image_writes	Запись в 3D-изображение
cl_khr_byte_addressable_store	Запись с побайтовой адресацией
cl_khr_fp16	Вычисления с половинной точностью

Атомарные функции

- Типы памяти:
  - Локальная
  - Глобальная
- Типы данных:
  - 32-разрядные целые
  - 64-разрядные целые
- Набор функций:
  - Базовый
    - `int atom_{add|sub|xchg}(int *p, int val)`
    - `int atom_{inc|dec}(int *p)`
    - `int atom_cmpxchg(int *p, int cmp, int val)`
  - Расширенный
    - `int atom_{min|max}(int *p, int val)`
    - `int atom_{and|or|xor}(int *p, int val)`

Выводы

- Позволяет распараллеливать задачи обработки изображений и видео
- Поддержка большого количества устройств
  - GPU
  - CPU
  - PS3, XBOX
  - Embedded profile
- Один код для всех устройств
- Сходство с CUDA
- Реализации ожидаются в ближайшее время
  - NVIDIA
  - AMD
  - Intel

Список литературы

1. http://www.khronos.org/registry/cl/
2. http://www.nvidia.com/object/cuda_opencl.html
3. http://www.nvidia.com/object/cuda_home.html

Вопросы

?

Лаборатория компьютерной графики и мультимедиа

Видеогруппа это:

- Выпускники в аспирантурах Англии, Франции, Швейцарии (в России в МГУ и ИПМ им. Келдыша)
- Выпускниками защищено 5 диссертаций
- Наиболее популярные в мире сравнения видеокодеков
- Более 3 миллионов скачанных фильтров обработки видео