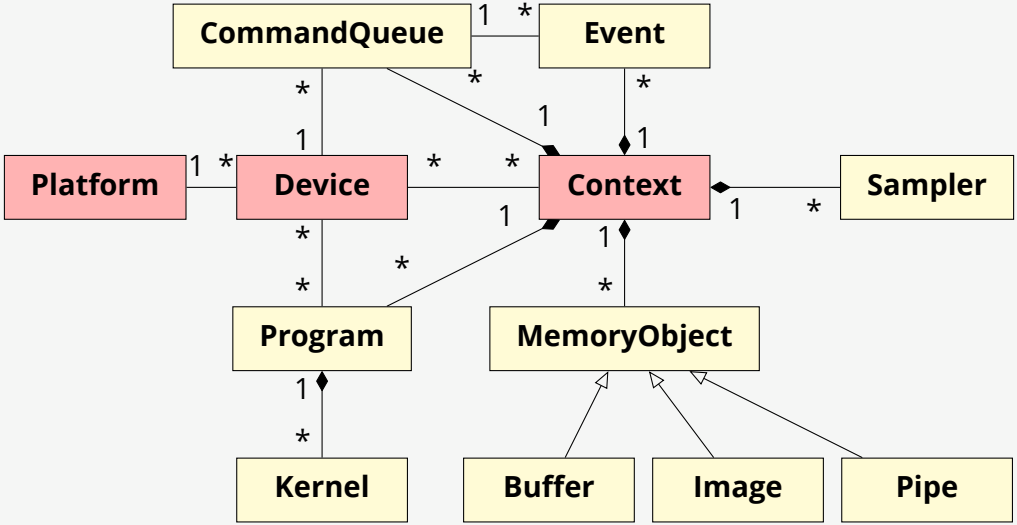
Программирование видеокарт на OpenCL

Архитектура OpenCL



Инициализация OpenCL

Вариант №1 (переменные среды)

Командная строка:

```
export MY_OPENCL_PLATFORM=MESA # название платформы
export MY_OPENCL_DEVICE=Clover # название устройства
./my-opencl-app
```

Файл `my-opencl-app.cc`:

```
if (auto* platform_name = std::getenv("MY_OPENCL_PLATFORM")) {...} else {...}
if (auto* device_name = std::getenv("MY_OPENCL_DEVICE")) {...} else {...}
```

Вариант №2 (конфигурационный файл)

Файл `my-opencl-app.conf`:

```
platform-name = MESA
device-name = Clover
```

Файл `my-opencl-app.cc`:

```
std::ifstream in("my-opencl-app.conf");
if (in.is_open()) {...} else {...}
```

Верификация кода на OpenCL

В программе нужно реализовать алгоритм X.

Порядок действий:

Пример теста:

```
auto eps = 1e-6; // точность
auto n = result_cpu.size(); // размер массива
for (size_t i=0; i<n; ++i) {
  if (std::abs(result_gpu[i] - result_cpu[i]) > eps) {
    std::cerr << "Bad value at i= " << i
              << ", cpu=" << result_cpu[i]
              << ", gpu=" << result_gpu[i] << '\n';
    throw std::runtime_error("verification failed");
  }
}
```

Измерение производительности в C++

```
using namespace std::chrono;
auto t0 = high_resolution_clock::now();
...
auto t1 = high_resolution_clock::now();
auto dt = duration_cast<microseconds>(t1-t0).count();

// календарное время в микросекундах
std::clog << "Elapsed time: " << dt << "us\n";

// пропускная способность =
// кол-во элементов × размер элемента / время
std::clog << "Bandwidth: "
  << (n*sizeof(float)*1e-9)/(dt*1e-6) << "GB/s\n";
// пиковая для Radeon RX 5700: 448 GB/s
```

- `std::clock` измеряет процессорное время.
- Большинство таймеров имеют точность $\pm 10^{-6}$ секунд.

Измерение производительности в OpenCL

```
... // создаем очередь команд с опцией CL_QUEUE_PROFILING_ENABLE
cl::Event a, b, c;
queue.enqueueWriteBuffer(..., &a);    // копируем на видеокарту
queue.enqueueNDRangeKernel(..., &b);  // запускаем ядро
queue.enqueueReadBuffer(..., &c);     // копируем обратно
queue.flush();                        // ждем завершения команд
...
auto t0 = a.getProfilingInfo<CL_PROFILING_COMMAND_START>();
auto t1 = a.getProfilingInfo<CL_PROFILING_COMMAND_END>();
```

Тип события	Описание
CL_PROFILING_COMMAND_QUEUED	команда встала в очередь
CL_PROFILING_COMMAND_SUBMIT	команда отправлена на устройство
CL_PROFILING_COMMAND_START	команда начала выполняться
CL_PROFILING_COMMAND_END	команда закончила выполняться

Запуск ядра

app.cc:

```
const std::string src = "...";
cl::Program prg(context, src);
cl::Kernel kernel(prg, "vector_add");
kernel.setArg(0, ...); // a
kernel.setArg(1, ...); // b
kernel.setArg(2, ...); // c
queue.enqueueNDRangeKernel(kernel,
  cl::NullRange,     // отступ
  cl::NDRange(100),  // общее количество потоков
  cl::NullRange);    // количество потоков в группе
```

src:

```
kernel void
vector_add(global float* a, global float* b, global float* c) {
  int i = get_global_id(0); // индекс потока по первому измерению
  c[i] = a[i] + b[i]; }
```

Индексация потоков

```
queue.enqueueNDRangeKernel(kernel,
  cl::NullRange,
  cl::NDRange(4,4),   // глобальный
  cl::NDRange(2,2))   // локальный
```

Объединение операций чтения/записи

app.cc:

```
queue.enqueueNDRangeKernel(kernel,
  cl::NullRange,     // отступ
  cl::NDRange(100),  // общее количество потоков
  cl::NullRange);    // количество потоков в группе
```

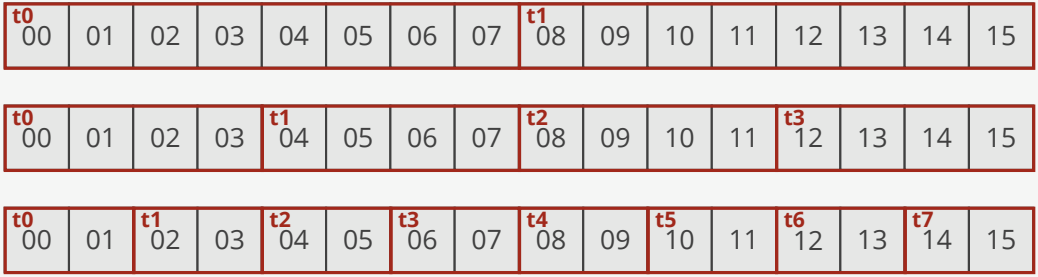
app.cl:

```
kernel void
vector_add_bad(global float* a,
               global float* b,
               global float* c) {
  int i = get_global_id(0);
  c[i*2] = a[i*2] + b[i*2];
}
```

Пропускная способность в два раза ниже по сравнению с `vector_add`.

Расположение в памяти

Ячейки памяти:



- Порядок потоков важен, а количество байт нет.
- Размер шины для Radeon RX 5700: 256 бит (макс. 32 байта объединяется в одну операцию чтения).

Локальная память

Транспонирование матрицы без локальной памяти:

```
kernel void
transpose(global const float* in,    // исходная матрица
          global float* out,         // трансп. матрица
          int n) {                   // размер матрицы
  int i = get_global_id(0);          // строка
  int j = get_global_id(1);          // колонка
  out[j*n + i] = in[i*n + j];
}
```

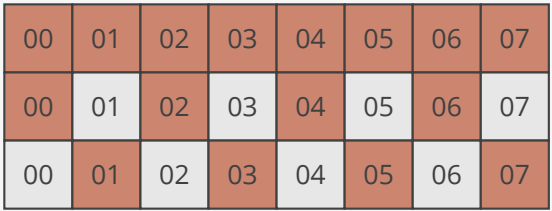
Транспонирование матрицы с локальной памятью:

```
kernel void
transpose(global const float* in,    // исходная матрица
          global float* out,         // трансп. матрица
          int n,                     // размер матрицы
          local float* tile,         // блок матрицы
          int tn) {                  // размер блока
  int i = get_global_id(0);          // строка матрицы
  int j = get_global_id(1);          // колонка матрицы
  int ti = get_local_id(0);          // строка блока
  int tj = get_local_id(1);          // колонка блока
  tile[ti*tn + tj] = in[i*n + j];
  barrier(CLK_LOCAL_MEM_FENCE);      // синхронизация
  out[i*n + j] = tile[tj*tn + ti];   // потоков
}
```

Расхождение потоков внутри фронта

Расхождение:

```
if (i%2 == 0) {
...
} else {...}
```



Нет расхождения:

```
// N=64 для AMD, N=32 для NVIDIA
kernel void no_divergence() {
  int i = get_global_id(0);
  if (i/N == 0) { /* первыйфронт  */ }
  else if (i/N == 1) { /* второйфронт  */ }
}
```

Расхождение:

```
float bisection(float x0, float x1, float param,
                float eps) {
  do {
    ...
  } while (abs(b-a) > eps || ...);      // расхождение
}

kernel void
divergence(float x0, float x1,          // промежуток
           global const float* params,  // параметры функции
           global float* result) {      // выходной массив
  int i = get_global_id(0);
  result[i] = bisection(x0, x1, params[i], eps);
}
```

Фрагментация ресурсов

Одна группа потоков:

- X_1 байт локальной памяти
- Y_1 байт приватной памяти
- Z_1 потоков

Один мультипроцессор:

- X_2 байт локальной памяти
- Y_2 байт приватной памяти
- Z_2 потоков

Вычисление оптимального размера группы:

```
... = device.getInfo<CL_DEVICE_*>(); // параметры видеокарты
... = kernel.getWorkGroupInfo<CL_KERNEL_*>(); // параметры ядра
cl::NDRange local_size = ...;
queue.enqueueNDRangeKernel(kernel,
  cl::NullRange,        // отступ
  cl::NDRange(128),     // общее количество потоков
  local_size);          // количество потоков в группе
```

Параметр	Описание
CL_DEVICE_LOCAL_MEM_SIZE	объем локальной памяти мультипроцессора
CL_DEVICE_MAX_WORK_GROUP_SIZE	макс. количество потоков в группе
CL_DEVICE_MAX_WORK_ITEM_SIZES	макс. количество потоков в группе по измерениям
CL_DEVICE_IMAGE*	макс. размеры изображений
CL_KERNEL_WORK_GROUP_SIZE	макс. размеры группы потоков
CL_KERNEL_LOCAL_MEM_SIZE	объем локальной памяти
CL_KERNEL_PRIVATE_MEM_SIZE	объем приватной памяти

Порядок действий при написании программы

1. Код для процессора.
2. Код для видеокарты.
3. Верификация.
4. Измерение производительности.
5. Оптимизация
   - Объединение операций чтения/записи.
   - Локальная память.
   - Расхождение потоков.
   - Фрагментация.