# opencl中的数组操作

在OpenCL中，使用Buffer对数组进行包装，然后将其设置为核函数参数，执行完后，使用enqueueMapBuffer函数将其映射到主机上，或使用enqueueReadBuffer将其读到主机内存。

OpenCL编程的一般步骤：

1. 获取平台

|  |
| --- |
| vector<Platform> platforms;  Platform::get(&platforms); |

1. 指定一平台，获取设备

|  |
| --- |
| auto platform = platforms[2];  vector<Device> devices;  platform.getDevices(CL\_DEVICE\_TYPE\_GPU, &devices); |

1. 创建上下文

上下文和设置的关系是一对多的关系，在创建上下文时，构造函数中传入设置的向量。

|  |
| --- |
| Context context(devices); |

1. 创建程序并编译

从源码文本中创建程序，先将源码文本读入到字符串中，并创建Sources对象，Sources对象实际上是一个vecotr<pair<string, int>>类型，int为字符串的长度 + 1，以下第4行代码即为创建了包含一个元素的pair<string, int>类型的vector，在program构造函数中传入上下文和sources。

|  |
| --- |
| ifstream helloWorldFile("proc.cl");  string src(istreambuf\_iterator<char>(helloWorldFile), (istreambuf\_iterator<char>()));  Program::Sources sources(1, make\_pair(src.c\_str(), src.length() + 1));  Program program(context, sources); |

1. 创建核

准备数据并设置参数，主机的数组数据需要使用Buffer将其包装，才能作为核的参数。

|  |
| --- |
| Buffer aBuffer = Buffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)A);  Buffer bBuffer = Buffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)B);  Buffer cBuffer = Buffer(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_USE\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)C);  Kernel kernel(program, "vadd");  kernel.setArg(0, aBuffer);  kernel.setArg(1, bBuffer);  kernel.setArg(2, cBuffer); |

1. 创建命令序列并执行

基于上下文和设备创建命令序列，使用enqueueNDRangeKernel进行数据划分并执行核函数，这里global\_size设置为NDRange(BUFFER\_SIZE)，local\_size设置为NullRange

|  |
| --- |
| CommandQueue queue(context, device);  queue.enqueueNDRangeKernel(kernel, NullRange, NDRange(BUFFER\_SIZE), NullRange); |

1. 读取执行结果

使用内enqueueReadBuffer将数据从设备读到主机，或使用内存映射enqueueMapBuffer将设备内存对象映射到主机，在使用完后要解除映射enqueueUnmapMemObject。

|  |
| --- |
| //读取结果  int output1[BUFFER\_SIZE];  queue.enqueueReadBuffer(cBuffer, CL\_TRUE, 0, BUFFER\_SIZE \* sizeof(float), output1);  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << output1[i] << " ";  }  cout << endl;  int\* output2 = (int\*)queue.enqueueMapBuffer(cBuffer, CL\_TRUE, CL\_MAP\_READ, 0, BUFFER\_SIZE \* sizeof(int));  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << output2[i] << " ";  }  cout << endl;  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << C[i] << " ";  }  cout << endl;  queue.enqueueUnmapMemObject(cBuffer, (void\*)output1); |

完整代码：

|  |
| --- |
| #include "stdafx.h"  #include <CL\cl.hpp>  #include <iostream>  #include <fstream>  #include <vector>  using namespace cl;  using namespace std;  #define BUFFER\_SIZE 20  int A[BUFFER\_SIZE];  int B[BUFFER\_SIZE];  int C[BUFFER\_SIZE];  int main()  {  for (int i = 0; i < BUFFER\_SIZE; i++)  {  A[i] = i;  B[i] = i \* 2;  C[i] = 0;  }  vector<Platform> platforms;  Platform::get(&platforms);  auto platform = platforms[2];  vector<Device> devices;  platform.getDevices(CL\_DEVICE\_TYPE\_GPU, &devices);  Device device = devices.front();  //1.2 创建context  Context context(devices);  //2.1 创建program  ifstream helloWorldFile("proc.cl");  string src(istreambuf\_iterator<char>(helloWorldFile), (istreambuf\_iterator<char>()));  Program::Sources sources(1, make\_pair(src.c\_str(), src.length() + 1));  Program program(context, sources);  //2.2 编译program  program.build("-cl-std=CL1.2");  //准备数据  Buffer aBuffer = Buffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)A);  Buffer bBuffer = Buffer(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_COPY\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)B);  Buffer cBuffer = Buffer(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_USE\_HOST\_PTR, BUFFER\_SIZE \* sizeof(int), (void\*)C);  //设置参数  Kernel kernel(program, "vadd");  kernel.setArg(0, aBuffer);  kernel.setArg(1, bBuffer);  kernel.setArg(2, cBuffer);  //执行核  CommandQueue queue(context, device);  queue.enqueueNDRangeKernel(kernel, NullRange, NDRange(BUFFER\_SIZE), NullRange);  //读取结果  int output1[BUFFER\_SIZE];  queue.enqueueReadBuffer(cBuffer, CL\_TRUE, 0, BUFFER\_SIZE \* sizeof(float), output1);  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << output1[i] << " ";  }  cout << endl;  int\* output2 = (int\*)queue.enqueueMapBuffer(cBuffer, CL\_TRUE, CL\_MAP\_READ, 0, BUFFER\_SIZE \* sizeof(int));  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << output2[i] << " ";  }  cout << endl;  for (int i = 0;i < BUFFER\_SIZE;i++)  {  cout << C[i] << " ";  }  cout << endl;  queue.enqueueUnmapMemObject(cBuffer, (void\*)output1);  cin.get();  return 0;  } |

