# 异构计算试验报告

---实验3：基于opencl的矩阵乘法

第一部分：实验环境

OS：Windows 10

CPU：intel(R) Core(TM) i7-10510U CPU@ 1.80Ghz

GPU: NVIDIA GeForce MX250

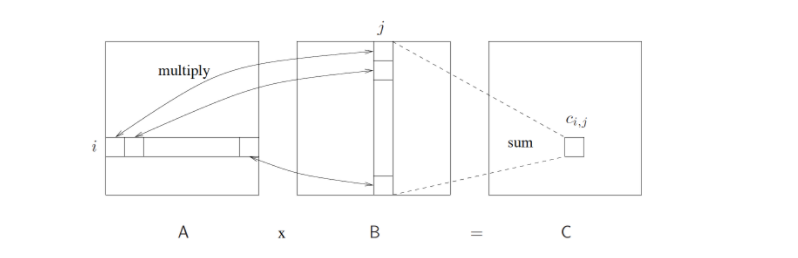
opencl版本：2019

第二部分：实验内容

在OpenCL下使用GPU完成矩阵乘法

第三部分：实验原理

使用多重循环，逐步计算矩阵的值



定义宏

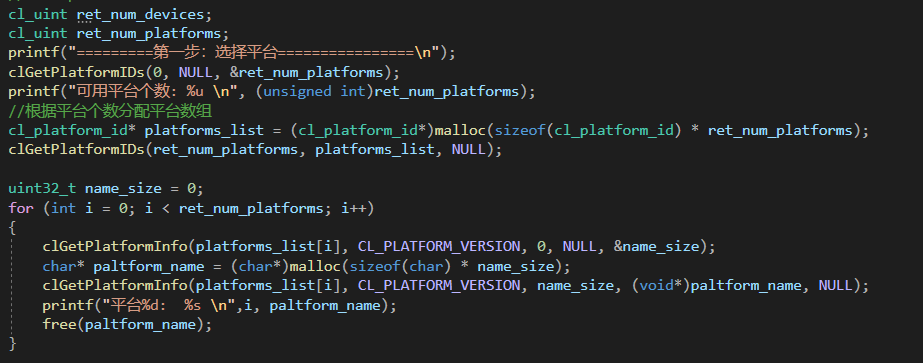


第一个宏 用于使用 下标 作为索引，寻址矩阵中的某个值

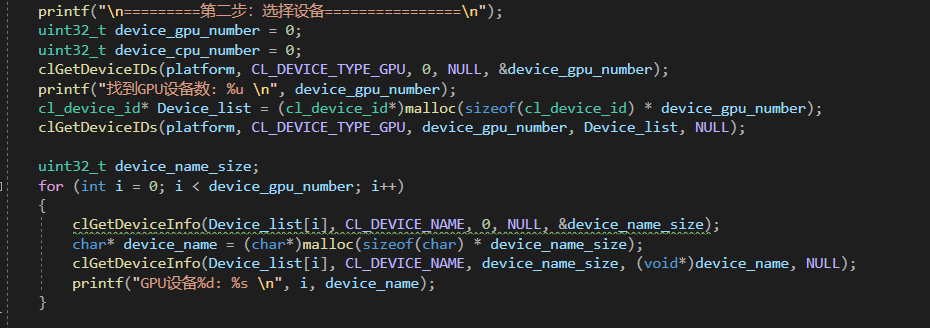
第二个宏 用虚拟地址作为索引，寻址矩阵中某个元素的地址

运行opnecl的步骤为

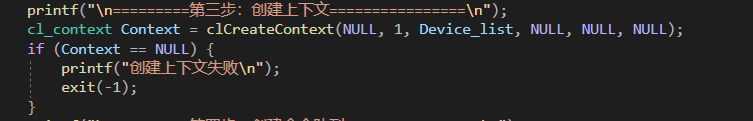
1. 查询平台



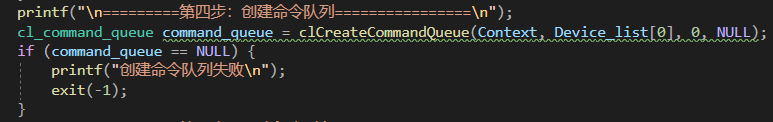
1. 查询设备



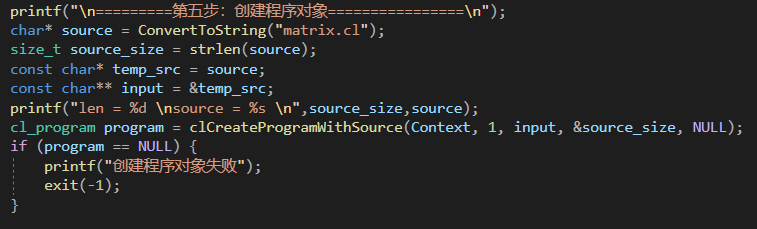
1. 创建上下文



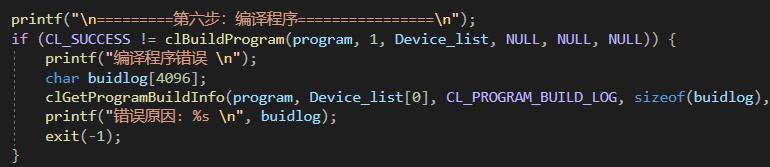
1. 创建命令队列



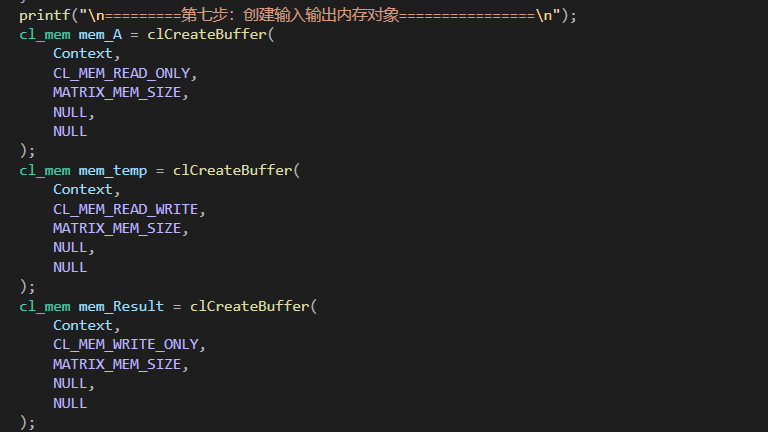
1. 创建程序对象



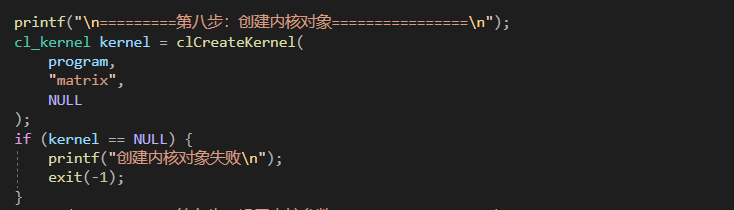
1. 编译程序



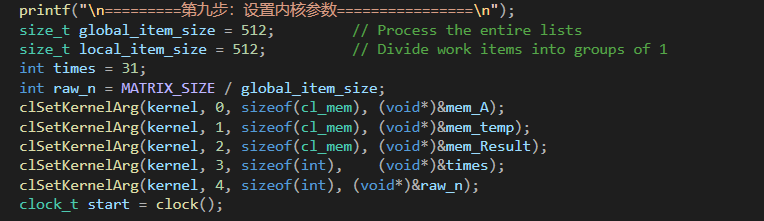
1. 创建输入输出对象



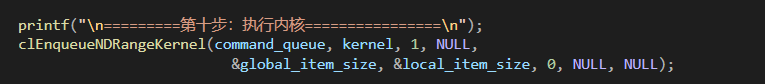
1. 创建内核对象



1. 设置内核参数

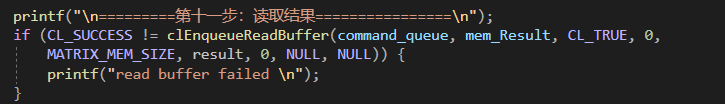


1. 执行内核

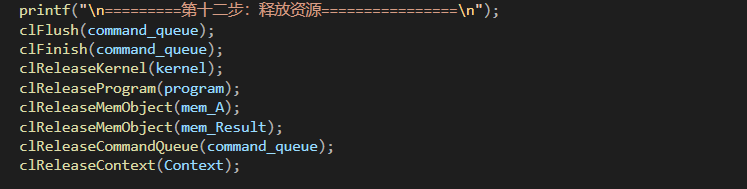


1. 读取结果

程序在执行内核阶段并不会阻塞，而是阻塞在读取结果的阶段



1. 释放资源



第四部分：性能分析

计算规模：512 \* 512 双精度浮点矩阵，迭代自乘 32 次

为了同步内存，将所有的计算放在一个work\_group中，使用barrier进行内存同步

计算配置：work\_group = 1;

work\_items = 512;

每一个线程负责对矩阵中的一行进行计算

测试1：



测试2：



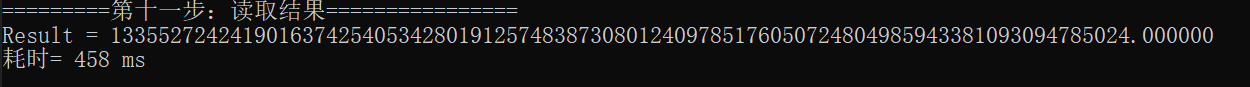
测试3：



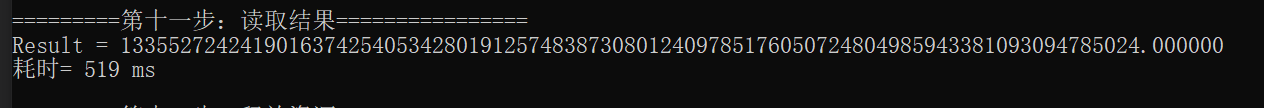
平均用时 = 3506ms

快速算法

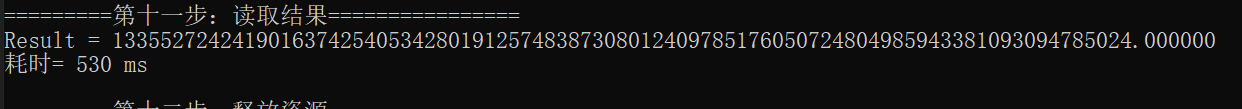
测试1



测试2



测试3

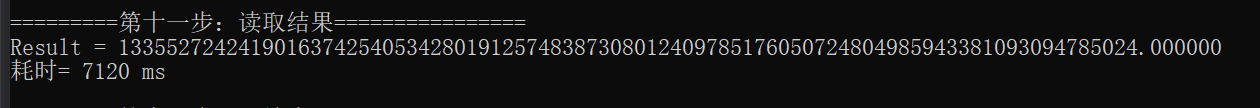


平均用时 = 502 ms

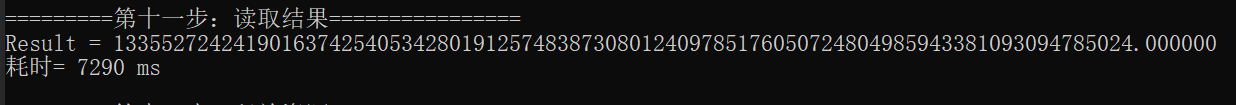
CPU部分，基于上面的代码框架，很容易使用CPU实现

暴力算法

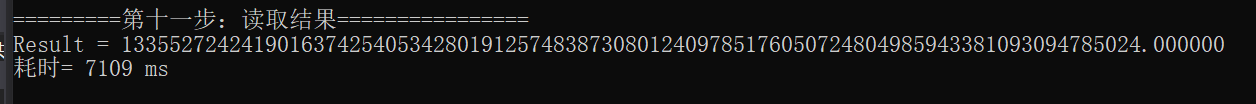
测试1



测试2



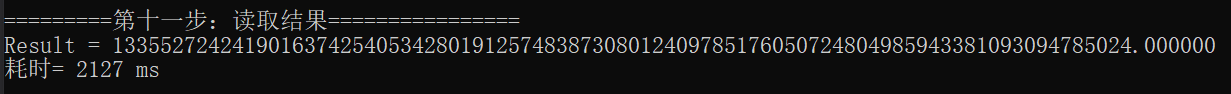
测试3



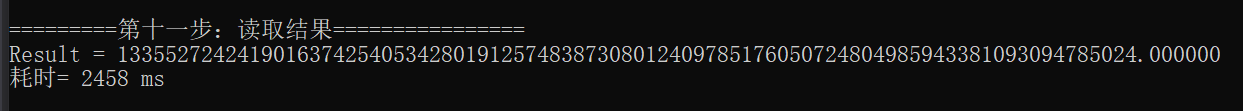
平均用时: 7173ms

高效算法

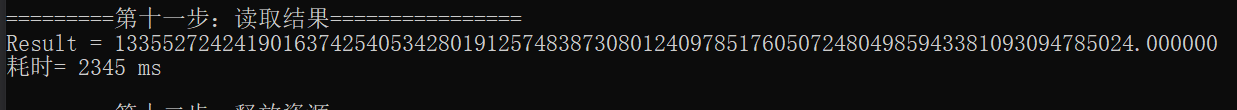
测试1



测试2



测试3



平均时间：2310ms

纵向对比：至此，关于矩阵乘法一共使用过的计算方法为

1. CPU单核 暴力计算
2. CPU并行 暴力计算
3. GPU并行 暴力计算
4. CPU 单核 快速算法
5. CPU 并行 快速算法
6. GPU 并行 快速算法
7. OpenCL + GPU并行 暴力算法
8. OpenCL + GPU并行 快速算法
9. OpenCL + CPU并行 暴力算法
10. OpenCL + CPU并行 快速算法

综合为表格为（单位ms）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CPU单核 | CPU并行 | GPU并行 | OpenCL+CPU并行 | OpenCL +GPU 并行 |
| 暴力算法 | 24205 | 6820 | 2220 | 7173 | 3506 |
| 快速算法 | 3714 | 1773 | 360 | 2310 | 502 |

数据分析

加速比分析，以CPU单核暴力计算为1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CPU单核 | CPU并行 | GPU并行 | OpenCL+CPU并行 | OpenCL+GPU  并行 |
| 暴力算法 | 1 | 3.55 | 10.90 | 3.37 | 6.90 |
| 快速算法 | 6.52 | 13.65 | 67.24 | 10.48 | 48.21 |

数据分析

第五部分：总结

相较于CUDA,OpenCL编程模型会更加复杂，编程的难度更高。OpenCL模型与CUDA有很多相似之处，包括对内存和线程的管理。OpenCL最大的优势在在于OpenCL专注与在不同的 异构平台 进行高性能计算，而CUDA只可以在Nvidia 的显卡进行计算。在实际的计算过程中，还是面向专一计算平台的 CUDA 更快一些