

软件系统优化A3实验报告

环境配置

实验平台

本次的实验我进行的平台是台式机，CPU为*Intel 13600kf*，GPU为*NVIDIA GeForce RTX 4070 Ti SUPE*。

cuda配置

由于之前进行过大模型的本地部署，因此安装过cuda，cuda版本如下

```
Anaconda Prompt
(base) C:\Users\Harry>nvidia-smi
Wed Nov 13 17:21:02 2024




+-----+
| NVIDIA-SMI 561.09                  | Driver Version: 561.09      | CUDA Version: 12.6   |
+-----+-----+
| GPU   Name      Perf          | Driver-Model  | Bus-Id        | Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap | Pwr:Usage/Cap |           |   Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
| 0  NVIDIA GeForce RTX 4070 ...  | WDDM          | 00000000:01:00:0  On  |          | 0%          Default |
| 0%   32C   PS      9W / 304W    |                | 872MiB / 16376MiB |          |             MIG M.   |
|=====+=====+
|                                     |                          |                   |
+-----+-----+

Processes:
+-----+-----+
| GPU   GI   CI       PID   Type   Process name                      | GPU Memory |
|   ID   ID   ID             |            | Usage      |
+-----+-----+
| 0  N/A  N/A   4236   C+G   ...ne\Binaries\Win64\EpicWebHelper.exe | N/A        |
| 0  N/A  N/A   7596   C+G   ...n\NVIDIA App\CEF\NVIDIA Overlay.exe | N/A        |
| 0  N/A  N/A   8964   C+G   ...8608\office6\promecfpluginhost.exe  | N/A        |
| 0  N/A  N/A  10192   C+G   ...8608\office6\promecfpluginhost.exe  | N/A        |
| 0  N/A  N/A  10260   C+G   ...cef\cef.win7x64\steamwebhelper.exe  | N/A        |
| 0  N/A  N/A  12064   C+G   ...les\Microsoft OneDrive\OneDrive.exe | N/A        |
| 0  N/A  N/A  12224   C+G   ...inaries\Win64\EpicGamesLauncher.exe | N/A        |
| 0  N/A  N/A  15612   C+G   ...Search_cw5nlh2txyewy\SearchApp.exe  | N/A        |
| 0  N/A  N/A  17904   C+G   ...5nlh2txyewy\ShellExperienceHost.exe | N/A        |
| 0  N/A  N/A  19544   C+G   ...oogle\Chrome\Application\chrome.exe | N/A        |
| 0  N/A  N/A  19676   C+G   ...5\extracted\runtime\WeChatAppEx.exe  | N/A        |
| 0  N/A  N/A  19872   C+G   ...64_8wekyb3d8bbwe\CalculatorApp.exe  | N/A        |
| 0  N/A  N/A  20132   C+G   ...ekyb3d8bbwe\PhoneExperienceHost.exe | N/A        |
| 0  N/A  N/A  21508   C+G   C:\Windows\explorer.exe                 | N/A        |
| 0  N/A  N/A  22164   C+G   ...n\NVIDIA App\CEF\NVIDIA Overlay.exe | N/A        |
| 0  N/A  N/A  22220   C+G   ...es (x86)\Microsoft VS Code\Code.exe  | N/A        |
| 0  N/A  N/A  22360   C+G   ...on\wallpaper_engine\wallpaper64.exe  | N/A        |
| 0  N/A  N/A  22656   C+G   ...8608\office6\promecfpluginhost.exe  | N/A        |
| 0  N/A  N/A  23472   C+G   ...t.LockApp_cw5nlh2txyewy\LockApp.exe  | N/A        |
| 0  N/A  N/A  23664   C+G   ...CBS_cw5nlh2txyewy\TextInputHost.exe  | N/A        |
| 0  N/A  N/A  23740   C+G   ...8608\office6\promecfpluginhost.exe  | N/A        |
| 0  N/A  N/A  24148   C+G   ...siveControlPanel\SystemSettings.exe  | N/A        |
| 0  N/A  N/A  26136   C+G   ...0_x64__kzh8wxbdkxb8p\DCv2\DCv2.exe  | N/A        |
+-----+-----+

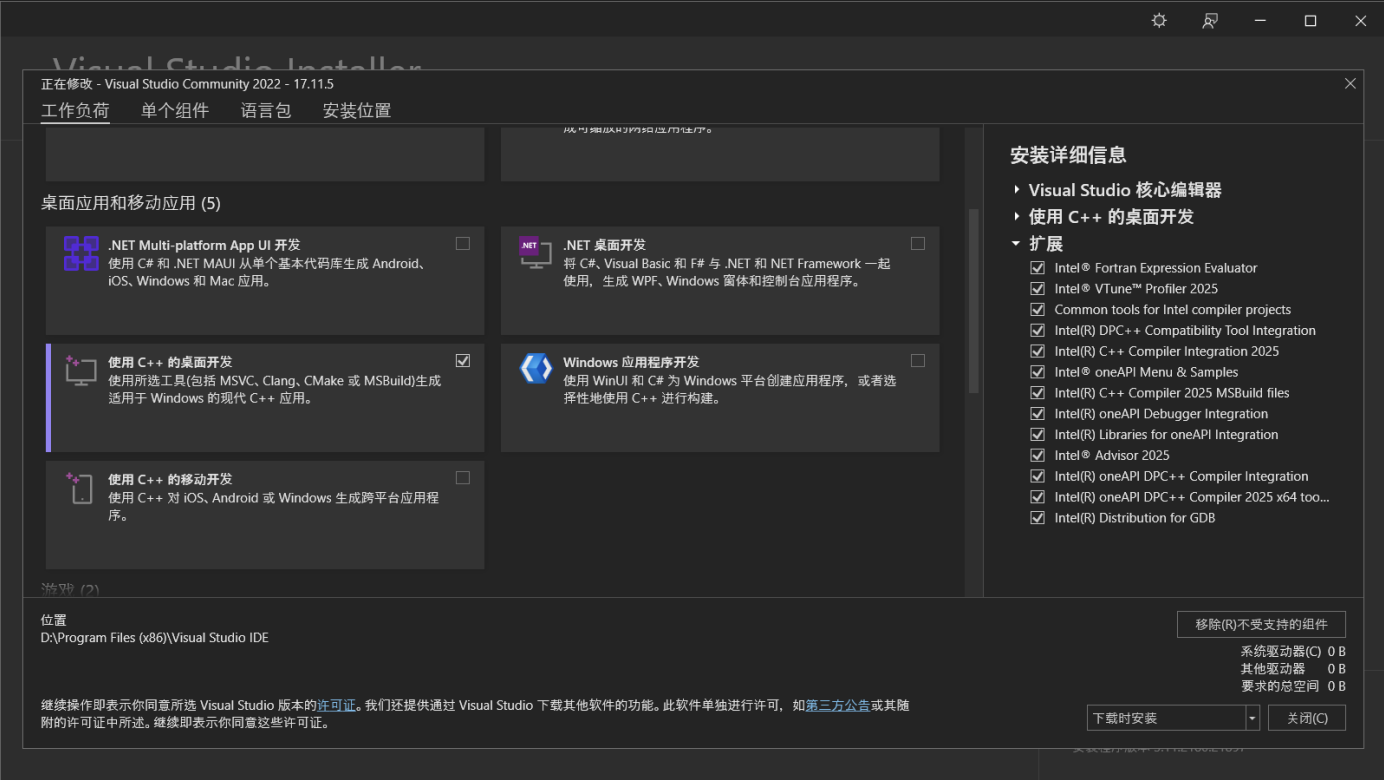
(base) C:\Users\Harry>
```

安装opeapi-base-toolkit和one-api-for-nvidia-gpus

下载后根据GUI完成安装

	intel-oneapi-base-toolkit-2025.0.0.8...	2024/11/12 23:56	文件夹	
	intel-oneapi-base-toolkit-2025.0.0.8...	2024/11/12 23:53	应用程序	2,967,535...
	oneapi-for-nvidia-gpus-2025.0.0-cud...	2024/11/12 23:54	Windows Install...	952 KB

安装Visual Studio并安装桌面c++开发环境



完成配置

完成上述的环境安装和配置后，我们运行环境配置脚本

```
Intel(r) oneAPI

0 N/A N/A 7596 C+G ...n\NVIDIA App\CEF\NVIDIA Overlay.exe N/A
0 N/A N/A 8964 C+G ...8608\office6\promecfpluginhost.exe N/A
0 N/A N/A 10192 C+G ...8608\office6\promecfpluginhost.exe N/A
0 N/A N/A 10260 C+G ...cef\cef.win7x64\steamwebhelper.exe N/A
0 N/A N/A 12064 C+G ...les\Microsoft OneDrive\OneDrive.exe N/A
0 N/A N/A 12224 C+G ...inaries\Win64\EpicGamesLauncher.exe N/A
0 N/A N/A 15612 C+G ...Search_cw5nlh2txyewy\SearchApp.exe N/A
0 N/A N/A 17904 C+G ...5nlh2txyewy\ShellExperienceHost.exe N/A
0 N/A N/A 19544 C+G ...oogle\Chrome\Application\chrome.exe N/A
0 N/A N/A 19676 C+G ...5\extracted\runtime\WeChatAppEx.exe N/A
0 N/A N/A 19872 C+G ...64_8wekyb3d8bbwe\CalculatorApp.exe N/A
0 N/A N/A 20132 C+G ...ekyb3d8bbwe\PhoneExperienceHost.exe N/A
0 N/A N/A 21508 C+G C:\Windows\explorer.exe N/A
0 N/A N/A 22164 C+G ...n\NVIDIA App\CEF\NVIDIA Overlay.exe N/A
0 N/A N/A 22220 C+G ...es (x86)\Microsoft VS Code\Code.exe N/A
0 N/A N/A 22360 C+G ...on\wallpaper_engine\wallpaper64.exe N/A
0 N/A N/A 22656 C+G ...8608\office6\promecfpluginhost.exe N/A
0 N/A N/A 23472 C+G ...t.LockApp_cw5nlh2txyewy\LockApp.exe N/A
0 N/A N/A 23664 C+G ...CBS_cw5nlh2txyewy\TextInputHost.exe N/A
0 N/A N/A 23740 C+G ...8608\office6\promecfpluginhost.exe N/A
0 N/A N/A 24148 C+G ...siveControlPanel\SystemSettings.exe N/A
0 N/A N/A 26136 C+G ..._0_x64_kzh8wxbdkxb8p\DCv2\DCv2.exe N/A

(base) C:\Users\Harry>"C:\Program Files (x86)\Intel\oneAPI\setvars.bat"
:: initializing oneAPI environment...
Initializing Visual Studio command-line environment...
Visual Studio version 17.11.5 environment configured.
"D:\Program Files (x86)\Visual Studio IDE\"
Visual Studio command-line environment initialized for: 'x64'
: advisor -- latest
: compiler -- latest
: dal -- latest
: debugger -- latest
: dev-utilities -- latest
: dnnl -- latest
: dpcpp-ct -- latest
: dpl -- latest
: ipp -- latest
: ippcp -- latest
: mkl -- latest
: ocloc -- latest
: pti -- latest
: tbb -- latest
: umf -- latest
: vtune -- latest
:: oneAPI environment initialized ::

(base) C:\Users\Harry>
```

成功运行

我们来编译并运行一下测试代码

```
Intel(R) oneAPI
<LOADER>[INFO]: unloaded adapter 0x00007FFB372F0000
<LOADER>[INFO]: unloaded adapter 0x00007FFB37290000
<LOADER>[INFO]: ----> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda async.cpp -o a.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>async.cpp(55,67): warning: format specifies type 'long' but the argument has type 'int64_t'
(aka 'long long') [-Wformat]
55     printf("\n Start CPU Computation, Number of Elems = %ld \n", N);
                                     ^~~~
                                     %lld
1 warning generated.
D:\Code\GitCode\oneAPI_course\code>async.cpp(55,67): warning: format specifies type 'long' but the argument has type 'int64_t'
(aka 'long long') [-Wformat]
55     printf("\n Start CPU Computation, Number of Elems = %ld \n", N);
                                     ^~~~
                                     %lld
1 warning generated.

D:\Code\GitCode\oneAPI_course\code>a.exe
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_level_zero.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB2F120000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_opencl.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB2FEB0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Start CPU Computation, Number of Elems = 10000000

CPU Computation, Time = 1.803700

GPU Computation, Time = 4.226500

Total Computation, Time = 4.226500

Task Done!
<LOADER>[INFO]: unloaded adapter 0x00007FFB2F120000
<LOADER>[INFO]: unloaded adapter 0x00007FFB2FEB0000
<LOADER>[INFO]: ----> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>
```

程序成功选择了 **NVIDIA GeForce RTX 4070 Ti SUPER** 作为计算设备，并且通过 CUDA 后端进行计算。

Part 1 实验课练习

任务1

学习**basic_parafor.cpp**代码，了解主机内存与设备内存（CPU、GPU）的分配、拷贝、释放等函数的应用，以及并行计算的 parallel_for 函数的使用

我们首先可以编译并运行一下**basic_parafor.cpp**

```
Intel(R) oneAPI
<LOADER>[INFO]: loaded adapter 0x00007FFB2FEB0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Start CPU Computation, Number of Elems = 10000000

CPU Computation, Time = 1.803700

GPU Computation, Time = 4.226500

Total Computation, Time = 4.226500

Task Done!
<LOADER>[INFO]: unloaded adapter 0x00007FFB2F120000
<LOADER>[INFO]: unloaded adapter 0x00007FFB2FEB0000
<LOADER>[INFO]: ---> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda basic_parafor.cpp -o basic_parafor.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>basic_parafor.exe
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_level_zero.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB268B0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_opencl.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB2FEB0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Data Result
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Task Done!
<LOADER>[INFO]: unloaded adapter 0x00007FFB268B0000
<LOADER>[INFO]: unloaded adapter 0x00007FFB2FEB0000
<LOADER>[INFO]: ---> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>
```

USM官方文档

Kinds of USM

USM builds upon Unified Addressing to define a shared address space where pointer values in this space always refer to the same location in memory. USM defines three types of memory allocations: `host`, `device`, and `shared`.

The following `enum` is used to refer to the different types of allocations inside of a SYCL program:

```
namespace sycl::usm {
    enum class alloc : /* unspecified */ {
        host,
        device,
        shared,
        unknown
    };
} // namespace sycl::usm
```

USM allocation type	Description
<code>host</code>	Allocations in host memory that are accessible by a device (in addition to the host).
<code>device</code>	Allocations in device memory that are not accessible by the host.
<code>shared</code>	Allocations in shared memory that are accessible by both host and device.

我们可以看到，官方将**USM**即**Undefined Shared memory**分为三种：

1. host

主机内存分配，这些内存可以被 **设备** 访问（除了主机本身之外）。

这种类型的内存分配是在主机（CPU）内存中进行的，但它同时允许设备（例如 GPU）访问这块内存。这种内存常用于主机和设备之间的数据传输或共享。

2. device

设备内存分配，这些内存 **不能** 被主机访问。

这种内存分配发生在设备内存中，例如 GPU 的显存，主机程序无法直接访问这些内存，只有设备（GPU 或其他加速器）能访问。设备内存通常用于存储需要在设备上处理的数据。

3. shared

共享内存分配，这些内存可以被 **主机和设备** 共同访问。

共享内存是一种特殊的内存类型，既能在主机（CPU）上使用，也能在设备（如 GPU）上使用。这种内存类型适用于主机和设备之间需要频繁交换数据的场景。

USM的分配

首先是分配主机内存的 `sycl::malloc_host`

我们可以查看**SYCL 官方文档**

`sycl::malloc_host`

```
template <typename T>
T* sycl::malloc_host(size_t count,
                    const sycl::context& syclContext,
                    const sycl::property_list& propList = {});

void* sycl::malloc_host(size_t numBytes,
                      const sycl::queue& syclQueue,
                      const sycl::property_list& propList = {});
```

我们可以看到**`basic_parafor.cpp`**使用的是 **`sycl::malloc_host`** with **`sycl::queue`** 中的 `T* sycl::malloc_host`

对于参数，我们可以查看Parameters

Parameters	
<code>alignment</code>	Alignment of allocated data.
<code>numBytes</code>	Allocation size in bytes.
<code>count</code>	Number of elements.
<code>syclDevice</code>	See sycl::device .
<code>syclQueue</code>	See sycl::queue .
<code>syclContext</code>	See sycl::context .
<code>propList</code>	Optional property list.

使用 `T* sycl::malloc_host` 来分配内存需要我们给定两个必须参数，一个是 `count`，用于给定元素的数量，另一个是 `syclQueue` 即分配内存的 SYCL 队列。

对于 `syclQueue`，我们来继续查看文档

`sycl::queue`

```
class queue;
```

The `sycl::queue` connect a host program to a single device. Programs submit tasks to a device via the `sycl::queue` and may monitor the `sycl::queue` for completion. A program initiates the task by submitting a command group to a `sycl::queue`. The command group defines a kernel function, the prerequisites to execute the kernel function, and an invocation of the kernel function on an index space. After submitting the command group, a program may use the `sycl::queue` to monitor the completion of the task for completion and errors. A `sycl::queue` can wait for all command groups that it has submitted by calling `wait` or `wait_and_throw`.

可以看到 `sycl::queue` 用于连接一个主机程序和一个单独的设备（实验中我们连接了 GPU），我们可以通过其来进行提交命令组、监控任务完成、同步与异步。

因此通过***basic_parafor.cpp***中的

```
int *host_mem = malloc_host<int>(N, my_gpu_queue);
```

我们为GPU在主机内存中分配了N个int型大小的空间并获得了其指针。

sycl::malloc_device

```
void* sycl::malloc_device(size_t numBytes,
                          const sycl::device& syclDevice,
                          const sycl::context& syclContext,
                          const sycl::property_list& propList = {});

template <typename T>
T* sycl::malloc_device(size_t count,
                      const sycl::device& syclDevice,
                      const sycl::context& syclContext,
                      const sycl::property_list& propList = {});

void* sycl::malloc_device(size_t numBytes,
                          const sycl::queue& syclQueue,
                          const sycl::property_list& propList = {});
```

参数与使用方法和 `sycl::malloc_host` 相同

因此我们通过

```
int *device_mem = malloc_device<int>(N, my_gpu_queue);
```

为GPU在GPU内存中分配了N个int型大小的空间并获得了其指针。需要注意的是其分配的内存只能由设备（本实验中为GPU）直接访问，而不能直接由主机程序访问。

USM的拷贝

官方文档如下

memcpy

```
void memcpy(void* dest, const void* src, size_t numBytes);
```

Copies `numBytes` of data from the pointer `src` to the pointer `dest`. The `dest` and `src` parameters must each either be a host pointer or a pointer within a USM allocation that is accessible on the handler's device. If a pointer is to a USM allocation, that allocation must have been created from the same context as the handler's queue.

For more detail on USM, please see [USM Basic Concept](#).

可见该函数用于在USM拷贝内存，我们需给定参数 `dest` 和 `src` 指针，并给定拷贝数据的大小。

该函数在***basic_parafor.cpp***中使用如下

```
// Copy from host(CPU) to device(GPU)
my_gpu_queue.memcpy(device_mem, host_mem, N * sizeof(int)).wait();
```

这里的 `wait` 指的是等待拷贝完成后再继续执行接下来的代码。

wait

```
void wait();
```

Wait for the event and the command associated with it to complete.

如此我们将 `host` 内存中的内容拷贝到了 `device` 内存中。

USM的释放

官方文档如下

```
sycl::free
```

```
void sycl::free(void* ptr, const sycl::context& syclContext);  
void sycl::free(void* ptr, const sycl::queue& syclQueue);
```

Frees an allocation. The memory pointed to by `ptr` must have been allocated using one of the USM allocation routines. `syclContext` must be the same `sycl::context` that was used to allocate the USM memory. The memory is freed without waiting for commands operating on it to be completed. If commands that use this memory are in-progress or are enqueued the behavior is undefined. In the second form the context is automatically extracted from the `syclQueue` that is passed in, as a convenience.

我们可以发现该函数有两种使用方式

一种是传入指定了内存分配时使用的上下文 `syclContext` & `sycl::context`

另一种是传入传入队列对象 `syclQueue` & `sycl::queue`

显然在`basic_parafor.cpp`中我们使用第二种

```
free(host_mem, my_gpu_queue);
free(device_mem, my_gpu_queue);
```

任务2

修改basic_parafor.cpp代码，将申请的内存空间修改为本机与设备共享内存空间。并思考主机内存和设备内存修改为共享内存空间的好处。

修改代码为共享内存空间

我们首先通过官方文档来看怎么来申请共享内存空间

sycl::malloc_shared

[illegible]

```

        const sycl::property_list& propList = {}));

template <typename T>
T* sycl::malloc_shared(size_t count,
    const sycl::device& syclDevice,
    const sycl::context& syclContext,
    const sycl::property_list& propList = {}));

void* sycl::malloc_shared(size_t numBytes,
    const sycl::queue& syclQueue,
    const sycl::property_list& propList = {}));

template <typename T>
T* sycl::malloc_shared(size_t count,
    const sycl::queue& syclQueue,
    const sycl::property_list& propList = {}));

void* sycl::aligned_alloc_shared(size_t alignment, size_t numBytes,
    const sycl::device& syclDevice,
    const sycl::context& syclContext,
    const sycl::property_list& propList =
{});

template <typename T>
T* sycl::aligned_alloc_shared(size_t alignment, size_t count,
    const sycl::device& syclDevice,
    const sycl::context& syclContext,
    const sycl::property_list& propList =
{});

void* sycl::aligned_alloc_shared(size_t alignment, size_t numBytes,
    const sycl::queue& syclQueue,
    const sycl::property_list& propList =
{});

template <typename T>

```

```
T* sycl::aligned_alloc_shared(size_t alignment, size_t count,
                             const sycl::queue& syclQueue,
                             const sycl::property_list& propList =
{});
```

Parameters

<code>alignment</code>	Alignment of allocated data.
<code>numBytes</code>	Allocation size in bytes.
<code>count</code>	Number of elements.
<code>syclDevice</code>	See sycl::device .
<code>syclQueue</code>	See sycl::queue .
<code>syclContext</code>	See sycl::context .
<code>propList</code>	Optional property list.

On successful USM shared allocation, these functions return a pointer to the newly allocated memory, which must eventually be deallocated with `sycl::free` in order to avoid a memory leak. If there are not enough resources to allocate the requested memory, these functions return `nullptr`.

When the allocation size is zero bytes (`numBytes` or `count` is zero), these functions behave in a manner consistent with C++ `std::malloc`. The value returned is unspecified in this case, and the returned pointer may not be used to access storage. If this pointer is not null, it must be passed to `sycl::free` to avoid a memory leak.

使用方法类似，我们只需要给定申请的元素类型、元素个数以及SYCL队列，同时我们也不必再对内存数据进行拷贝。注意最后需要释放内存。如此我们代码修改如下

```
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;

constexpr int N = 10;

int main() {
    queue my_gpu_queue(sycl::gpu_selector_v);

    std::cout << "Selected GPU device: " <<
        my_gpu_queue.get_device().get_info<info::device::name>() <<
        "\n";
```

```

// 申请共享内存，主机和设备共享此内存
int *shared_mem = malloc_shared<int>(N, my_gpu_queue);

for (int i = 0; i < N; i++) {
    shared_mem[i] = i;
}

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

    h.parallel_for(range{N}, [=](id<1> item) {
        shared_mem[item] *= 2;
    });

});

my_gpu_queue.wait();

printf("\nData Result\n");
for (int i = 0; i < N; i++) {
    printf("%d, ", shared_mem[i]);
}
printf("\nTask Done!\n");

free(shared_mem, my_gpu_queue);

return 0;
}

```

我们进行编译并运行

```
Intel(r) oneAPI

D:\Code\GitCode\oneAPI_course\code>basic_parafor.exe
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_level_zero.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB268B0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_opengl.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB2FEB0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Data Result
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Task Done!
<LOADER>[INFO]: unloaded adapter 0x00007FFB268B0000
<LOADER>[INFO]: unloaded adapter 0x00007FFB2FEB0000
<LOADER>[INFO]: ---> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda basic_parafor_sharemem.cpp -o basic_parafor_share_m
.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>basic_parafor_share_mem.exe
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_level_zero.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB268B0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_opengl.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB2FEB0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Data Result
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Task Done!
<LOADER>[INFO]: unloaded adapter 0x00007FFB268B0000
<LOADER>[INFO]: unloaded adapter 0x00007FFB2FEB0000
<LOADER>[INFO]: ---> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>
```

成功完成修改

共享内存的好处

共享内存是主机和设备之间的一种高效数据交换机制，它的主要好处包括：

- 提高数据访问效率和减少延迟。
- 简化主机和设备之间的数据传输。
- 减少内存开销并优化内存使用。
- 提升并行计算性能。
- 增强设备资源的利用率。
- 简化数据管理和内存分配。

共享内存不仅适用于主机和设备之间的数据传输，还大大提高了计算性能，尤其是在需要并行计算的场景中。

这里我们可以进行一个对比测试，我们结合一下前面的两种内存的代码，并计时

```
#include <sycl/sycl.hpp>
#include <iostream>
#include <chrono>

using namespace sycl;
constexpr int N = 10;

void test_malloc_host_device(queue& my_gpu_queue) {
    auto start = std::chrono::high_resolution_clock::now();

    int* host_mem = malloc_host<int>(N, my_gpu_queue);
    int* device_mem = malloc_device<int>(N, my_gpu_queue);

    // Init CPU data
    for (int i = 0; i < N; i++) {
        host_mem[i] = i;
    }

    // Copy from host(CPU) to device(GPU)
    my_gpu_queue.memcpy(device_mem, host_mem, N * sizeof(int)).wait();

    // Submit the content to the queue for execution
    my_gpu_queue.submit([&](handler& h) {
        h.parallel_for(range{N}, [=](id<1> item) {
            device_mem[item] *= 2;
        });
    });

    // Wait the computation done
    my_gpu_queue.wait();

    // Copy back from GPU to CPU
    my_gpu_queue.memcpy(host_mem, device_mem, N * sizeof(int)).wait();
}
```

```

printf("\nMalloc Host & Device Results:\n");
for (int i = 0; i < N; i++) {
    printf("%d, ", host_mem[i]);
}
printf("\n");

free(host_mem, my_gpu_queue);
free(device_mem, my_gpu_queue);

auto end = std::chrono::high_resolution_clock::now();
std::cout << "Malloc Host & Device Execution Time: "
    << std::chrono::duration<double, std::milli>(end -
start).count() << " ms\n";
}

void test_malloc_shared(queue& my_gpu_queue) {
    auto start = std::chrono::high_resolution_clock::now();

    int* shared_mem = malloc_shared<int>(N, my_gpu_queue);

    // Init data
    for (int i = 0; i < N; i++) {
        shared_mem[i] = i;
    }

    // Submit task to the queue
    my_gpu_queue.submit([&](handler& h) {
        h.parallel_for(range{N}, [=](id<1> item) {
            shared_mem[item] *= 2;
        });
    });

    // Wait the computation done
    my_gpu_queue.wait();

    printf("\nMalloc Shared Results:\n");

```



```

    for (int i = 0; i < N; i++) {
        printf("%d, ", shared_mem[i]);
    }
    printf("\n");

    free(shared_mem, my_gpu_queue);

    auto end = std::chrono::high_resolution_clock::now();
    std::cout << "Malloc Shared Execution Time: "
                << std::chrono::duration<double, std::milli>(end -
start).count() << " ms\n";
}

int main() {
    queue my_gpu_queue(sycl::gpu_selector_v);

    std::cout << "Selected GPU device: "
                <<
my_gpu_queue.get_device().get_info<info::device::name>() << "\n";

    test_malloc_host_device(my_gpu_queue);
    test_malloc_shared(my_gpu_queue);

    return 0;
}

```

可以看到结果确实共享内存更高效

```

D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda test_shared_mem.cpp -o test_shared_mem.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>test_shared_mem.exe
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER

Malloc Host & Device Results:
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Malloc Host & Device Execution Time: 135.904 ms

Malloc Shared Results:
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
Malloc Shared Execution Time: 42.7996 ms

D:\Code\GitCode\oneAPI_course\code>%

```

任务3

创建一个新文件（vector_add.cpp），使用ND_range实现一个向量加法程序

nd_range

首先我们查看什么是nd_range

sycl::nd_range

```
template <int Dimensions = 1>
class nd_range;
```

Template parameters

Dimensions

Number of dimensions in the `sycl::nd_range`.

The `sycl::nd_range` class defines the iteration domain of both the work-groups and the overall dispatch.

A `sycl::nd_range` comprises two `sycl::range` parameters: the whole range over which the kernel is to be executed, and the range of each work group.

The `sycl::nd_range` class template provides the [Common By-value Semantics](#).

据此我们可以得知，通过两个 `sycl::range` 参数，`sycl::nd_range` 可以表示如何将整个计算任务分解成多个工作组，以及每个工作组内部的任务如何分配和调度。

代码实现

接下来我们来开始完成代码。

我们选用的方案是，分配host和device内存，拷贝数据后进行运算，最后将结果拷贝回来并释放内存。这里我们按序初始化了向量用于vector_add的运算。

```
#include <sycl/sycl.hpp>
#include <iostream>
using namespace sycl;

constexpr int N = 1024; // 向量的大小
```

```
int main() {
    // 创建队列, 选择 GPU 设备
    queue my_gpu_queue(sycl::gpu_selector_v);

    std::cout << "Selected GPU device: " <<
        my_gpu_queue.get_device().get_info<info::device::name>() <<
        "\n";

    // 在主机上创建并初始化向量
    int *A = malloc_host<int>(N, my_gpu_queue);
    int *B = malloc_host<int>(N, my_gpu_queue);
    int *C = malloc_host<int>(N, my_gpu_queue);

    for (int i = 0; i < N; i++) {
        A[i] = i;    // 向量A的元素初始化为索引值
        B[i] = 2*i;  // 向量B的元素初始化为2倍的索引值
    }

    // 在设备上分配内存
    int *d_A = malloc_device<int>(N, my_gpu_queue);
    int *d_B = malloc_device<int>(N, my_gpu_queue);
    int *d_C = malloc_device<int>(N, my_gpu_queue);

    // 将数据从主机传输到设备
    my_gpu_queue.memcpy(d_A, A, N * sizeof(int)).wait();
    my_gpu_queue.memcpy(d_B, B, N * sizeof(int)).wait();

    // 定义工作组大小和全局工作大小
    size_t local_size = 256;    // 每个工作组中的元素数
    size_t global_size = N;     // 全局工作项数量

    // 使用ND_range提交向量加法计算
    my_gpu_queue.submit([&](handler& h) {
        // 设置设备内存的访问权限
        h.parallel_for<nd_range<1>>(nd_range<1>(global_size,
            local_size), [=](nd_item<1> item) {
```

```

        // 获取当前工作项的索引
        size_t idx = item.get_global_id(0);

        if (idx < N) { // 确保索引在合法范围内
            d_C[idx] = d_A[idx] + d_B[idx]; // 向量加法
        }
    });
});

// 等待GPU计算完成
my_gpu_queue.wait();

// 将结果从设备拷贝回主机
my_gpu_queue.memcpy(C, d_C, N * sizeof(int)).wait();

// 输出结果
std::cout << "First 10 elements of the result vector C:\n";
for (int i = 0; i < 10; i++) {
    std::cout << C[i] << ", ";
}
std::cout << "\n";

// 释放内存
free(A, my_gpu_queue);
free(B, my_gpu_queue);
free(C, my_gpu_queue);
free(d_A, my_gpu_queue);
free(d_B, my_gpu_queue);
free(d_C, my_gpu_queue);

return 0;
}

```

然后我们编译并运行

```
D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda vector_add.cpp -o vector_add.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>vector_add.exe
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_level_zero.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB52D10000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_opencl.dll)
<LOADER>[INFO]: loaded adapter 0x00007FFB46AC0000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_cuda.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_hip.dll)
<LOADER>[INFO]: loaded adapter 0x0000000000000000 (C:\Program Files (x86)\Intel\oneAPI\compiler\latest\bin\ur_adapter_native_cpu.dll)
SYCL_UR_TRACE: Requested device_type: info::device_type::gpu
SYCL_UR_TRACE: Selected device: -> final score = 2000
SYCL_UR_TRACE: platform: NVIDIA CUDA BACKEND
SYCL_UR_TRACE: device: NVIDIA GeForce RTX 4070 Ti SUPER
Selected GPU device: NVIDIA GeForce RTX 4070 Ti SUPER
First 10 elements of the result vector C:
0, 3, 6, 9, 12, 15, 18, 21, 24, 27,
<LOADER>[INFO]: unloaded adapter 0x00007FFB52D10000
<LOADER>[INFO]: unloaded adapter 0x00007FFB46AC0000
<LOADER>[INFO]: ---> urLoaderTearDown() -> UR_RESULT_SUCCESS

D:\Code\GitCode\oneAPI_course\code>
```

完成vector_add。

Part 2 课后作业

作业1

改写gemm_basic代码26、27行，利用work group和local work item的坐标来计算global坐标

代码改写

我们修改代码，将原本的直接调用 `get_global_id()` 修改为计算 `group_id*block_size+local_id` 得到即可

```

// 获取当前局部工作项和工作组的坐标
int local_row = index.get_local_id(0);
int local_col = index.get_local_id(1);
int group_row = index.get_group(0);
int group_col = index.get_group(1);

// 计算全局坐标
int row = group_row * block_size + local_row;
int col = group_col * block_size + local_col;

```

测试代码

修改后编译发现，存在报错

```

Intel(r) oneAPI
C:\Users\Harry>D
'D' 不是内部或外部命令，也不是可运行的程序
或批处理文件。

C:\Users\Harry>D:

D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda gemm_basic_global.cpp -o gemm_basic_global.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code\gemm_basic_global.cpp(78,12): error: call to 'fabs' is ambiguous
78 |         if( fabs(cpu_res[i] - gpu_res[i]) > 1e-3) {
D:\Program Files (x86)\Visual Studio IDE\VC\Tools\MSVC\14.41.34120\include\cmath(117,40): note: candidate function
117 | _NODISCARD _Check_return_ inline float fabs(_In_ float _Xx) noexcept /* strengthened */ {
C:\Program Files (x86)\Intel\oneAPI\compiler\2025.0\include\sycl\detail\builtins\math_functions.inc(139,23): note:
candidate function
139 | BUILTIN_GENF(ONE_ARG, fabs)
D:\Code\GitCode\oneAPI_course\code\gemm_basic_global.cpp(158,19): error: use of undeclared identifier 'cl'
158 |     auto propList = cl::sycl::property_list {cl::sycl::property::queue::enable_profiling()};
D:\Code\GitCode\oneAPI_course\code\gemm_basic_global.cpp(160,21): error: use of undeclared identifier 'cl'
160 |     queue my_gpu_queue( cl::sycl::gpu_selector_v, propList);
3 errors generated.

D:\Code\GitCode\oneAPI_course\code>

```

第一处报错 fabs 存在歧义，我们修改为 std::fabs 。

第二出为 cl::sycl 已经被改为 sycl ，修改即可。

编译并运行

```
D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda gemm_basic_global.cpp -o gemm_basic_global.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>gemm_basic_global.exe
Problem size: c(1024,1024) = a(1024,1024) * b(1024,1024)

Performance Flops = 2147483648.000000,
GPU Computation Time = 11.613884 (ms);
CPU Computaiton Time = 585.660242 (ms);

D:\Code\GitCode\oneAPI_course\code>
```

可以看到我们成功完成作业1

作业2

修改程序输入数据的大小，设定为 $M=N=K=2000$ ，修改程序，并使其通过正确性测试。

代码改写

只需改写输入的mnk即可

测试代码

```
D:\Code\GitCode\oneAPI_course\code>icx-cl -fsycl -fsycl-targets=nvptx64-nvidia-cuda gemm_basic_global.cpp -o gemm_basic_global.exe
Intel(R) oneAPI DPC++/C++ Compiler for applications running on Intel(R) 64, Version 2025.0.0 Build 20241008
Copyright (C) 1985-2024 Intel Corporation. All rights reserved.

D:\Code\GitCode\oneAPI_course\code>gemm_basic_global.exe
Problem size: c(2000,2000) = a(2000,2000) * b(2000,2000)

Performance Flops = 16000000000.000000,
GPU Computation Time = 84.174252 (ms);
CPU Computaiton Time = 2754.480908 (ms);

D:\Code\GitCode\oneAPI_course\code>
```

成功完成，我们可以发现，GPU在大规模的矩阵运算上有着明显的优势。