



Казанский
федеральный
университет

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

API OpenCL

OpenCL C

- ▶ Стандарт ISO C99 с некоторыми ограничениями
- ▶ Расширения языка
 - Векторные типы
 - Функции для рабочих элементов/групп
 - Синхронизация
 - Спецификаторы размещения в памяти
 - Встроенные функции
 - Спецификатор `__kernel` – функция-ядро



Ограничения OpenCL C

- ▶ Нет указателей на функции
- ▶ Нет битовых полей
- ▶ Нет массивов переменной длины
- ▶ Рекурсия не поддерживается
- ▶ Нет стандартных заголовков
- ▶ Тип `double` опционален (поддерживается большинством реализаций)



Типы OpenCL

Скалярный тип	Векторный тип (n = 2, 4, 8, 16)	Тип для кода на хосте
char, uchar	charn, ucharn	cl_char<n>, cl_uchar<n>
short, ushort	shortn, ushortn	cl_short<n>, cl_ushort<n>
int, uint	intn, uintn	cl_int<n>, cl_uint<n>
long, ulong	longn, ulongn	cl_long<n>, cl_ulong<n>
float	floatn	cl_float<n>



Типы OpenGL

```
float4 f = (float4)(1.0f, 2.0f, 3.0f, 4.0f);  
uint4 u = (uint4)(1); // u будет (1, 1, 1, 1)  
float4 f = (float4)((float2)(1.0f, 2.0f), (float2)(3.0f, 4.0f));  
float4 f = (float4)(1.0f, 2.0f); // ошибка
```

```
float2 pos;  
pos.x = 1.0f;  
pos.y = 1.0f;  
pos.z = 1.0f ; // ошибка, только 2 компоненты!
```

```
float4 c;  
c.x = 1.0f;  
c.y = 1.0f;  
c.z = 1.0f;  
c.w = 1.0f;
```



Типы OpenGL

Векторные компоненты	Численные индексы
2 компоненты	0, 1
4 компоненты	0, 1, 2, 3
8 компонент	0, 1, 2, 3, 4, 5, 6, 7
16 компонент	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a/A, b/B, c/C, d/D, e/E, f/F

```
float8 f;  
f.s0 = 1.0f; // 1я компонента  
f.s7 = 1.0f; // 8я компонента  
  
float16 x;  
f.sa = 1.0f; // или f.sA, 10я компонента  
f.sF = 1.0f; // или f.sF, 16я компонента
```



Типы OpenCL

Суффикс доступа	Возвращаемые значения
.lo	Возвращает левую половину вектора
.hi	Возвращает правую половину вектора
.odd	Возвращает нечетные компоненты вектора
.even	Возвращает четные компоненты вектора

```
float4 f = (float4) (1.0f, 2.0f, 3.0f, 4.0f);  
float2 low, high;  
float2 o, e;
```

```
low = f.lo; // возвращает f.xy (1.0f, 2.0f)  
high = f.hi; // возвращает f.zw (3.0f, 4.0f)  
o = f.odd; // возвращает f.yw (2.0f, 4.0f)  
e = f.even; // возвращает f.xz (1.0f, 3.0f)
```



Типы OpenGL

- ▶ Для векторных типов поддерживается поэлементное выполнение основных операторов C: +, -, *, /, &, | и т.д.

```
int4 vi0, vi1; int v = 5;  
vi1 = vi0 + v;
```

```
//аналогично  
vi1.x = vi0.x + v;  
vi1.y = vi0.y + v;  
vi1.z = vi0.z + v;  
vi1.w = vi0.w + v;
```

```
float4 u(1), v(1), w(1);
```

```
w = u + v
```

```
//аналогично
```

```
w.x = u.x + v.x;
```

```
w.y = u.y + v.y;
```

```
w.z = u.z + v.z;
```

```
w.w = u.w + v.w;
```

```
w.odd = v.odd + u.odd;
```

```
//аналогично
```

```
w.y = v.y + u.y;
```

```
w.w = v.w + u.w;
```



Типы OpenCL

- ▶ Неявное преобразование скалярных типов и указателей
- ▶ Для векторных типов требуется явное преобразование
 - *convert_<dest_type>(source_type)*

```
int4 i;  
float4 f = convert_float4( i );
```



Типы OpenCL

- ▶ Явное преобразование общего вида
 - `convert_<dest_type><_sat><rounding>(source_type)`
- ▶ Два параметра
 - Характер округления
 - `_rte` – округление до ближайшего четного
 - `_rtz` – округление в сторону нуля
 - `_rtp` – округление в сторону $+\infty$
 - `_rtn` – округление в сторону $-\infty$
 - Поведение при выходе за границы типа
 - `_sat` – выходящие за пределы значения округляются до ближайших значений целевого типа; NaN в 0



Типы OpenCL

```
short4 s;  
//отрицательные числа округляются до 0  
ushort4 u = convert_ushort4_sat( s );  
  
float4 f;  
//поведение f > INT_MAX, f < INT_MIN и NaN зависит от реализации  
//округление - отбрасывается дробная часть  
int4 i = convert_int4( f );  
  
//f > INT_MAX к INT_MAX, f < INT_MIN к INT_MIN, f = NaN к 0  
//округление - отбрасывается дробная часть  
int4 i2 = convert_int4_sat( f );  
  
// так же как для i2 но округление до ближайшего целого числа  
int4 i3 = convert_int4_sat_rte( f );
```



Спецификаторы памяти

- ▶ `__global`
 - Объект размещается в глобальной памяти устройства
- ▶ `__local`
 - Объект размещается в быстрой локальной памяти
 - Общая память для всех элементов группы
- ▶ `__constant`
 - Память только для чтения в глобальной памяти
- ▶ `__private`
 - Быстрая память доступная одному элементу



Спецификаторы памяти

- ▶ Все аргументы функции-ядра располагаются в частной(*private*) памяти
- ▶ Указатели в аргументах функции обязаны быть объявлены со спецификаторами памяти *__global*, *__local* или *__constant*
- ▶ Присваивание указателя из одного адресного пространства другому запрещено
- ▶ Преобразование типа из одного адресного пространства в другое вызывает UB



Встроенные функции

```
// возвращает количество измерений пространства задачи
```

```
uint get_work_dim()
```

```
// возвращает общее количество элементов по направлению dimidx
```

```
size_t get_global_size(dimidx)
```

```
// возвращает общее число элементов в группе по направлению dimidx
```

```
size_t get_local_size(dimidx)
```

```
// возвращает уникальный номер элемента по направлению dimidx
```

```
size_t get_global_id(dimidx)
```



Встроенные функции

```
// возвращает уникальный номер элемента в группе по направлению  
dimidx
```

```
size_t get_local_id(dimidx)
```

```
// возвращает общее число групп по направлению dimidx
```

```
size_t get_num_groups(dimidx)
```

```
// возвращает уникальный номер группы по направлению dimidx
```

```
size_t get_group_id(dimidx)
```



Атомарные операции

- ▶ OpenCL поддерживает атомарные операции для типа `int`
 - `atomic_add`, `atomic_sub`, `atomic_inc`, `atomic_dec`
 - `atomic_or`, `atomic_and`, `atomic_xor`
 - `atomic_min`, `atomic_max`
 - `atomic_xchg`, `atomic_cmpxchg`
- ▶ Только `atomic_xchg` поддерживает тип `float`



C++ и OpenCL

- ▶ cl.hpp содержит обертку над API C в стиле C++
- ▶ Позволяет писать программу с использованием стандартных контейнеров и других инструментов C++
- ▶ Код легче читается
- ▶ Код медленнее чем «чистый C»*

*В некоторых случаях



C++ и OpenCL

```
//получение вектор платформ
std::vector<cl::Platform> platforms;
cl::Platform::get(&platforms);

//получение вектор устройств для каждой платформы
std::vector<cl::Device> devices;
for (auto &plat_id : platforms)
    getDevices(CL_DEVICE_TYPE_ALL, &devices);

//создание контекста
cl::Context context(CL_DEVICE_TYPE_DEFAULT);

//загрузка кода ядра, создание и компиляция программного объекта
cl::Program program(context, ProgramSource, true);
```



C++ и OpenCL

```
//создание очереди
cl::CommandQueue queue(context);
//создание ядра
auto kernel = cl::make_kernel</*список типов параметров*/>(program,
"kernel");
//загрузка данных в память устройства с хоста
data_device = cl::Buffer(context, data_host.begin(),
data_host.end(), true);
//запуск ядра
kernel(cl::EnqueueArgs(queue, cl::NDRange(count)), /*список
параметров*/);
//загрузка данных в память хоста с устройства
cl::copy(queue, data_device, data_host.begin(), data_host.end());
```



Сумма векторов

- ▶ Пример: сумма двух векторов в стиле C++
- ▶ Функция-ядро вынесена в отдельный c-файл



Иерархия памяти

Ширина шины

Частная память
 $O(2-3)$ слов в такт/элемент

Локальная память
 $O(10)$ слов в такт/группа

Глобальная память
 $O(100-200)$ Гбайт/с

Память хоста
 $O(1-100)$ Гбайт/с

VS

Объем памяти

Частная память
 $O(10)$ слов/элемент

Локальная память
 $O(1-10)$ Кбайт/группу

Global memory
 $O(1-10)$ Гбайт

Host memory
 $O(1-100)$ Гбайт



Частная память

- ▶ Несколько десятков 4-байтных слов на рабочий элемент
- ▶ Там размещаются переменные, объявленные в ядре
- ▶ Каждый рабочий элемент обладает своим набором таких переменных
- ▶ Если частная память кончается, переменные будут размещаться в глобальной памяти



Локальная память

- ▶ Десятки Кбайт на рабочую группу
- ▶ Используется всеми элементами одной рабочей группы
- ▶ Передача данных между локальной и глобальной памятью производится в ядрах
- ▶ Переменные объявленные со спецификатором `__local`



Локальная память

- ▶ Локальная память используется для создания буфера с часто используемыми данными из глобальной памяти
- ▶ Использование локальной памяти в OpenCL это не «серебряная пуля»



Локальная память

- ▶ У CPU нет локальной памяти – приложения, использующие локальную память могут потерять в производительности при исполнении на CPU
- ▶ Размер и скорость кэшей в GPU увеличивается – возможен прирост скорости без прямого участия программиста



Глобальная память

- ▶ Равна размеру оперативной памяти
- ▶ Здесь размещаются динамические массивы и локальные переменные не уместившиеся в частной памяти
- ▶ Переменные объявленные со спецификатором `__global`



Согласованность памяти

- ▶ В OpenCL используется модель памяти с нестрогой согласованностью
- ▶ Состояние памяти видимое для отдельного рабочего элемента не является гарантированно корректным для всех рабочих элементов
- ▶ Рабочий элемент корректно видит свои операции чтения/записи



Согласованность памяти

- ▶ Для элементов рабочей группы согласованность локальной и глобальной гарантирована после барьеров синхронизации
- ▶ Барьерная согласованность не гарантирована для элементов разных рабочих групп



Синхронизация в OpenCL

- ▶ Синхронизация возможна только между рабочими элементами, входящими в одну рабочую группу
- ▶ Синхронизация между рабочими группами выполняющими одно и то же ядро невозможна
- ▶ Два основных вида синхронизации:
 - Барьеры
 - Барьеры памяти



Синхронизация

- ▶ `void barrier()`
 - Рабочий элемент встретивший барьер ждет, пока все элементы группы не подойдут к барьеру
- ▶ Если барьер расположен в ветвлении то
 - Ветвь должны выбрать все элементы группы
 - Ветвь не должен выбрать ни один элемент группы



Синхронизация

- ▶ Барьер памяти – гарантирует, что все операции чтения и/или записи завершены у данного барьера

```
void mem_fence(mem_fence_flag)//все операции чтения и записи
```

```
void read_mem_fence(mem_fence_flag)//все операции чтения
```

```
void write_mem_fence(mem_fence_flag)//все операции записи
```



Произведение матриц

- ▶ Пример: произведение квадратных матриц
- ▶ Двумерная глобальная сетка



Произведение матриц

- ▶ Пример: оптимизация перемножения матриц
- ▶ Одномерная область вычислений размером n
- ▶ Каждый рабочий элемент обсчитывает свою строку матрицы C
- ▶ Использование частной памяти – массив размером n для хранения строки матрицы A
- ▶ Использование локальной памяти – копирование столбца матрицы B в локальную память





Казанский федеральный
УНИВЕРСИТЕТ

ВЫСШАЯ ШКОЛА
информационных технологий
и информационных систем

Задание на практику

Транспонирование матрицы на OpenCL