南开大学

Nankai University

南　　开　　大　　学

计 算 机 学 院

并行程序设计实验报告

GPU 编程

蒋薇

年级：2021 级

专业：计算机科学与技术

指导教师：王刚

2023 年 6 月 27 日

摘要

关键字：Parallel，GPU，

目录

一、 矩阵乘法

(一) 问题三、四

3.Test the matrix computation performance for different tile_X and tile_Y sizes (tile_X and tile_Y may be of different size), and analyze the results and figure out the reasons.

测试不同的 tile_X 和 tile_Y 大小的矩阵计算性能（tile_X 和 tile_Y 可以是不同的大小），分析结果并找出原因。

矩阵乘法代码 请点击这里

4.Increase the size of the input register so as to read in more rows and columns at a time, and test and analyze the performance.

增加输入 register 的大小，一次读取更多的行和列，然后测试和分析性能。

矩阵乘法代码 请点击这里

(二) 题目分析

三维数组，第一维（K）表示批量大小，这意味着输入包含 K 个独立的样例。第二维（M）表示输入的类别或分类，这在多类别分类问题中可能会使用。第三维（N）表示特征维度，即用于表示每个样例的特征或属性数量。

例如，在图像识别任务中，K 可以表示批量大小，M 可以表示不同的物体类别，N 可以表示图像的像素特征。K 维度相互独立，二维交叉熵计算会在每个平面上进行。在每个批量大小 K 中，都会计算出一个大小为 N 的 loss 数组，表示每个样例在每个特征维度上的损失值。

应用程序的开发者只需要开发一次代码，就可以让代码在跨平台的异构系统上执行，底层的硬件架构可以是 CPU、GPU、FPGA、神经网络处理器等。由此可见，使用 oneAPI 编写的程序既可以利用加速器提高程序性能，又具有可移植性。

一个 oneAPI 运行环境由一个主机和一系列设备组成。主机通常是一个多核 CPU，而设备是一个或多个 GPU、FPGA，或是其他加速器。主机的处理器也可以进行并行计算。

(三) 测试及结果

用 Intel oneAPI 基础工具套件中的 C++/SYCL 直接编程语言实现异构编程改变 tileX 、tileY

$GEMMsize, M = 512, N = 512, K = 512$

$workgroupsize = 4$

$repeattime = 10$

```
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2332368.v-qsvr-1           ...ub-singleuser u196201         00:03:11 R jupyterhub
2332584.v-qsvr-1           your.sh          u196201                0 Q batch

Waiting for Output ████████████████████████ Done!

########################################################################
#      Date:           Tue 27 Jun 2023 05:27:41 AM PDT
#    Job ID:           2332584.v-qsvr-1.aidevcloud
#      User:           u196201
# Resources:           cput=75:00:00,neednodes=1:gen11:ppn=2,nodes=1:gen11:ppn=2,walltime=06:00:(
########################################################################

## u196201 is compiling oneMKL_introduction Module0 -- gemm with usm - 7 of 7 your.cpp
Problem size: c(512,512) = a(512,512) * b(512,512)

GEMM size M = 512, N = 512, K = 512
Work-Group size = 4 * 4, tile_X = 2, tile_Y = 2
Performance Flops = 268435456.000000,
GPU Computation Time = 6.586435 (ms);
CPU Computaiton Time = 18.843197 (ms);

########################################################################
# End of output for job 2332584.v-qsvr-1.aidevcloud
# Date: Tue 27 Jun 2023 05:27:54 AM PDT
########################################################################
```

图 1: Caption

```
Done!

########################################################################
#      Date:           Tue 27 Jun 2023 07:01:49 AM PDT
#    Job ID:           2332644.v-qsvr-1.aidevcloud
#      User:           u196201
# Resources:           cput=75:00:00,neednodes=1:gen11:ppn=2,nodes=1:gen11:ppn=2,walltime=06:00:00
########################################################################

## u196201 is compiling oneMKL_introduction Module0 -- gemm with usm - 7 of 7 your.cpp
Problem size: c(512,512) = a(512,512) * b(512,512)

GEMM size M = 512, N = 512, K = 512
Work-Group size = 4 * 4, tile_X = 16, tile_Y = 16
Performance Flops = 268435456.000000,
GPU Computation Time = 14.882038 (ms);
CPU Computaiton Time = 18.838009 (ms);

########################################################################
# End of output for job 2332644.v-qsvr-1.aidevcloud
# Date: Tue 27 Jun 2023 07:02:03 AM PDT
########################################################################
```

图 2: Caption

Performance Flops = 268435456.000000
表

tileX	tileY	Performance Flops	GPU computation time(ms)	CPU Computation time(ms)
2	2	268435456	6.586435	18.843197
4	4	268435456.000000	3.336934	18.893013
8	8	268435456.000000	1.851502	26.686701
16	16	268435456.000000	14.882038	18.838009

表 1: 改变 tileX、tileY 实验结果 (单位:ms)

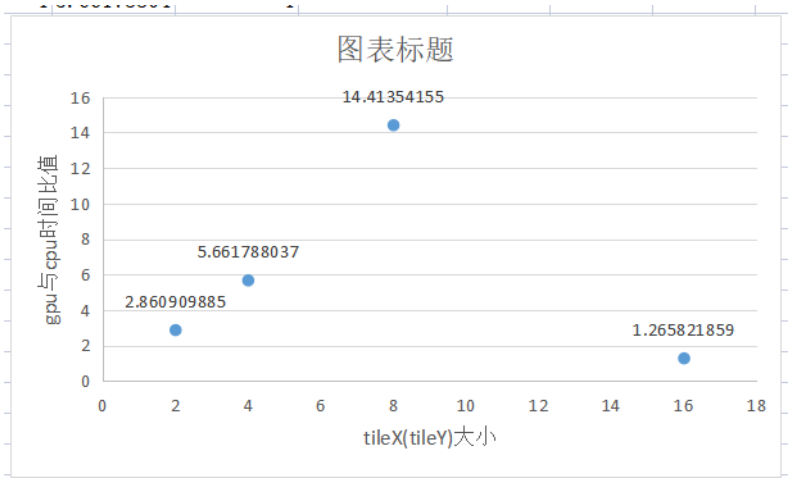

图 3: Caption

比较基本 SYCL 实现和数学核库实现的执行时间，我们可以发现对于 1024x1024 的小矩阵大小，基本 SYCL 实现比 MKL 实现表现更好。当矩阵大小变大时，MKL 实现显著优于基本 SYCL 实现。

下面的图表显示了矩阵大小为 1024x1024、5120x5120 和 10240x10240 的各种硬件上的执行时间。

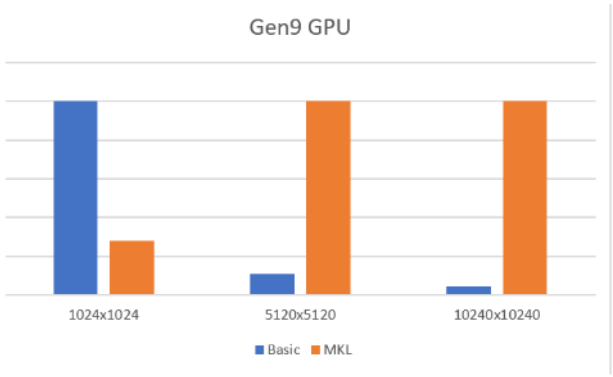

图 4: Caption

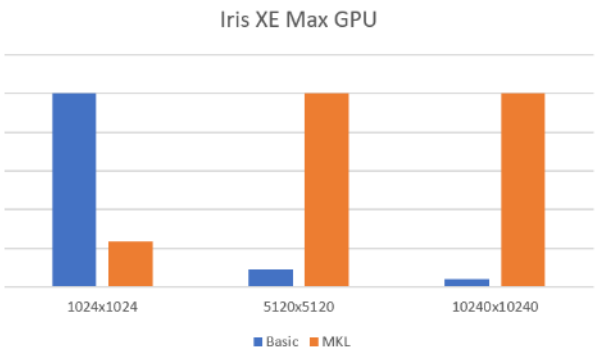

图 5: Caption

随着数组大小的增加，性能有所提高。

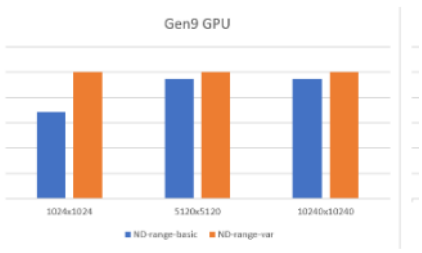

图 6: Caption

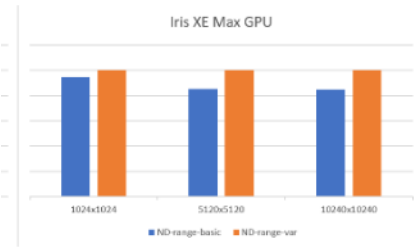

图 7: Caption

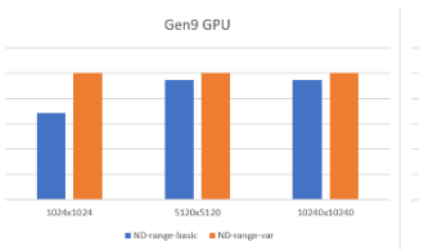

图 8: Caption

（四） 实验分析

基于 Intel oneAPI 的应用性能分析工具分析、优化方法介绍

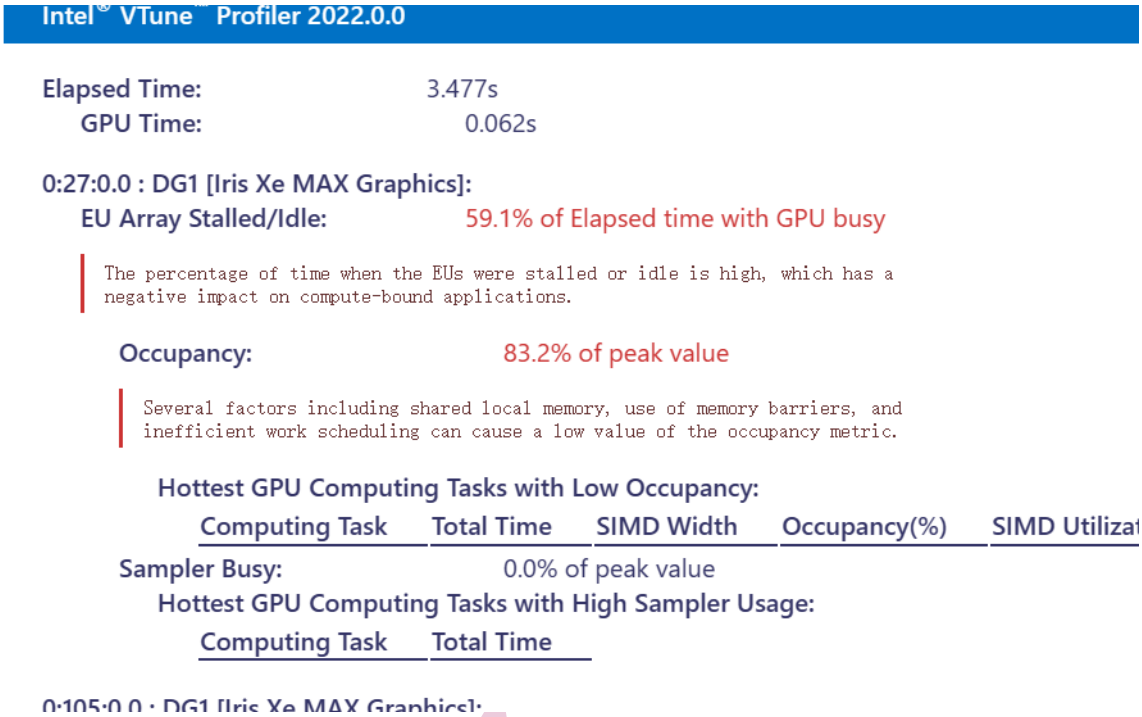

图 9: Caption

我们探讨了如何查询平台以获取更详细的信息，以便更好地选择工作组大小。此外，我们还讨论了为什么了解使用的算法及其如何影响选择工作组大小非常重要。最后，我们展示了参数化对性能的影响，并产生了更多的加速。

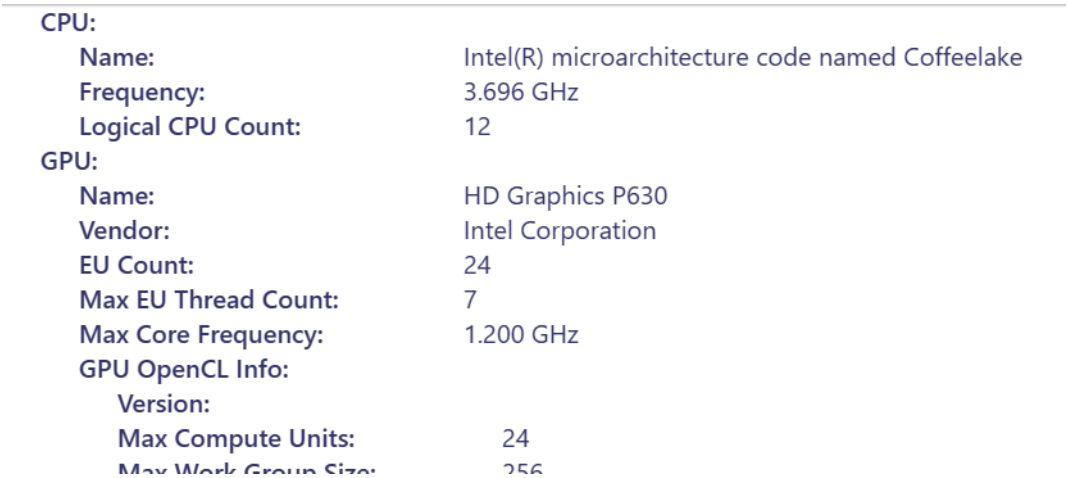

图 10: Caption

需要注意的是，通过编写自己的参数化方案，我们需要创建比使用 oneMKL 库所需的更多代码行。

使用 oneAPI 和 SYCL 编写异构代码时，提供了一种方法论，可供选择使用库，或者如果库不可用，使用 SYCL 和 oneAPI 编写代码。

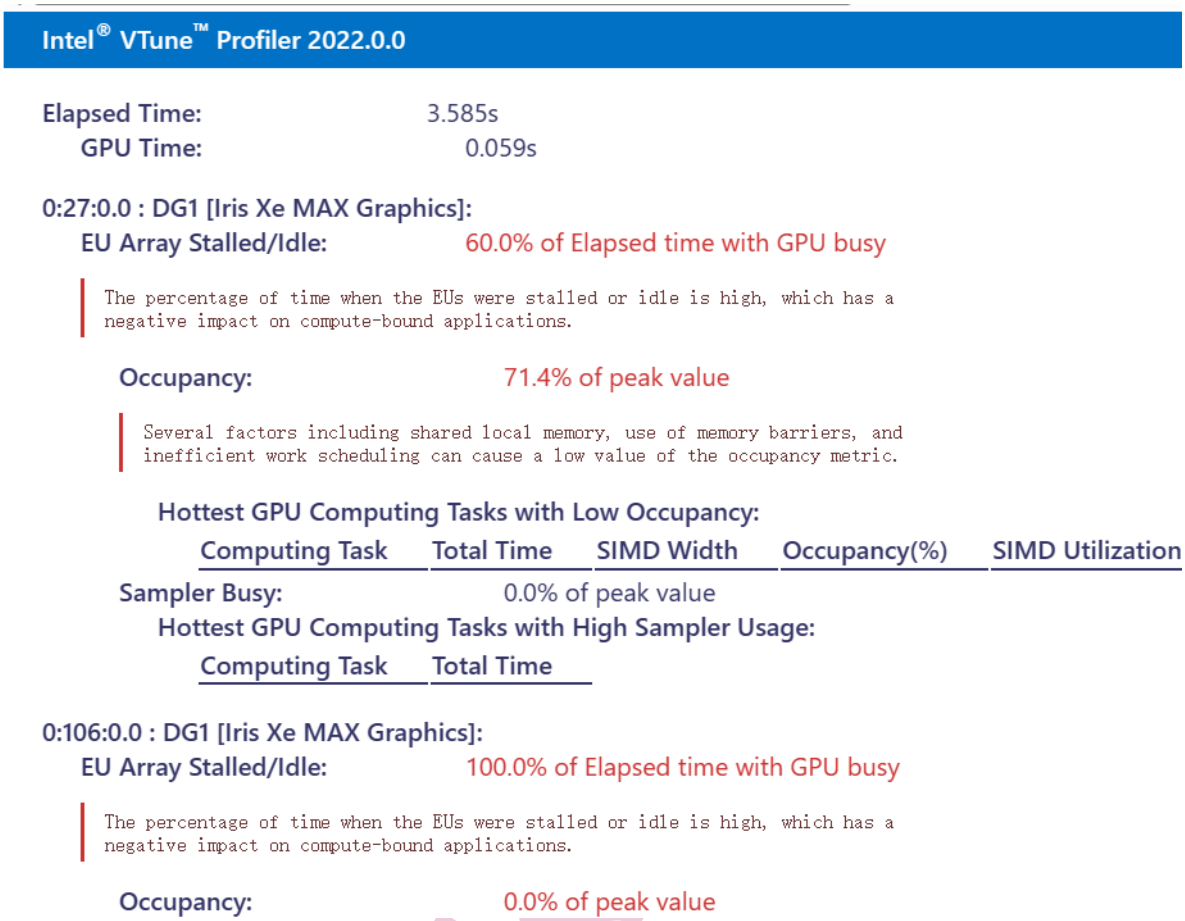

图 11: Caption

几个因素可能导致占用率指标值低，其中包括共享本地内存、使用内存屏障和低效的工作调度。执行单元停滞或空闲的时间百分比很高，这对计算密集型应用程序产生了负面影响。矩阵的大小和工作组对内核在各平台上的性能都产生影响。我们看到硬件上的改进，但我们可以看到工作调度可能更好地优化，以利用 DG1 和 ATS 上可用的更多 EU 和 CPU。

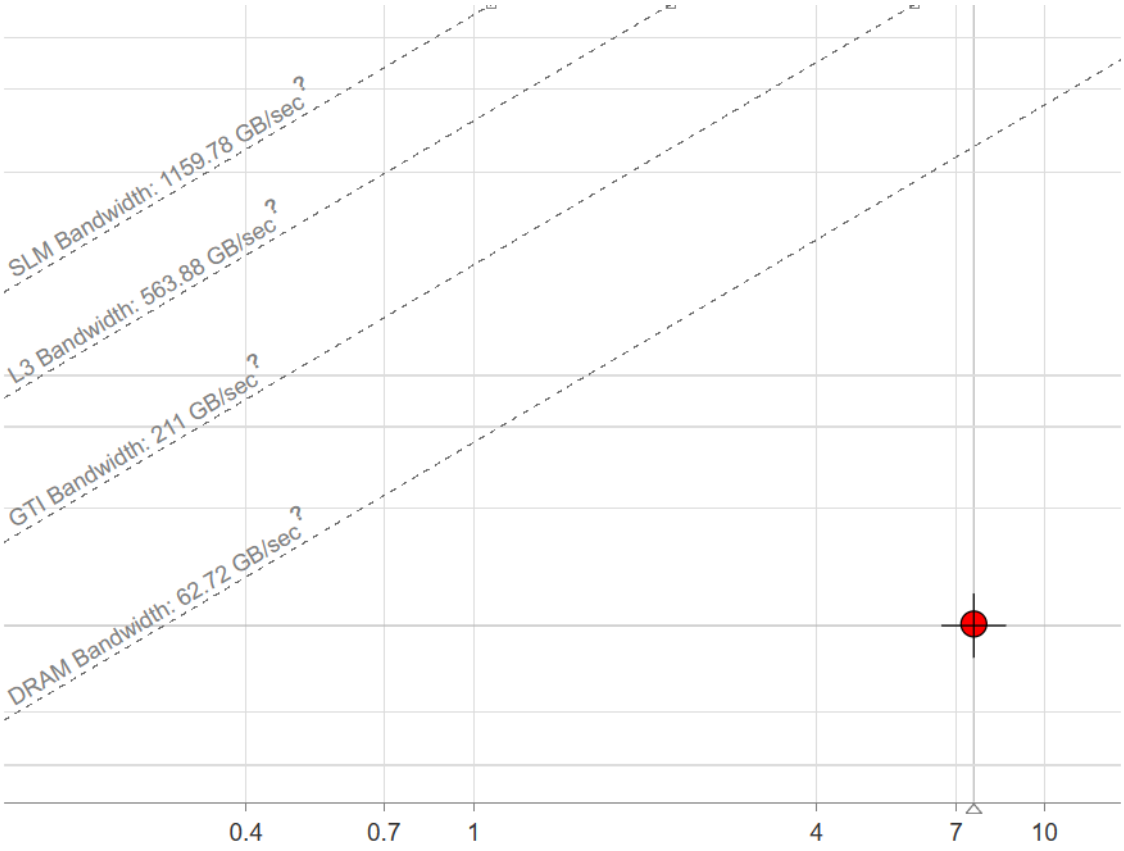

图 12: Caption

运行“选择卸载设备”部分中的单元格，选择要在其上运行代码的目标设备。接下来，运行“构建和运行”部分中的单元格，以工作组优化的方式编译和执行本地内存实现的代码。

通过遵循 oneAPI GPU 优化指南，可以进行许多优化。oneAPI GPU 优化指南, 涵盖了与内核的编码、提交和执行相关的主题。减少, 子组, 避免寄存器溢出, 共享本地内存, 消除条件检查, 内核启动, 在加速器卸载中使用库, 在 SYCL 内核中使用标准库函数, 使用 oneAPI 数学内核库（oneMKL）高效实现傅里叶相关, 同时在设备上执行多个内核, 内核中的线程同步, Restrict 指令, 将内核提交到多个队列, 避免冗余队列构建, 选择工作组大小的考虑.

二、 oneAPI 高斯消去的 GPU 并行化编程

(一) 算法分析

oneAPI 工具集包括多种工具和库，其中包括 DPC++ 编译器、Intel MKL、Intel TBB 等，这些工具和库可以协同工作，实现高效的并行计算。DPC++ 编译器是英特尔 oneAPI 工具集中的一个重要组件，可以将 C++ 代码编译成可在 GPU 上运行的代码，实现高效计算。

并行化分析第一重循环遍历矩阵所有行，当前选中行（第 i 行）便被用来消去其他行；第二重循环被用来遍历第 i 行“下面”的所有矩阵行（第 j 行）；而第三重循环则被用来将第 j 行减去第 i 行，来实现将位于 (j, i) 位置的矩阵元素置零的作用。

我们还可以在创建队列时，将队列绑定到不同设备上，例如：

queue(cpu_selector); queue(gpu_selector); 可以由此指定我们的并行代码要在什么设备上执行，我们不需要了解 GPU/FPGA 等加速器具体的编程语言，就可以将计算任务以统一的编

程方式卸载到这些加速器上

针对第一个部分，我们可以使用 parallel_for 语句将其卸载到加速器上并行计算，具体代码如下：

代码

第一部分

```
    q.submit([&](handler& h) {
        accessor m{ buf, h, read_write };
        h.parallel_for(range(n - k), [=](auto idx) {
                int j = k + idx;
                m[k][j] = m[k][j] / m[k][k];
        });
});
```

针对第二个部分，我们仍然可以使用 parallel_for，不过需要注意的是，这里使用的不再是一维的数据划分，而是二维的数据划分，具体代码如下：

代码

第二部分

```
    q.submit([&](handler& h) {
        accessor m{ buf, h, read_write };
        h.parallel_for(range(n - (k + 1), n - (k + 1)), [=](auto idx) {
                int i = k + 1 + idx.get_id(0);
                int j = k + 1 + idx.get_id(1);
                m[i][j] = m[i][j] - m[i][k] * m[k][j];
        });
});
```

除此之外，还有一点需要注意，由于 A[i,k] 元素在此部分中会被所有线程使用，因此不能在此阶段置 0，需要等这部分全部执行完成之后再置 0。因此需要补充第三个阶段，将右下角 k*k 大小的矩阵的第一列除 A[k,k] 以外的元素置 0，具体代码如下：

代码

第三部分

```
    q.submit([&](handler& h) {
        accessor m{ buf, h, read_write };
        h.parallel_for(range(n - (k + 1)), [=](auto idx) {
                int i = k + 1 + idx;
                m[i][k] = 0;
        });
});
```

在最外层 n 重循环执行完之后，其实只是将计算任务下发到了加速器，这些计算任务不一定真的执行完了，因此我们需要手动调用 q.wait()，等待任务队列中所有任务都执行完毕，该算法再结束。

预热

为了防止初始化操作、cache 等其他因素影响算法执行时间的测量，因此在开始计时之前，先让算法运行一次，第一次不计入运行时间，然后再运行 k 次，以这 k 次运行的平均时间作为算法执行时间。

(二) 结果分析

GPU 型号：Intel(R) UHD Graphics P630

CPU 型号：Intel(R) Xeon(R) E-2176G CPU @ 3.70GHz

表 2

问题规模 n	串行算法	GPU 并行算法
16…	0.470433	1.42439
32	0.812061	2.292
64	6.41777	4.03059
128	50.598	5.60625
256	406.665	7.42548
512	3236.67	75.7885
1024	25885.7	590.462

表 2: 串行算法与基于 buffer 实现的并行算法的性能对比算法执行时间，单位：ms

当问题规模 n 非常小的情况下，串行算法的效率是最高的。因为如果我们使用 GPU 做并行化，那么就需要将矩阵从主机的内存复制到 GPU 的内存中，然后再让 GPU 开始计算，等 GPU 计算完之后，还要把矩阵再从 GPU 的内存复制到主机的内存中。在问题规模 n 非常小的情况下，这些额外的开销占比是非常大的，因此在此时 GPU 版本算法要比串行算法的执行时间更长。

当问题规模 n 比较大时，GPU 并行算法耗时更短，比较符合预期效果。

表 3

问题规模 n	GPU 并行算法（buffer）	GPU 并行算法（USM）
16	1.42439…	1.59059
32	2.292	2.02017
64	4.03059	2.9549
128	5.60625	4.81008
256	7.42548	5.91026
512	75.7885	89.6216
1024	590.462	616.103

表 3: 基于 buffer 实现的并行算法与基于 USM 实现的并行算法的性能对比

性能差异不大，使用 USM 更快地将现有 C/C++ 程序移植到 DPC++，并且相比于使用 buffer+accessor，不会带来很多额外的性能开销。

三、 线上学习

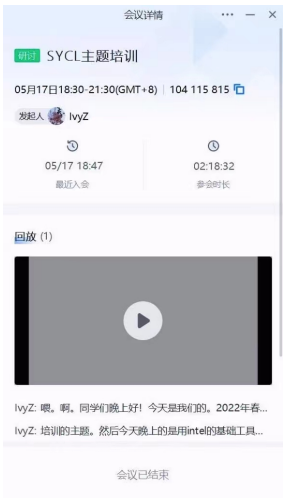

图 13: Caption

图 14: Caption

图 15: Caption

学习过程及内容

SYCL 主题培训，我知道了 SYCL 是一种编程模型，允许开发者为异构系统（如 CPU、GPU 和 FPGA）编写高性能代码。它建立在 OpenCL 之上，并提供了一个 C++ 抽象层，使开发者能够使用熟悉的 C++ 语言结构和库来编写这些异构系统的代码。SYCL 模型旨在与硬件无关，意味着相同的代码将适用于不同的硬件平台。它还提供了对数据并行性和任务并行性的支持，适用于各种应用，包括科学计算、机器学习和计算机视觉。

oneAPI tools 培训，我知道了 oneAPI tools 的目标是为开发者提供一个统一的编程模型和工具集，使他们能够轻松地将应用程序部署到不同的硬件平台上，并实现最佳的性能和能效，有 oneAPI 编译器、oneAPI 分析器、oneAPI 库、oneAPI 工具箱。