**英特尔oneAPI创新大使征文**

英特尔的oneAPI是英特尔提供的统一编程模型和软件开发框架。 其目的在于简化可充分利用英特尔各种硬件架构，包括 CPU、GPU 和 FPGA的应用程序的开发。

英特尔oneAPI 提供了一组工具、库和框架，使开发人员能够编写跨不同硬件平台的高性能代码。 它支持多种编程语言，包括 C++、Fortran 和数据并行 C++。 借助 英特尔oneAPI，开发人员可以使用熟悉的编程模型编写代码并针对不同的硬件架构，而无需对代码进行重大修改。

英特尔 oneAPI 的关键组件包括： 1、oneAPI Base Toolkit：它包括编译器、库和工具，用于跨 CPU、GPU 和 FPGA 优化和并行化代码。2、oneAPI HPC Toolkit：它专注于高性能计算 (HPC) 工作负载，并为 HPC 开发提供额外的工具和库。3、oneAPI AI Analytics Toolkit：它专为人工智能 (AI) 和分析工作负载而设计，并为深度学习、机器学习和数据分析提供库和工具。4、oneAPI IoT Toolkit：专为物联网（IoT）应用量身定制，提供用于开发物联网解决方案的工具和库。

通过使用 oneAPI 编程模型和工具，开发人员可以编写可在不同类型的英特尔硬件上高效执行的代码，释放提高性能和能效的潜力。 它促进了一种统一且可扩展的软件开发方法，使开发人员能够利用英特尔硬件产品组合的全部功能。

以下使用oneAPI实现一个矩阵乘法。

#include <CL/sycl.hpp>

constexpr int N = 16;

using namespace sycl;

class IntelGPUSelector : public device\_selector {

public:

int operator()(const device& Device) const override {

const std::string DeviceName = Device.get\_info<info::device::name>();

const std::string DeviceVendor = Device.get\_info<info::device::vendor>();

return Device.is\_gpu() && (DeviceName.find("Intel") != std::string::npos) ? 100 : 0;

}

};

int main() {

IntelGPUSelector d;

queue q(d);

sycl::range<2> size(N, N);

std::vector<vector<float>> v1(N,vector<int>(N,2.0f));

std::vector<vector<float>> v2(N,vector<int>(N,3.0f));

std::vector<vector<float>> v3(N,vector<int>(N,0.0f));

buffer<float, 2> buf1(v1.data(),size);

buffer<float, 2> buf2(v2.data(),size);

buffer<float, 2> buf3(v3.data(),size);

q.submit([&](handler& h) {

auto acc1 = buf1.get\_access<sycl::access::mode::read>(h);

auto acc2 = buf2.get\_access<sycl::access::mode::read>(h);

auto acc3 = buf3.get\_access<sycl::access::mode::write>(h);

h.parallel\_for<class MatrixMultiply>(size, [=](sycl::id<2> idx){

float sum = 0.0f;

for (int i = 0; i < N; i++) {

sum += acc1[idx[0]][i] \* acc2[i][idx[1]];

}

acc3[idx] = sum;

});

});

q.wait();

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

std::cout << v3[i \* N + j] << " ";

}

std::cout << std::endl;

}

return 0;

}

在导入相关库后，我们创建三个矩阵v1，v2，v3，创建用于并行计算的SYCL队列，并创建SYCL缓冲区。定义内核后，我们在内核中使用访问器，读取输入矩阵v1与v2，结果写入v3.