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

高斯消去法的 GPU 编程

姓名：姚知言

年级：2022 级

专业：计算机科学与技术

指导教师：王刚

2024 年 6 月 10 日

摘要

本次实验围绕 GPU 编程进行学习和时间，第一部分在 Intel Devcloud 网站上学习了 oneAPI 和 SYCL 等相关 GPU 编程知识及范式。第二部分尝试进行了 CUDA 的高斯消元编程。

关键字：并行，GPU，oneAPI，SYCL，CUDA

目录

一、 Intel DevCloud 网站学习

（一） oneAPI Intro

1. 学习笔记

a) oneAPI

在当前环境下，由于没有通用的编程语言或 API，不同的硬件结构往往需要使用不同的语言和库进行编程，这将大大增加开发人员的学习成本，降低复用性。

oneAPI 通过提供统一的编程模型来简化不同模型开发的成本。它为我们提供了用于表达并行性的统一和简化的语言和库，并在一系列硬件（包括 CPU、GPU、FPGA）上提供的原生高级语言性能，如图1所示。

b) SYCL

SYCL 是一项 Khronos 标准，是基于 OpenCL 构建的 C++ 异构并行编程框架。SYCL 的发明不仅减轻了程序员的学习压力，更使得编译器能够更好的分析和优化程序。

c) DPC++

数据并行 C++（DPC++）是 oneAPI 的 SYCL 编译器实现，结合了 C++ 和 SYCL 的优势。DPC++ 可以在主机上调用 SYCL 程序，并将计算卸载到加速器。这一功能由程序员来调控，通过使用熟悉的 C++ 和库结构，并添加一些功能，如用于工作目标的队列、用于数据管理的缓冲区和用于并行性的 parallel_for，指示应卸载计算和数据的哪些部分。

d) OneAPI 的 HPC 单结点工作流

整体结构如图2所示。加速代码可以通过 SYCL 或指令编写。通过 DPC++ 可以完成从 CUDA 到 SYCL 的迁移。Fortran 可以在 OpenMP 中使用指令的样式。C++ 程序扩展了基于指令的样式选项，此外，OpenCL 应用程序可以保留 OpenCL 语言或迁移到 SYCL。

e) OneAPI 编程模型

分为平台模型，执行模型，内存模型和内核编程模型四个类型。

平台模型基于 SYCL 平台模型。通常是执行程序主要部分（特别是应用程序作用域和命令组作用域）的基于 CPU 的系统。

执行模型基于 SYCL 执行模型，主要作用为定义并指定代码（称为内核）如何在设备上执行以及与控制主机交互，通过命令组协调主机和设备之间的执行和数据管理。

内存模型基于 SYCL 内存模型，定义了主机和设备如何与内存交互，协调主机和设备之间的内存分配和管理，从而适应不同主机和设备配置。

内核编程模型基于 SYCL 内核编程模型。它支持主机和设备之间的由程序员指定的显式并行执行方式。

f) SYCL 程序的编译与运行

通常，在本地环境下运行 oneAPI SYCL 程序，需要通过以下步骤。

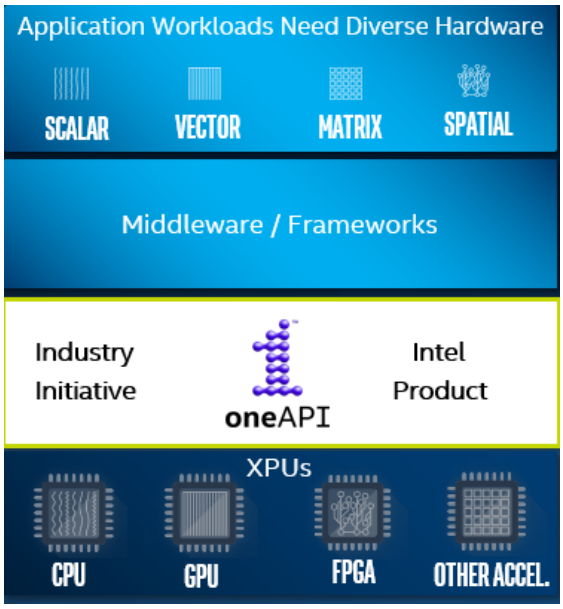


图 1: oneAPI

运行 SYCL 程序

```
source /opt/intel/inteloneapi/setvars.sh
icpx -fsycl simple.cpp -o simple
./simple
```

2. 练习完成结果

该部分练习完成结果如图3和图4所示。

(二) SYCL Program Structure

1. 学习笔记

a) 设备

设备可以实现 oneAPI 的系统加速器，包含用于查询有关设备的信息的成员函数。设备类常用于创建多个设备的 SYCL 程序。

b) 设备选择器

主要作用是在运行时选择特定设备，以根据用户的个性化需求执行内核。

c) 队列

队列用于提交要由 SYCL 运行时执行的命令组，是一种将工作提交到设备的机制。一个队列映射到一个设备，多个队列可以映射到同一设备，如图5所示。

d) 内核

内核类在调用内核调度函数的时候调用，封装了用于在实例化命令组时在设备上执行代码的方法和数据。

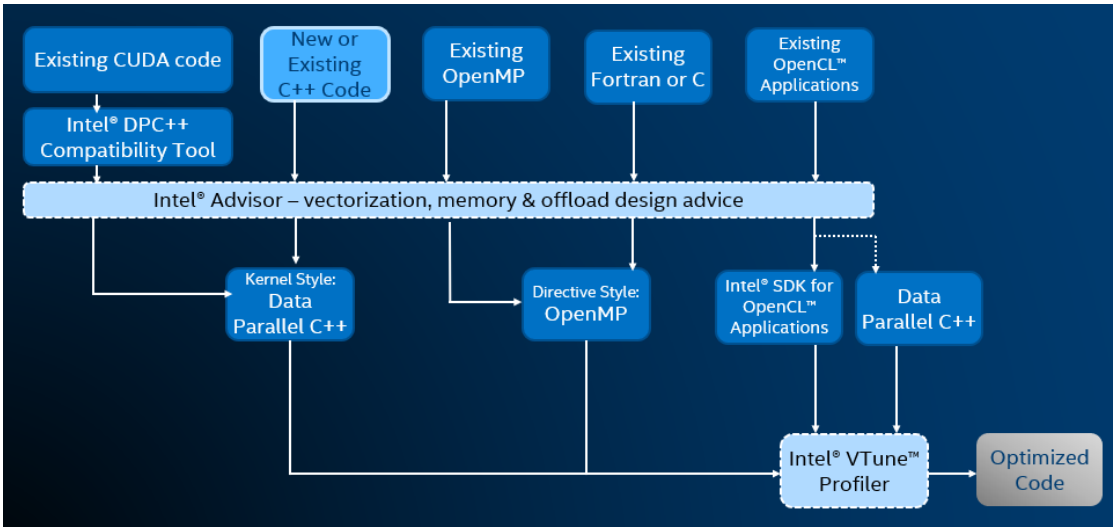


图 2: OneAPI 的 HPC 单结点工作流

```
[1]: %%writefile lab/simple.cpp
//==============================================================
// Copyright © Intel Corporation
//
// SPDX-License-Identifier: MIT
// ==============================================================
#include <sycl/sycl.hpp>
using namespace sycl;
static const int N = 16;
int main(){
  //# define queue which has default device associated for offload
  queue q;
  std::cout << "Device: " << q.get_device().get_info<info::device::name>() << "\n";

  //# Unified Shared Memory Allocation enables data access on host and device
  int *data = malloc_shared<int>(N, q);

  //# Initialization
  for(int i=0; i<N; i++) data[i] = i;

  //# Offload parallel computation to device
  q.parallel_for(range<1>(N), [=] (id<1> i){
    data[i] *= 2;
  }).wait();

  //# Print Output
  for(int i=0; i<N; i++) std::cout << data[i] << "\n";

  free(data, q);
  return 0;
}
Overwriting lab/simple.cpp
```

图 3: oneAPI intro 练习完成结果 1

```
[2]: ! chmod 755 q; chmod 755 run_simple.sh;if [ -x "$(command -v qsub)" ]; then ./q run_simple.sh; else ./run_simple.sh; fi
Job has been submitted to Intel(R) DevCloud and will execute soon.

Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2552672.v-qsvr-1           ...ub-singleuser u223006         00:00:25 R jupyterhub
2552678.v-qsvr-1           run_simple.sh    u223006                0 Q batch

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

########################################################################
#      Date:           Sat Jun  8 09:34:49 PM PDT 2024
#    Job ID:           2552678.v-qsvr-1.aidevcloud
#      User:           u223006
# Resources:           cput=75:00:00,neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime=06:00:00
########################################################################

## u223006 is compiling SYCL_Essentials Module1 -- oneAPI Intro sample - 1 of 1 simple.cpp
Device: Intel(R) UHD Graphics
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
```

图 4: oneAPI intro 练习完成结果 2

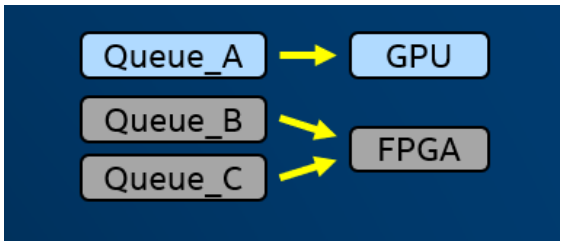


图 5: 队列的工作机制

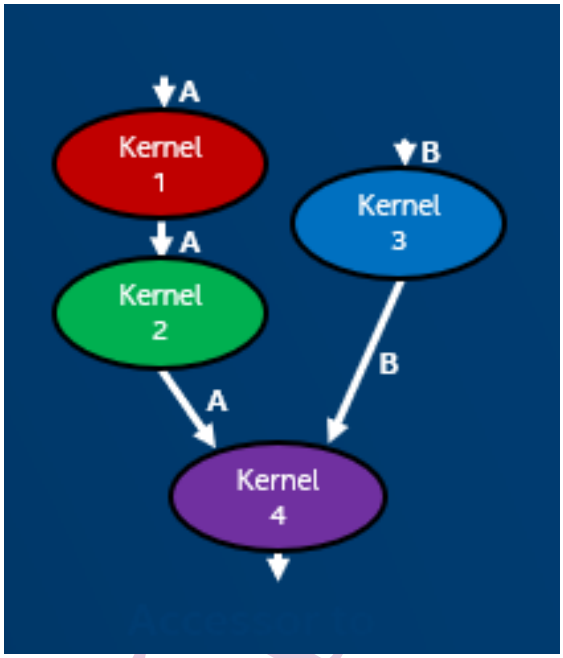


图 6: 访问器的数据以来关系

e) SYCL 语言运行

SYCL 语言运行由 C++ 类，模板和库构成。

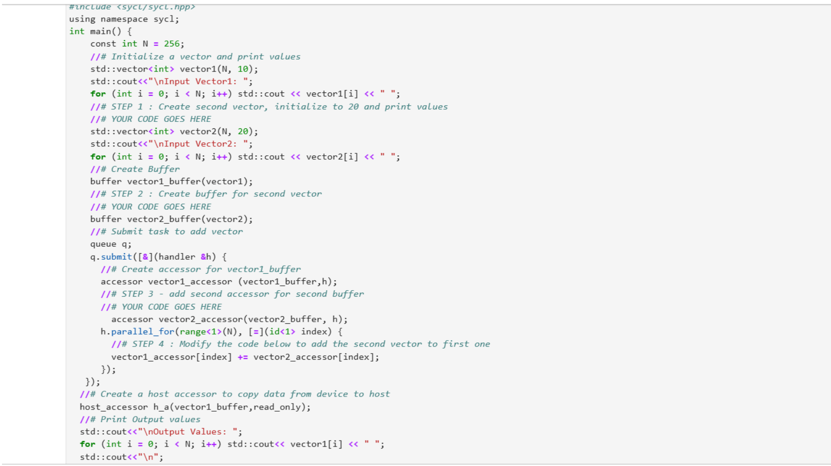
应用程序作用域和命令组作用域在主机上执行，可以使用全部的 c++ 功能。内核在设备上执行，可以执行的 c++ 功能存在限制。

f) 并行内核

分为基本并行内核和 ND-Range 并行内核（一个 for 循环的简单形式）。内核对于卸载 for 循环使其同步并行执行非常有意义。

g) 缓冲区与访问器

缓冲区将数据封装在设备和主机的 SYCL 应用程序中，通过访问器访问。如图6所示，访问器在 SYCL 图中创建对内核执行进行排序的数据依赖关系。如果两个内核使用相同的缓冲区，则第二个内核需要等待第一个内核完成以避免争用条件。



```
#include <sycl/sycl.hpp>
using namespace sycl;
int main() {
    const int N = 256;
    //# Initialize a vector and print values
    std::vector<int> vector1(N, 10);
    std::cout<<"\nInput Vector1: ";
    for (int i = 0; i < N; i++) std::cout << vector1[i] << " ";
    //# STEP 1 : Create second vector, initialize to 20 and print values
    //# YOUR CODE GOES HERE
    std::vector<int> vector2(N, 20);
    std::cout<<"\nInput Vector2: ";
    for (int i = 0; i < N; i++) std::cout << vector2[i] << " ";
    //# Create Buffer
    buffer vector1_buffer(vector1);
    //# STEP 2 : Create buffer for second vector
    //# YOUR CODE GOES HERE
    buffer vector2_buffer(vector2);
    //# Submit task to add vector
    queue q;
    q.submit([&](handler &h) {
      //# Create accessor for vector1_buffer
      accessor vector1_accessor (vector1_buffer,h);
      //# STEP 3 - add second accessor for second buffer
      //# YOUR CODE GOES HERE
        accessor vector2_accessor(vector2_buffer, h);
      h.parallel_for(range<1>(N), [=](id<1> index) {
        //# STEP 4 : Modify the code below to add the second vector to first one
        vector1_accessor[index] += vector2_accessor[index];
      });
   });
  //# Create a host accessor to copy data from device to host
  host_accessor h_a(vector1_buffer,read_only);
  //# Print Output values
  std::cout<<"\nOutput Values: ";
  for (int i = 0; i < N; i++) std::cout<< vector1[i] << " ";
  std::cout<<"\n";
```

图 7: SYCL Program Structure 练习完成结果 1

```
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2552862.v-qsvr-1           ...ub-singleuser u223006         00:00:39 R jupyterhub
2552930.v-qsvr-1           ...vector_add.sh u223006         00:00:03 R batch
2552933.v-qsvr-1           ...vector_add.sh u223006                0 Q batch

Waiting for Output  Done⬇

########################################################################
#      Date:           Sun Jun  9 08:09:22 AM PDT 2024
#    Job ID:           2552933.v-qsvr-1.aidevcloud
#      User:           u223006
# Resources:           cput=75:00:00,neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime=06:00:00
########################################################################

## u223006 is compiling SYCL_Essentials Module2 -- SYCL Program Structure sample - 7 of 7 vector_add.cpp

Input Vector1: 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
Input Vector2: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
Output Values: 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30

########################################################################
```

图 8: SYCL Program Structure 练习完成结果 2

h) 主机访问器与缓冲区销毁

使用主机缓冲区访问目标的访问器。在命令组范围之外创建的，并且其访问到的数据将在主机上可用。这些用于通过构造主机访问器对象将数据同步回主机。缓冲区销毁是将数据同步回主机的另一种方法。

i) 自定义设备选择器

可以通过自己的需求，选择特定供应商/名称/优先级的设备。

2. 练习完成结果

该部分代码实现如图7所示，运行结果如图8所示。

（三） Unified Shared Memory

1. 学习笔记

统一共享内存（USM）是 SYCL 中的一种内存管理。USM 通过基于指针的方法，使用 c 语言中的 malloc 或者 c++ 语言中的 new 进行内存分配，并移植现有代码到 SYCL。是否使用

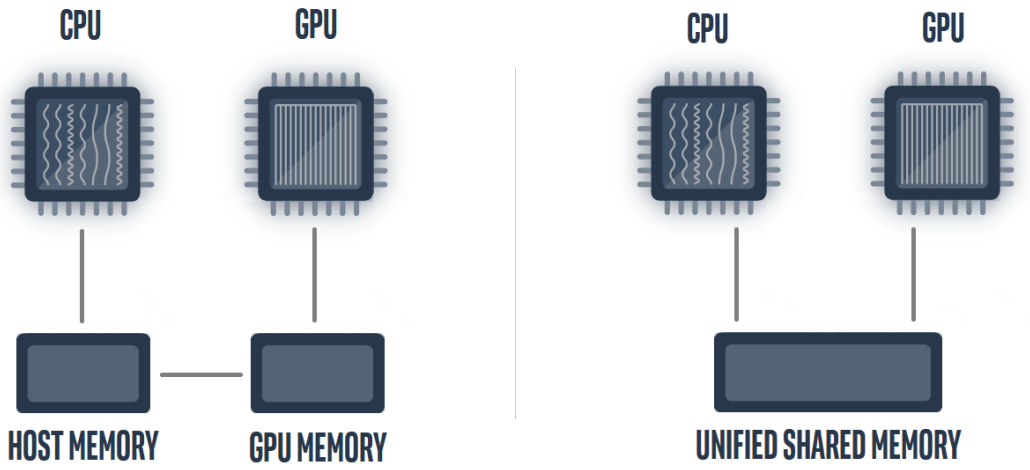


图 9: 是否使用 USM 的区别

USM 的区别如图9所示。

USM 的类型有以下几种，显式的设备上的分配 malloc_device（仅可以在设备上访问），隐式的主机上的分配 malloc_host（可以在设备和主机上访问），隐式的共享分配 malloc_shared（可以在设备和主机上访问）。

a) 语法

运行 SYCL 程序

```
//以共享分配为例
//分配方式1
int *data = malloc_shared<int>(N, q);
//分配方式2
int *data = static_cast<int *>(malloc_shared(N * sizeof(int), q));
//释放方式
free(data, q);
```

b) 解决数据依赖的方式

- q.wait(): 在上一个任务执行完再开始下一个任务。
- in_order queue: 规定任务执行顺序。
- depends_on: 规定任务开始之前必须完成的动作。

2. 练习完成结果

该部分代码实现如图10和图11所示，运行结果如图12所示。

```
[13]: %%writefile lab/usm_lab.cpp
//==============================================================
// Copyright © Intel Corporation
//
// SPDX-License-Identifier: MIT
// ==============================================================
#include <sycl/sycl.hpp>
using namespace sycl;
static const int N = 1024;
int main() {
  queue q;
  std::cout << "Device : " << q.get_device().get_info<info::device::name>() << "\n";
  //intialize 2 arrays on host
  int *data1 = static_cast<int *>(malloc(N * sizeof(int)));
  int *data2 = static_cast<int *>(malloc(N * sizeof(int)));
  for (int i = 0; i < N; i++) {
    data1[i] = 25;
    data2[i] = 49;
  }
  //# STEP 1 : Create USM device allocation for data1 and data2
  //# YOUR CODE GOES HERE
  int *d_data1 = malloc_device<int>(N, q);
  int *d_data2 = malloc_device<int>(N, q);
  //# STEP 2 : Copy data1 and data2 to USM device allocation
  //# YOUR CODE GOES HERE
  q.memcpy(d_data1, data1, N * sizeof(int)).wait();
  q.memcpy(d_data2, data2, N * sizeof(int)).wait();
  //# STEP 3 : Write kernel code to update data1 on device with sqrt of value
  q.parallel_for(N, [=](auto i) {
    //# YOUR CODE GOES HERE
    d_data1[i] = static_cast<int>(sqrtf(d_data1[i]));
  }).wait();
```

图 10: Unified Shared Memory 练习完成结果 1

```
  q.memcpy(d_data2, data2, N * sizeof(int)).wait();
  //# STEP 3 : Write kernel code to update data1 on device with sqrt of value
  q.parallel_for(N, [=](auto i) {
    //# YOUR CODE GOES HERE
    d_data1[i] = static_cast<int>(sqrtf(d_data1[i]));
  }).wait();
  //# STEP 3 : Write kernel code to update data2 on device with sqrt of value
  q.parallel_for(N, [=](auto i) {
    //# YOUR CODE GOES HERE
    d_data2[i] = static_cast<int>(sqrtf(d_data2[i]));
  }).wait();
  //# STEP 5 : Write kernel code to add data2 on device to data1
  q.parallel_for(N, [=](auto i) {
  //# YOUR CODE GOES HERE
     d_data1[i] += d_data2[i];
  }).wait();
  //# STEP 6 : Copy data1 on device to host
  //# YOUR CODE GOES HERE
  q.memcpy(data1, d_data1, N * sizeof(int)).wait();
  //# verify results
  int fail = 0;
  for (int i = 0; i < N; i++) if(data1[i] != 12) {fail = 1; break;}
  if(fail == 1) std::cout << " FAIL"; else std::cout << " PASS";
  std::cout << "\n";
  //# STEP 7 : Free USM device allocations
  //# YOUR CODE GOES HERE
  free(d_data1, q);
  free(d_data2, q);
  free(data1);
  free(data2);
  //# STEP 8 : Add event based kernel dependency for the Steps 2 - 6
  return 0;
}

Overwriting lab/usm_lab.cpp
```

图 11: Unified Shared Memory 练习完成结果 2

```
[14]: ! chmod 755 run_usm_lab.sh; if [ -x "$(command -v qsub)" ]; then ./q run_usm_lab.sh; else ./run_usm_lab.sh; fi

Job has been submitted to Intel(R) DevCloud and will execute soon.

Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2552862.v-qsvr-1           ...ub-singleuser u223006          00:01:05 R jupyterhub
2552961.v-qsvr-1           run_usm_lab.sh   u223006                0 Q batch

Waiting for Output ################ Done!

########################################################################
#      Date:           Sun Jun  9 09:18:11 AM PDT 2024
#    Job ID:           2552961.v-qsvr-1.aidevcloud
#      User:           u223006
# Resources:           cput=75:00:00,neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime=06:00:00
########################################################################

## u223006 is compiling SYCL_Essentials Module3 -- SYCL Unified Shared Memory - 5 of 5 usm_lab.cpp
Device : Intel(R) UHD Graphics
 PASS

########################################################################
# End of output for job 2552961.v-qsvr-1.aidevcloud
# Date: Sun Jun  9 09:18:21 AM PDT 2024
########################################################################

Job Completed in 20 seconds.
```

图 12: Unified Shared Memory 练习完成结果 3

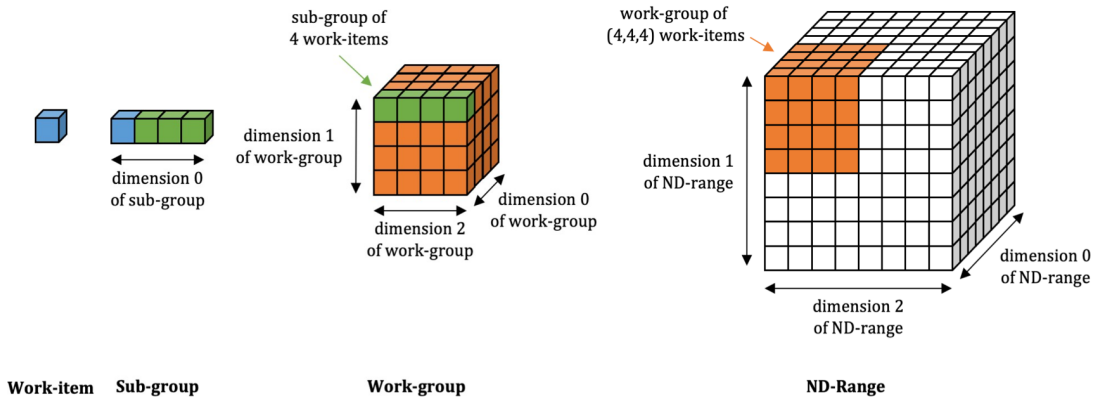


图 13: ND-range 内核

（四） Optional: SYCL Sub Groups

1. 学习笔记

在许多现代硬件平台上，工作组的一些子集是同时执行的或具有一定的执行计划顺序的。这些工作项子集称为子组。利用子组的技术有助于将执行映射到低级硬件，以实现更高的性能。

a) ND-Range 并行内核中的子组

ND-Range 内核的并行执行有助于对映射到硬件资源的工作项进行分组。有关于工作项，子组和工作组的关系以及 ND-Range 的运行方式如图13所示。

b) 子组的优势

- 子组中的工作项可以使用随机操作直接进行通信，而无需显式内存操作。
- 子组中的工作项可以使用子组屏障进行同步，并使用子组内存屏障来保证内存一致性。
- 子组中的工作项可以访问子组函数和算法，从而提供常见并行模式的快速实现。

c) 常用函数

- get_sub_group() 获取子组句柄
- get_local_id() 获取子组工作项索引
- get_local_range() 返回子组大小
- get_group_id() 返回子组索引
- get_group_range() 返回父工作组中子组数量

此外，子组中还有很多函数与算法 select_by_group，shift_group_left，shift_group_right，permute_group_by_xor，group_broadcast，reduce_over_group，exclusive_scan_over_group，inclusive_scan_over_group，any_of_group，all_of_group，none_of_group 等。通过灵活运用可以显著提高开发效率。

其中，较为有用的功能是隐式的完成子组内的通信。即 select_by_group(sg, x, id)，

```
  std::cout << "Device : " << q.get_device().get_info<info::device::name>() << "\n";
  //# allocate USM shared allocation for input data array and sg_data array
  int *data = malloc_shared<int>(N, q);
  int *sg_data = malloc_shared<int>(N/S, q);
  //# initialize input data array
  for (int i = 0; i < N; i++) data[i] = i;
  for (int i = 0; i < N; i++) std::cout << data[i] << " ";
  std::cout << "\n\n";
  //# Kernel task to compute sub-group sum and save to sg_data array
  //# STEP 1 : set fixed sub_group size of value S in the kernel below
  q.parallel_for(nd_range<1>(N, B), [=](nd_item<1> item) {
    auto sg = item.get_sub_group();
    auto i = item.get_global_id(0);
    //# STEP 2: Add all elements in sub_group using sub_group reduce
    //# YOUR CODE GOES HERE
    int sub_group_sum = reduce_over_group(sg, data[i], plus<>());
    //# STEP 3 : save each sub-group sum to sg_data array
    //# YOUR CODE GOES HERE
    if (sg.leader()) {
      sg_data[item.get_group(0) * sg.get_group_range()[0] + sg.get_group_id()[0]] = sub_group_sum;
    }
  }).wait();
  //# print sg_data array
  for (int i = 0; i < N/S; i++) std::cout << sg_data[i] << " ";
  std::cout << "\n";
  //# STEP 4: compute sum of all elements in sg_data array
  //# YOUR CODE GOES HERE
  int sum = std::accumulate(sg_data, sg_data + N/S, 0);
  std::cout << "\nSum = " << sum << "\n";
  //# free USM allocations
  free(data, q);
  free(sg_data, q);
  return 0;
}
Overwriting lab/sub_group_lab.cpp
```

图 14: SYCL Sub Groups 练习完成结果 1

```
[16]: ! chmod 755 run_sub_group_lab.sh; if [ -x "$(command -v qsub)" ]; then ./q run_sub_group_lab.sh; else ./run_sub_group_lab.sh; fi

Job has been submitted to Intel(R) DevCloud and will execute soon.

Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
2552862.v-qsvr-1           ...ub-singleuser u223006         00:01:20 R jupyterhub
2552971.v-qsvr-1           ..._group_lab.sh u223006                0 Q batch

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

########################################################################
#      Date:           Sun Jun  9 09:35:01 AM PDT 2024
#    Job ID:           2552971.v-qsvr-1.aidevcloud
#      User:           u223006
# Resources:           cput=75:00:00,neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime=06:00:00
########################################################################

## u223006 is compiling SYCL_Essentials Module4 -- SYCL Sub Groups - 7 of 7 sub_group_lab.cpp
Device : Intel(R) UHD Graphics
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 5
1 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 9
9 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 13
5 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 17
1 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 20
7 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 24
3 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 27
9 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 31
5 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 35
1 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 38
7 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 42
3 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 45
9 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 49
5 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 53
1 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 56
7 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 60
```

图 15: SYCL Sub Groups 练习完成结果 2

shift_group_left(sg, x, delta)，shift_group_right(sg, x, delta)，permute_group_by_xor(sg, x, mask) 函数。

2. 练习完成结果

该部分代码实现如图14所示，运行结果如图15和图16所示。

二、 高斯消元法在 CUDA 中的优化

(一) 算法设计

利用 CUDA 的编程范式对高斯消元法进行改进，以感受 GPU 编程的高效率。

(二) 编程实现

运行 SYCL 程序

```
1 #include <iostream>
2 #include <cuda_runtime.h>
```

```
7 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 42
3 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 45
9 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 49
5 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 53
1 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 56
7 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 60
3 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 63
9 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 67
5 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 71
1 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 74
7 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 78
3 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 81
9 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 85
5 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 89
1 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 92
7 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 96
3 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 99
9 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023

496 1520 2544 3568 4592 5616 6640 7664 8688 9712 10736 11760 12784 13808 14832 15856 16880 17904 18928 19952 20976 22000 23024 24048 25072 26096
27120 28144 29168 30192 31216 32240

Sum = 523776

################################################################################
# End of output for job 2552971.v-qsvr-1.aidevcloud
# Date: Sun Jun  9 09:35:16 AM PDT 2024
################################################################################

Job Completed in 21 seconds.
```

图 16: SYCL Sub Groups 练习完成结果 3

```cpp
#include <chrono>
#include <algorithm>
#include <cstdlib>
#define N 1024
#define BLOCK_SIZE 32
__global__ void division_kernel(float* data, int k, int n) {
    int tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < n) {
        float element = data[k * n + k];
        float temp = data[k * n + tid];
        data[k * n + tid] = temp / element;
    }
}
__global__ void eliminate_kernel(float* data, int k, int n) {
    int tx = blockDim.x * blockIdx.x + threadIdx.x;
    if (tx == 0)
        data[k * n + k] = 1.0f;
    int row = k + 1 + blockIdx.x;
    while (row < n) {
        int tid = threadIdx.x;
        while (k + 1 + tid < n) {
            int col = k + 1 + tid;
            float temp_1 = data[row * n + col];
            float temp_2 = data[row * n + k];
            float temp_3 = data[k * n + col];
            data[row * n + col] = temp_1 - temp_2 * temp_3;
            tid = tid + blockDim.x;
        }
        __syncthreads();
        if (threadIdx.x == 0) {
            data[row * n + k] = 0;
        }
        row += gridDim.x;
    }
}
```

```
38 void cud(float* data_D, int n) {
39     dim3 grid((n + BLOCK_SIZE - 1) / BLOCK_SIZE);
40     dim3 block(BLOCK_SIZE);
41     cudaError_t ret;
42     for (int k = 0; k < n; k++) {
43         division_kernel<<<grid, block>>>(data_D, k, n);
44         cudaDeviceSynchronize();
45         ret = cudaGetLastError();
46         if (ret != cudaSuccess) {
47             printf("division_kernel failed, %s\n", cudaGetErrorString(ret));
48         }
49 
50         eliminate_kernel<<<grid, block>>>(data_D, k, n);
51         cudaDeviceSynchronize();
52         ret = cudaGetLastError();
53         if (ret != cudaSuccess) {
54             printf("eliminate_kernel failed, %s\n", cudaGetErrorString(ret));
55         }
56     }
57 }
58 void common(float* data, int n) {
59     for (int k = 0; k < n; k++) {
60         float element = data[k * n + k];
61         for (int j = k; j < n; j++) {
62             data[k * n + j] /= element;
63         }
64         for (int i = k + 1; i < n; i++) {
65             float factor = data[i * n + k];
66             for (int j = k; j < n; j++) {
67                 data[i * n + j] -= factor * data[k * n + j];
68             }
69             data[i * n + k] = 0;
70         }
71     }
72 }
73 void op(float* data_H, int n, int iterations, void(*elimination_func)(float*,
      int)) {
74     auto start = std::chrono::high_resolution_clock::now();
75     for (int i = 0; i < iterations; i++) {
76         float* data_copy = new float[n * n];
77         std::copy(data_H, data_H + n * n, data_copy);
78         elimination_func(data_copy, n);
79         delete[] data_copy;
80     }
81     auto end = std::chrono::high_resolution_clock::now();
82     std::chrono::duration<double> diff = end - start;
83     std::cout << "Time taken for " << iterations << " iterations: " << diff.
         count() << " seconds" << std::endl;
```

```
PS C:\Users\Oathyzy\Desktop> nvcc cuda.cu -o cuda
cuda.cu
nvcc error   : 'cudafe++' died with status 0xC0000005 (ACCESS_VIOLATION)
```

图 17: 报错信息

```
84  }
85  int main() {
86      float* data_H = new float[N * N];
87      for (int i = 0; i < N; i++) {
88          for (int j = 0; j < N; j++) {
89              data_H[i * N + j] = static_cast<float>(rand() % 100);
90          }
91      }
92      int iterations = 10;
93      std::cout << "CUDA:" << std::endl;
94      op(data_H, N, iterations, [](float* data, int n) {
95          float* data_D;
96          cudaMalloc((void**)&data_D, n * n * sizeof(float));
97          cudaMemcpy(data_D, data, n * n * sizeof(float),
                cudaMemcpyHostToDevice);
98          cud(data_D, n);
99          cudaMemcpy(data, data_D, n * n * sizeof(float),
                cudaMemcpyDeviceToHost);
100         cudaFree(data_D);
101     });
102     std::cout << "common:" << std::endl;
103     op(data_H, N, iterations, common);
104     delete[] data_H;
105     return 0;
106 }
```


(三) 性能测试

在本机运行 cuda1 的时候，会报 ASSESS_VIOLATION 的错误，如17所示。起初我认为是我的代码结构有问题，但后来 fork 了一些网上的 cuda 代码后都发生了相同的问题。尝试上网寻找解决方法，发现以前的一些 CUDA 版本也有类似的问题，并有人提供了解决方案。然而，对于这个较新的 CUDA 版本我并未能找到有效的资料。我严格按照 lab0 给出的步骤安装了 cuda，以下我给出我的一些系统参数。

显卡：NVIDIA GeForce GTX 1650

系统版本：Windows 11

CUDA 版本：12.4（NVIDIA 控制面板给出的兼容属性为 12.4.74）

三、 总结

在本次实验中，我学习了 OneAPI 和 SYCL 的相关知识，并完成了 CUDA 的编程尝试，在这个过程中，我增进了对 GPU 编程的理解。CUDA 相关代码可在GitHub 仓库中查看。