

# OpenCL

## Лекторы:

[Боресков А.В. \(ВМиК МГУ\)](#)

[Харламов А.А. \(NVidia\)](#)

# Основы OpenCL

OpenCL - открытый кроссплатформенный стандарт для параллельных вычислений на гетерогенных устройствах

Изначально разработан Apple, сейчас поддерживается Khronos Group,

Первая версия (1.0) - конец 2008 года

Поддерживает GPU, CPU, Cell, DSP и многие другие устройства

# Основы OpenCL

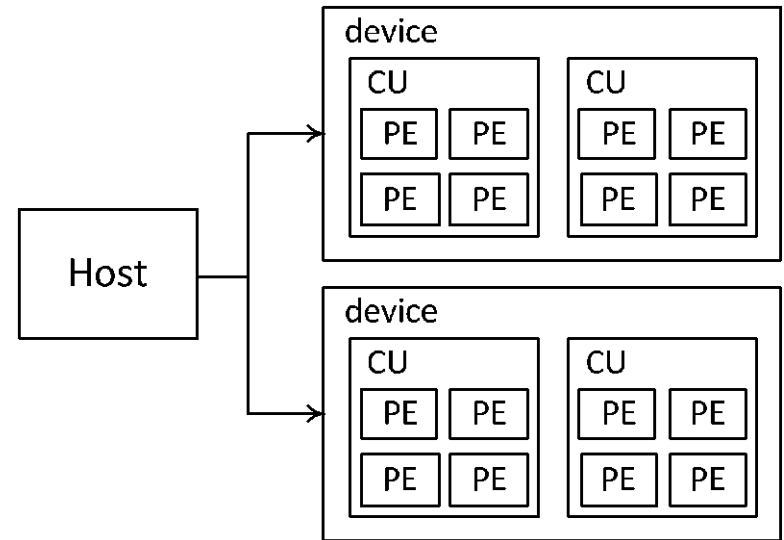
Основан на следующих обобщенных моделях

- Platform model
- Memory model
- Execution model
- Programming model

# OpenCL platform model

Платформа состоит из хоста (CPU) и одного или нескольких вычислительных устройств

Каждое устройство состоит из вычислительных блоков (Compute Unit), которые состоят из Processing Elements (PE)



# OpenCL platform model

Получение списка доступных платформ

```
cl_int clGetPlatformIDs (  
    cl_uint numEntries,  
    cl_platform_id * platforms,  
    cl_uint * numPlatforms );
```

# Получение списка доступных платформ

```
cl_platform_id  platform;
cl_device_id    device;
cl_uint         err ;
err = clGetPlatformIDs ( 1 , &platform , NULL ) ;
if ( err != CL_SUCCESS )
{
    p r i n t f ( "Error obtaining OpenCL platform. \n" ) ;
    return -1;
}
err = clGetDeviceIDs ( platform , CL_DEVICE_TYPE_GPU, 1 , &device , NULL ) ;
if ( err != CL_SUCCESS )
{
    p r i n t f ( "Error obtaining OpenCL device. \n" ) ;
    return -1;
}
```

# Получение информации о платформе

```
cl_int clGetPlatformInfo ( cl_platform_id platform ,  
    cl_platform_info pname ,  
    size_t valueBufSize,  
    void * valueBuf,  
    size_t * valueSize );
```

# Получение устройств и информации об устройстве

```
cl_int clGetDeviceIDs (  
    cl_platform_id platform,  
    cl_device_type deviceType,  
    cl_uint numEntries,  
    cl_device_id * devices,  
    cl_uint * numDevices );
```

```
cl_int clGetDeviceInfo (  
    cl_device_id device,  
    cl_deviceInfo pname,  
    size_t valueBufSize,  
    void * valueBuf,  
    size_t * valueSize );
```

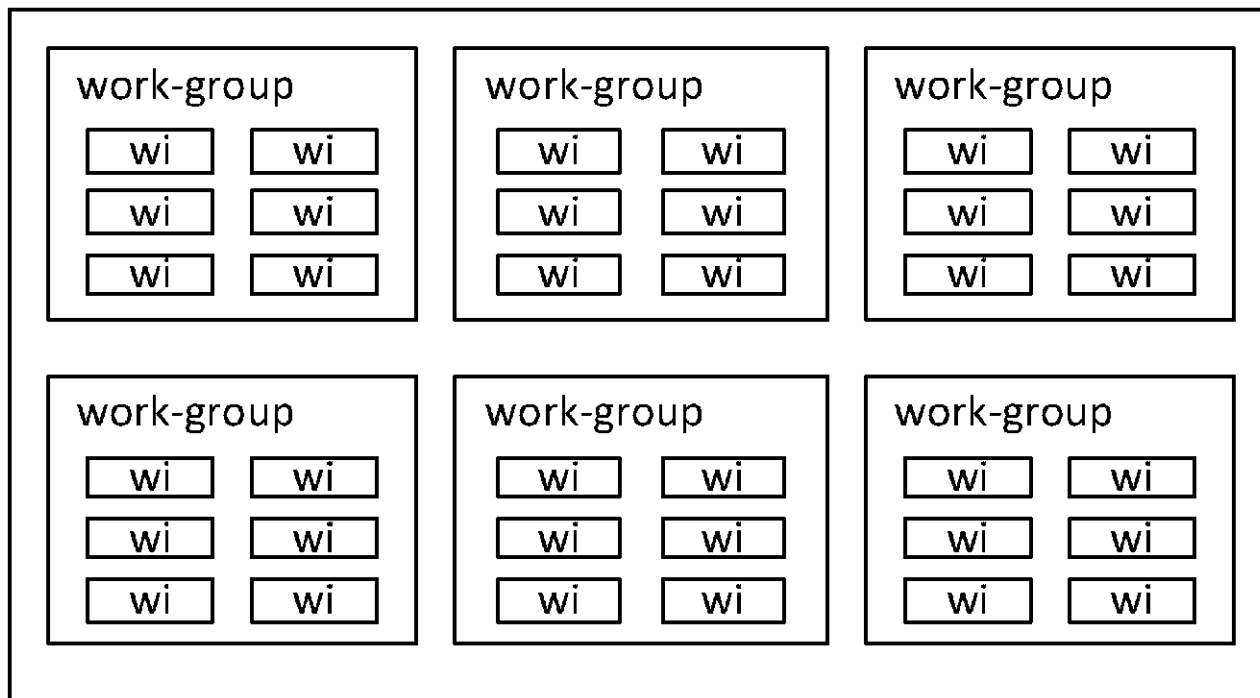


# Вычислительная модель OpenCL

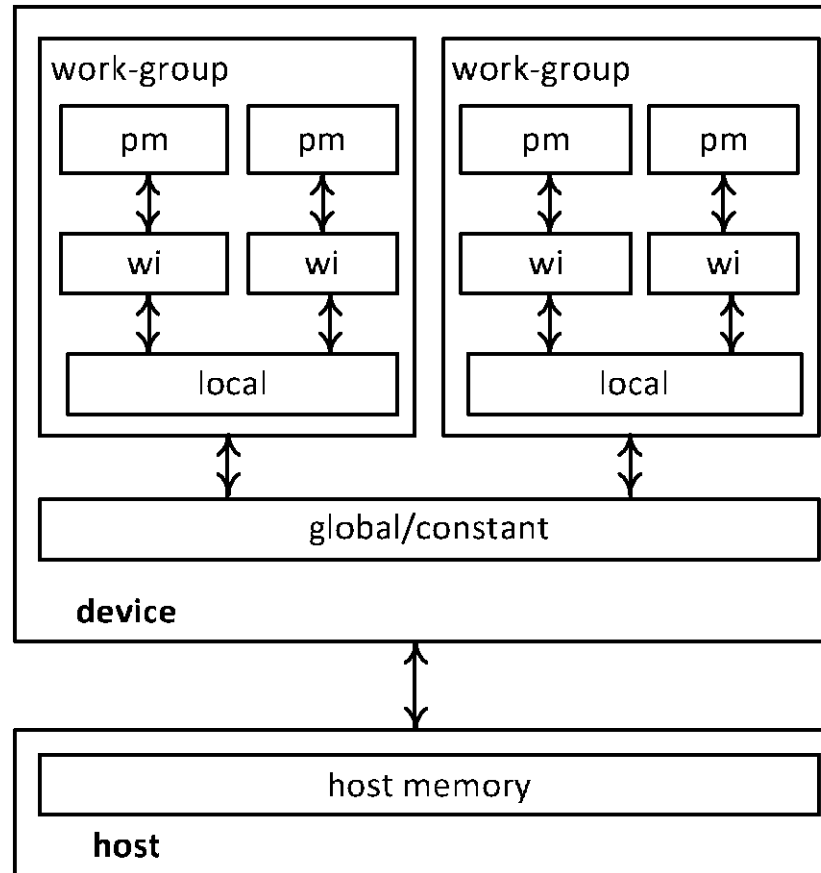
- Stream processing models
- Ядра (написанные на специальном основанном на C языке) запускаются на устройстве
- Ядро запускается для каждого элемента N-мерной вычислительной области (N=1,2,3) (ND-range) как work-item
- Work-item'ы группируются в work-group'ы

# Вычислительная модель OpenCL

NDRange



# OpenCL memory model



Типы памяти - global, constant, local, private

# Типы памяти в OpenCL

`__global` - как в CUDA

`__constant` - как в CUDA

`__local` - `shared` в CUDA

`__private` - как в CUDA

# Контекст OpenCL

Контекст OpenCL содержит в себе

- Устройства
- Ядра
- Объекты программ
- Memory objects
- Command queue

# Контекст OpenCL

```
cl_context clCreateContext (
    const cl_context_properties * props,
    cl_uint numDevices,
    const cl_device_id * devices,
    void (CL_CALLBACK * notify) ( const char * errInfo, const void *
        privateInfo, size_t cb, void * userData ),
    void * userData,
    cl_int * errCode );
```

# Command queue

Каждый девайс должен иметь свою очередь.

В очередь помещаются запросы на

- Выполнение ядра
- Операции с памятью
- Команды синхронизации

Помещаемые команды выполняются асинхронно. Могут выполняться in-order и out-of-order

# Command queue

```
cl_command_queue clCreateCommandQueue (  
    cl_context context,  
    cl_device_id device,  
    cl_command_queue_properties props,  
    cl_int * errCode );
```

## Props

- CL\_QUEUE\_OUT\_OF\_ORDER\_EXEC\_MODE\_ENABLE
- CL\_QUEUE\_PROFILING\_ENABLED



# Ядра

Пишутся на основанном на C99 языке из которого убраны

- Указатели на функции
- Битовые поля
- Массивы переменной длины
- Рекурсия
- Стандартные заголовки

Ядро помечается спецификатором  
`__kernel`

# Ядра

Добавлены новые типы (n=2,3,4,8,16)

charN, ucharN (на хосте cl\_charN,  
cl\_ucharN)

shortN, ushortN (cl\_shortN, cl\_ushortN)

intN, uintN (cl\_intN, cl\_uintN)

longN, ulongN (cl\_longN, cl\_ulongN)

floatN (cl\_floatN)

# Пример работы с типами

```
float4 f = (float4) (1.0f, 2.0f, 3.0f, 4.0f);  
float2 f1 = f.zx;  
uint4 u = (uint4) (0);  
float2 lo = f.lo;      // f.xy  
float2 ev = f.even;    // f.xz  
float ff = f.s3;      // f.z
```

# Получение информации в ядре

Функция	Что возвращает
<code>get_num_groups(idx)</code>	Размер ND-range в work-group'ах
<code>get_local_size(idx)</code>	Размер work-group в work-item'ах
<code>get_group_id(idx)</code>	Глобальный индекс work-group'ы
<code>get_local_id(idx)</code>	Локальный индекс work-item'а в текущей work-group'е
<code>get_global_id(idx)</code>	Глобальный индекс work-item'а в ND-range
<code>get_global_size(idx)</code>	Размер ND-range в work-item'ах

# Функции синхронизации

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

```
void mem_fence ( cl_mem_fence_flags flags );
```

```
void read_mem_fence ( cl_mem_fence_flags flags );
```

```
void write_mem_fence ( cl_mem_fence_flags flags );
```

# Пример ядра

```
__kernel void test ( __global float * a, int n )
{
    int idx = get_global_id ( 0 );
    if ( idx < n )
        a [idx] = sin ( idx * 3.1415926f / 1024.0f );
}
```

# Создание и компиляция программы

```
cl_program clCreateProgramWithSource (  
    cl_context context,  
    cl_uint count,  
    const char ** strings,  
    const size_t * lengths,  
    cl_int * errCode );
```

```
cl_int clBuildProgram (  
    cl_program program,  
    cl_uint numDevices,  
    const cl_device_id * devices,  
    const char * options,  
    void (*notify)(cl_program, void *),  
    void * userData );
```

# Создание ядра

```
cl_kernel clCreateKernel (  
    cl_program program,  
    const char * kernelName,  
    cl_int * errCode );
```



# Буфера

```
cl_mem clCreateBuffer (
    cl_context context, cl_mem_flags flags,
    size_t size , void * hostPtr,
    cl_int * errCode );

cl_int clEnqueueWriteBuffer (
    cl_command_queue queue, cl_mem buffer,
    cl_bool blockingWrite,
    size_t offset, size_t numBytes,
    const void * hostPtr,
    cl_uint numEvents, const cl_event * waitList,
    cl_event * event );
```

# Запуска ядра

```
cl_int clSetKernelArg ( cl_kernel kernel,  
    cl_int argIndex, size_t argSize,  
    const void * argPtr );
```

```
cl_int clEnqueueNDRangeKernel (  
    cl_command_queue queue,  
    cl_kernel kernel,  
    cl_uint workDim,  
    const size_t * globalWorkOffset,  
    const size_t * globalSize,  
    const size_t * localSize,  
    cl_uint numEvents, const cl_event * waitList,  
    cl_event * event );
```

# Ресурсы нашего курса

- [Steps3d.Narod.Ru](#)
- [Google Site CUDA.CS.MSU.SU](#)
- [Google Group CUDA.CS.MSU.SU](#)
- [Google Mail CS.MSU.SU](#)
- [Google SVN](#)
- [Tesla.Parallel.Ru](#)
- [Twirpx.Com](#)
- [Nvidia.Ru](#)