Введение в OpenCL

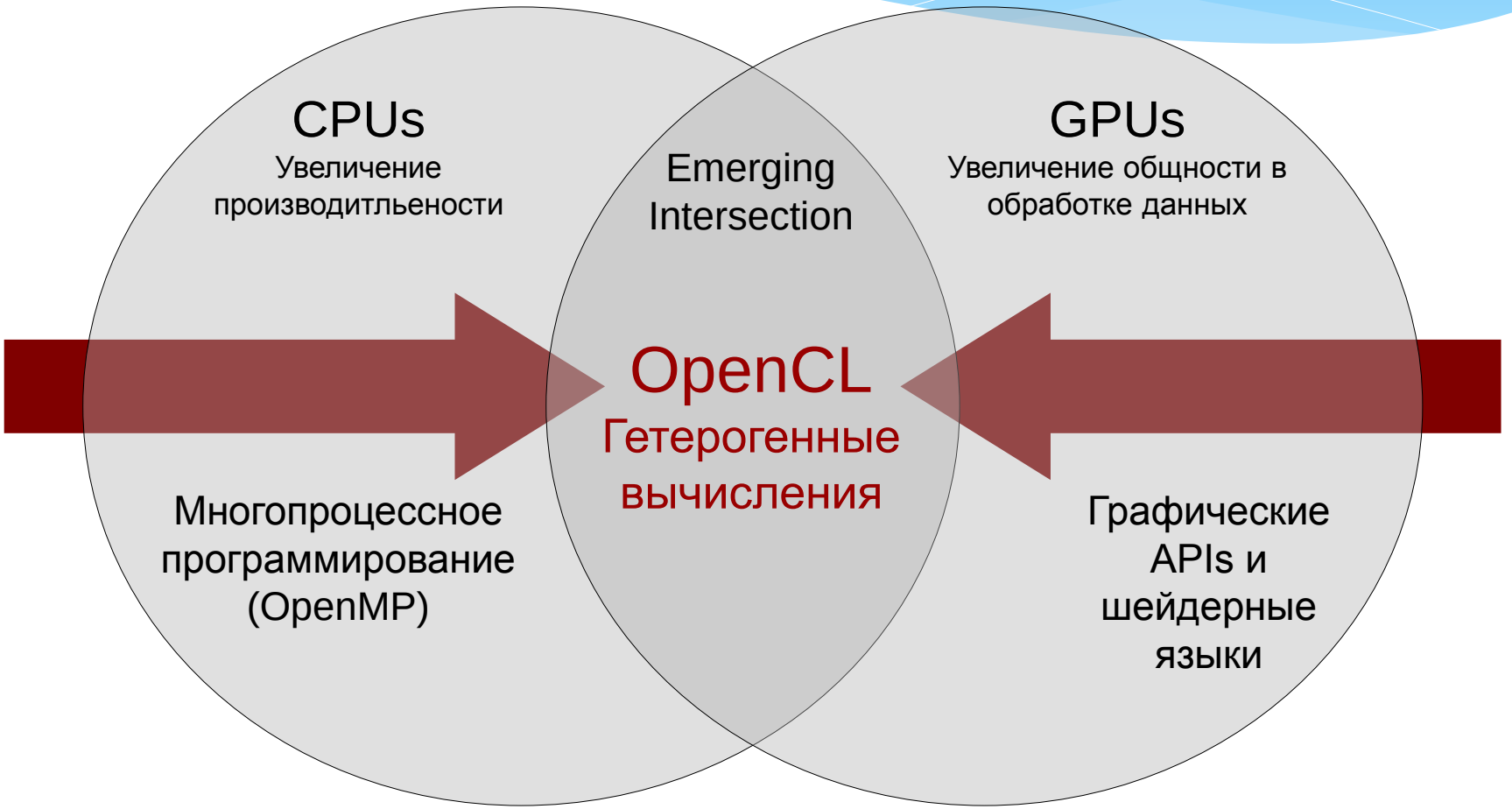
Романенко А.А.

arom@ccfit.nsu.ru

Новосибирский государственный университет

* OpenCL (Open Computing Language - открытый язык вычислений) представляет собой фреймворк для написания компьютерных программ, связанных с параллельными вычислениями на графических и центральных процессорах. OpenCL является полностью открытым стандартом, его использование не облагается лицензионными отчислениями.
* OpenCL разрабатывается и поддерживается некоммерческой организацией Khronos Group, в которую входят такие компании, как AMD, Intel, nVidia, Sun Microsystems, Apple и Sony Computer Entertainment.

Процессорный параллелизм



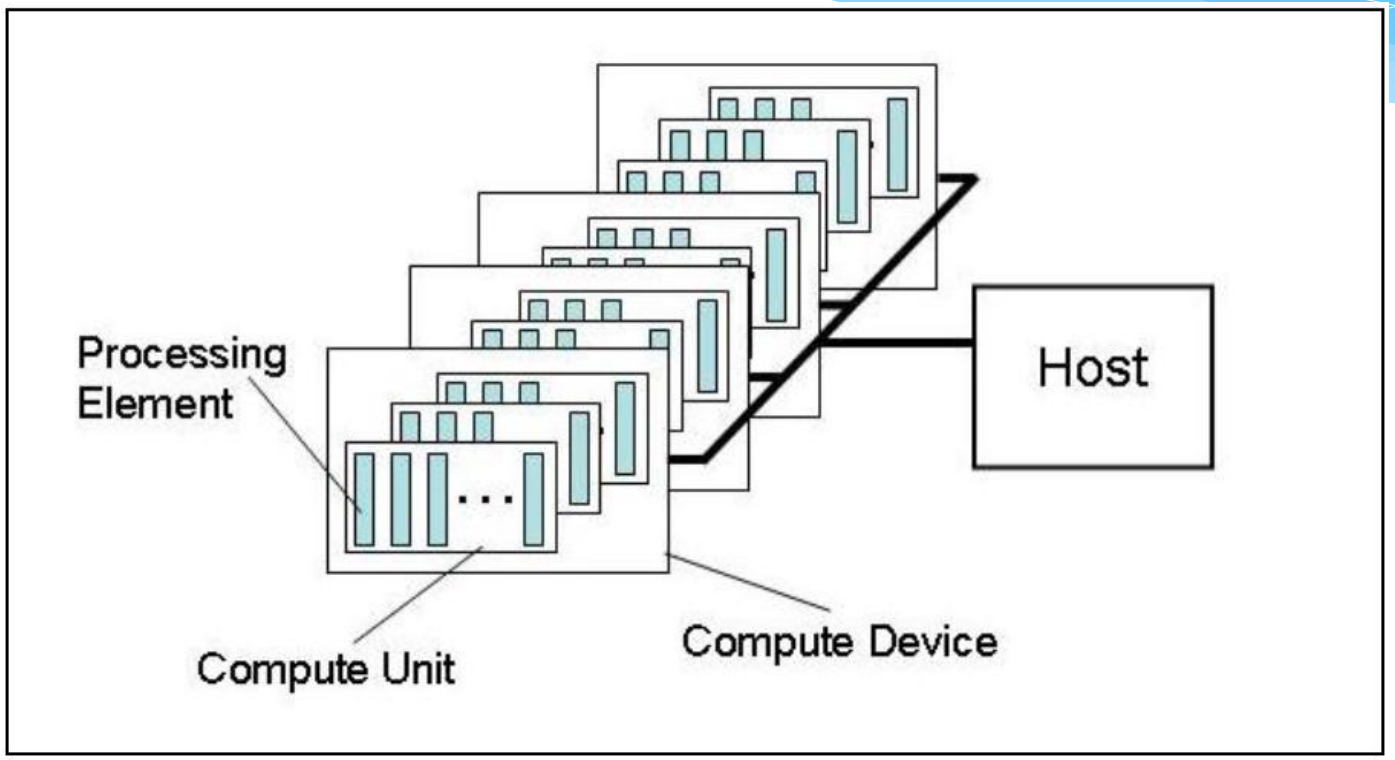
OpenCL Timeline

* 6 месяцев от подачи предложений до выпуска спецификации OpenCL 1.0
* Множество подтверждающих тестов по работе на разных архитектурах (GPU, CPU)
* Каждые 18 месяцев следующая версия спецификации

Модель OpenCL

* Платформа
* Модель памяти
* Модель выполнений
* Программная модель

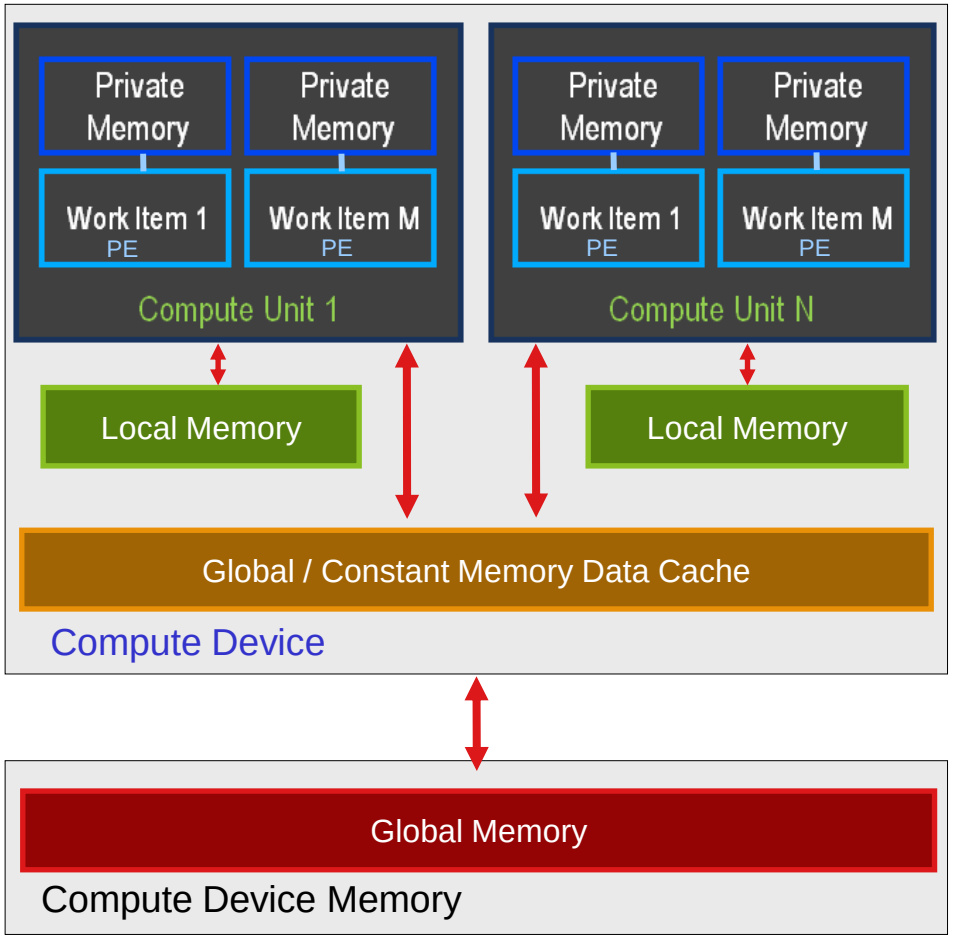
OpenCL Платформа



* Один **Host** + один или больше **Вычислительных устройств**
  * Каждое из **Вычислительных устройств** состоит из одного или большего числа **Вычислительных элементов**
    * Каждый из **Вычислительных элементов** в свою очередь разделен на несколько **Обрабатывающих элементов**

Модель памяти OpenCL

* Модель с разделяемой памятью
  * Relaxed consistency
* Множество различных адресных пространств
  * В зависимости от устройства могут объединяться
* Адресное пространство
  * Приветное – для одного работника
  * Локальное – для рабочей группы
  * Глобальное – доступно всем
  * Константное – всем тольео на чтение



Согласованость памяти (Section 3.3.1)

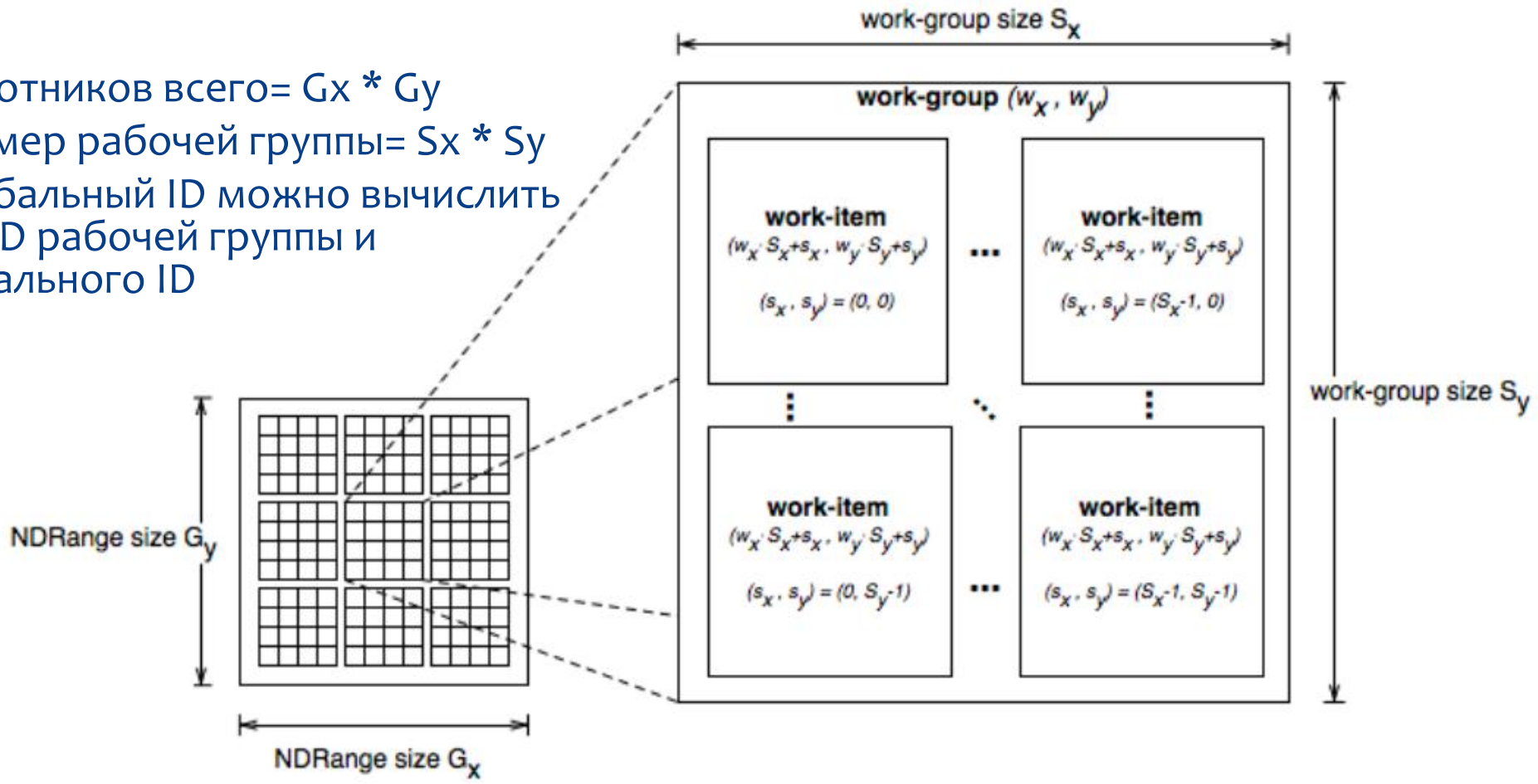
* В OpenCL используется “relaxed consistency memory model”
  * Состояние памяти, которое видит работник не обязательно является согласованым с состоянием памяти других работников
* Доступ к памяти согласован на операциях load/store в контексте одного работника
* Глобальная память является согласованой в рамках рабочей группы в точке синхронизации
* Глобальная память является согласованой в рамках рабочей группы в точке синхронизации, но не обязательно между разными рабочими группами
* Согласование памяти для объектов, разделяемых между командами, происходит принудительно в точке синхронизации

Модель выполнения

* Программа OpenCL:
  * Ядро – базовый элемент исполняемого кода — аналог функций в Си, CUDA ядра и пр.
  * Параллелизм по данным или по задачам
* Host программа
  * Набор вычислительных ядер и внутренних функций
* Исполнение ядер
  * Host программа вызывает ядро над индексным пространством NDRange
  * NDRange, “N-Dimensional Range”, может быть 1D, 2D или 3D
* Экземпляр вычислительного ядра в точке индексного пространства называтеся работником (work-item)
  * Каждый работник имеет уникальный глобальный идентификатор в индексном пространстве
* Работники групируются в рабочие группы (work-groups)
* Рабочая группа имеет уникальный идентификатор
* Работники внутри группы имеют уникальны идентификаторы

Исполнение ядра

* Работников всего= Gx * Gy
* Размер рабочей группы= Sx * Sy
* Глобальный ID можно вычислить из ID рабочей группы и локального ID



Контексты и очереди исполнения

* Контекст используется для хранения и управления состоянием “Вселенной”
* Ядра создаются и управляются Хостом в рамках конкретного контекста
  * Устройство
  * Ядро - OpenCL функция
  * Программный объект – исходный и бинарный код ядра
  * Объекты памяти
* Очередь команд – координирует исполнение ядер
  * Команды исполнения ядер
  * Команды работы с памятью: передача или отображение объектов данных на память
  * Команды синхронизации: задают порядок выполнения команд
* Очередь приложений производит запуск вычислительных ядер
  * Выбираются в порядке размещения
  * Исполнение либо в порядке очереди или без него
  * События могут использоваться для синхронизации исполнения

Модель программирования

* Параллелизм по данным
* Обязательно поддерживается всеми OpenCL вычислительными устройствами
* Определяется N- мерный вычислительный домен
  * Каждый независимый обрабатываемый элемент домена - работник
  * В каждом домене определяется количество работников = global work size
* Работники могут группироваться
  * В рамках группы работники могут взаимодействовать
  * Возможна синхронизация для упорядочивания доступа к памяти
* Рабочие группы исполняются параллельно
* Разбиение на группы может быть как явным, так и неявным

Модель программирования

* Параллелизм по данным
* Не все устройства поддерживают данный тип параллелизма

* Исполнение в виде отдельного работника
  * Вычислительное ядро на OpenCL
  * Поддержка нативных C / C++ функций

Основные программные структуры OpenCL

- Host программа
- Выбор вычислительного устройства

Runtime

- Создание контекстов
- Выделение памяти
- Компилирование вычислительных ядер
- Постановка задач в очередь исполнения

Platform Layer

- Синхронизация
- Освобождение ресурсов OpenCL
- Ядра
  - Язык C/C++ с некоторыми ограничениями и расщирениями

Language

Ограничения OpenCL C

* Запрет указателей на функции
* Указатель на указатель возможен только внутри ядра, но не как аргумент
* Не поддерживаются битовые поля
* Нет поддержки структур и массивов переменной длинны
* Нет поддержки рекурсии
* Запись по указателю типа с размером меньше чем 32 бита не поддерживается
* Тип данных Double не поддерживается, но зарезервирован
* Запись в 3D картинки на поддерживается

* Некоторые ограничения можно обойти через расширения

OpenCL vs. CUDA

* **Код ядра на CUDA C:**

```
__global__ void
vectorAdd(const float * a, const float * b, float * c){
    // Vector element index
    int nIndex = blockIdx.x * blockDim.x + threadIdx.x;
    c[nIndex] = a[nIndex] + b[nIndex];
}
```

* **Код ядра на OpenCL**

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

Размеры групп и сети в OpenCL

* get_local_id()
* get_work_dim()
* get_global_size()
* get_global_id()

OpenCL vs. CUDA. Инициализация

* **CUDA**

```
cuInit(0);
cuDeviceGet(&hDevice, 0);
cuCtxCreate(&hContext, 0, hDevice);
```

* **OpenCL**

```
cl_context hContext;
hContext = clCreateContextFromType(0, CL_DEVICE_TYPE_GPU,
                                   0, 0, 0);
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);

cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, aDevices[0], 0, 0);
```

OpenCL vs. CUDA. Создание ядра

* CUDA

```
CUmodule hModule;
cuModuleLoad(&hModule, “vectorAdd.cubin”);
cuModuleGetFunction(&hFunction, hModule, "vectorAdd");
```

* OpenCL

```
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, 1,
                                     sProgramSource, 0, 0);
clBuildProgram(hProgram, 0, NULL, NULL, NULL, NULL);

cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, “vectorAdd”, 0);
```

OpenCL vs. CUDA. Выделение памяти

* CUDA

```
CUdeviceptr pDevMemA, pDevMemB, pDevMemC;
int size = cnDimension * sizeof(float);
cuMemAlloc(&pDevMemA, size );
cuMemAlloc(&pDevMemB, size );
cuMemAlloc(&pDevMemC, size );
// copy host vectors to device
cuMemcpyHtoD(pDevMemA, pA, size );
cuMemcpyHtoD(pDevMemB, bB, size );
```

OpenCL vs. CUDA. Выделение памяти

* OpenCL

```
cl_mem hDevMemA, hDevMemB, hDevMemC;
hDevMemA = clCreateBuffer(hContext,
                  CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                  cnDimension * sizeof(cl_float),
                  pA,
                  0);
hDevMemB = clCreateBuffer(hContext,
                  CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
                  cnDimension * sizeof(cl_float),
                  pA,
                  0);
hDevMemC = clCreateBuffer(hContext,
                  CL_MEM_WRITE_ONLY,
                  cnDimension * sizeof(cl_float),0, 0);
```

OpenCL vs. CUDA. Параметры ядря

* CUDA (deprecated)
  * cuParamSeti(cuFunction, 0, pDevMemA);
  * cuParamSeti(cuFunction, 4, pDevMemB);
  * cuParamSeti(cuFunction, 8, pDevMemC);
  * cuParamSetSize(cuFunction, 12);
* OpenCL:
  * clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&hDevMemA);
  * clSetKernelArg(hKernel, 1, sizeof(cl_mem), (void *)&hDevMemB);
  * clSetKernelArg(hKernel, 2, sizeof(cl_mem), (void *)&hDevMemC);

OpenCL vs. CUDA. Запуск ядря

* CUDA (deprecated)

```
cuFuncSetBlockShape(cuFunction, cnBlockSize, 1, 1);
cuLaunchGrid      (cuFunction, cnBlocks, 1);
```

* CUDA

```
void *config[] = {
        CU_LAUNCH_PARAM_BUFFER_POINTER,  argBuffer,
        CU_LAUNCH_PARAM_BUFFER_SIZE,         &argBufferSize,
        CU_LAUNCH_PARAM_END
};
status = cuLaunchKernel(f, gx, gy, gz, bx, by, bz, sh, s, NULL, config);
```

* OpenCL

```
clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0, &cnDimension,
                                        &cnBlockSize, 0, 0, 0);
```

OpenCL vs. CUDA. Возврат результата

* CUDA

```
cuMemcpyDtoH((void*)pC, pDevMemC,
        cnDimension*sizeof(float));
```

* OpenCL

```
clEnqueueReadBuffer(hContext, hDeviceC, CL_TRUE, 0,
        cnDimension * sizeof(cl_float),  pC, 0, 0, 0);
```

Освобождение ресурсов

* OpenCL

```
clReleaseMemObject(hDevMemA);
clReleaseMemObject(hDevMemB);
clReleaseMemObject(hDevMemC);
free (aDevices);
clReleaseKernel (hKernel);
clReleaseProgram (hProgram);
clReleaseCommandQueue (hCmdQueue);
clReleaseContext (hContext);
```

Ресурсы OpenCL

* Khronos OpenCL Homepage
  http://www.khronos.org/opencl
* OpenCL 1.2 Specification
  http://www.khronos.org/registry/cl
* Онлайн документация
  http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/
* OpenCL at NVIDIA
  http://www.nvidia.com/object/cuda_opencl.html