# OpenCL程序流程

## 1.基本流程

编程基本流程：

1. Setup

获取platform和devices

创建context

创建queues

1. Compilation

创建program

编译program

创建kernels

1. 创建内存对象
2. 将数据写到GPU
3. 设置kernel参数
4. 执行kernel
5. 将数据读到host

## 例程1

编写一个HelloWorld.cl文件，内容如下：

\_\_kernel void HelloWorld(\_\_global char\* data)

{

data[0] = 'H';

data[1] = 'e';

data[2] = 'l';

data[3] = 'l';

data[4] = 'o';

data[5] = ' ';

data[6] = 'W';

data[7] = 'o';

data[8] = 'r';

data[9] = 'l';

data[10] = 'd';

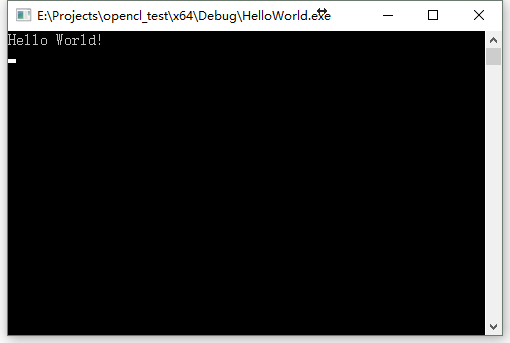
data[11] = '!';

data[12] = '\n';

}

编写一个主程序，内容如下：

|  |
| --- |
| #include <CL\cl.hpp>  #include <iostream>  #include <fstream>  #include <string>  using namespace std;  using namespace cl;  int main()  {  //1.Setup  //1.1 获取platform和devices  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);  //1.3 创建queues  CommandQueue queue(context, device);  //2.Compilation  //2.1 创建program  ifstream helloWorldFile("HelloWorld.cl");  std::string src(istreambuf\_iterator<char>(helloWorldFile), (istreambuf\_iterator<char>()));  Program::Sources sources(1, std::make\_pair(src.c\_str(), src.length() + 1));  Program program(context, sources);  //2.2 编译program  auto err = program.build("-cl-std=CL1.2");  //2.3 创建kernels  Kernel kernel(program, "HelloWorld", &err);  //3.创建内存对象  char buf[20];  Buffer memBuf(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_HOST\_READ\_ONLY, sizeof(buf));  //4.将数据写到GPU  //5.设置Kernel参数  kernel.setArg(0, memBuf);  //6.执行Kernel  queue.enqueueTask(kernel);  //7.将数据读到host  queue.enqueueReadBuffer(memBuf, CL\_TRUE, 0, sizeof(buf), buf);  cout << buf;  cin.get();  return 0;  } |



## 例程2

### 3.1 OpenCLHelper.h文件

|  |
| --- |
| #pragma once  #include <CL\cl.hpp>  #include <string>  cl::Program CreateProgram(std::string filename); |

### 3.2 OpenCLHelper.cpp文件

|  |
| --- |
| #include <CL\cl.hpp>  #include <fstream>  #include <iostream>  #include <string>  #include "OpenCLHelper.h"  using namespace std;  using namespace cl;  cl::Program CreateProgram(std::string filename)  {  //1.Setup  //1.1 获取platform和devices  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.Compilation  //2.1 创建program  ifstream helloWorldFile(filename.c\_str());  std::string src(istreambuf\_iterator<char>(helloWorldFile), (istreambuf\_iterator<char>()));  Program::Sources sources(1, std::make\_pair(src.c\_str(), src.length() + 1));  Program program(context, sources);  //2.2 编译program  auto err = program.build("-cl-std=CL1.2");  return program;  } |

### 3.3 ProcessArray.cl文件

|  |
| --- |
| \_\_kernel void ProcessArray(\_\_global int\* data, \_\_global int\* outData)  {  outData[get\_global\_id(0)] = data[get\_global\_id(0)] \* 2;  } |

### 3.4 main.cpp文件

|  |
| --- |
| #include <CL\cl.hpp>  #include <iostream>  #include <fstream>  #include <string>  #include <OpenCLHelper.h>  using namespace std;  using namespace cl;  void main()  {  auto program = CreateProgram("ProcessArray.cl");  auto context = program.getInfo<CL\_PROGRAM\_CONTEXT>();  auto devices = context.getInfo<CL\_CONTEXT\_DEVICES>();  auto& device = devices.front();    vector<int> vec(1024);  fill(vec.begin(), vec.end(), 1);  Buffer inBuf(context, CL\_MEM\_READ\_ONLY | CL\_MEM\_HOST\_NO\_ACCESS | CL\_MEM\_COPY\_HOST\_PTR, sizeof(int) \* vec.size(), vec.data());  Buffer outBuf(context, CL\_MEM\_WRITE\_ONLY | CL\_MEM\_HOST\_READ\_ONLY, sizeof(int)\*vec.size());  Kernel kernel(program, "ProcessArray");  kernel.setArg(0, inBuf);  kernel.setArg(1, outBuf);    CommandQueue queue(context, device);  queue.enqueueNDRangeKernel(kernel, NullRange, NDRange(vec.size()));  queue.enqueueReadBuffer(outBuf, CL\_FALSE, 0, sizeof(int) \* vec.size(), vec.data());  finish();  cin.get();  } |