

# 并行计算实验一报告

## ——OpenMP 及 CUDA 实验环境的搭建

PB20111701 叶升宇

PB20111689 蓝俊玮

### 说明

以下为 PB20111701 叶升宇的实验环境。

## OpenmMP 环境搭建

### 下载 CLion

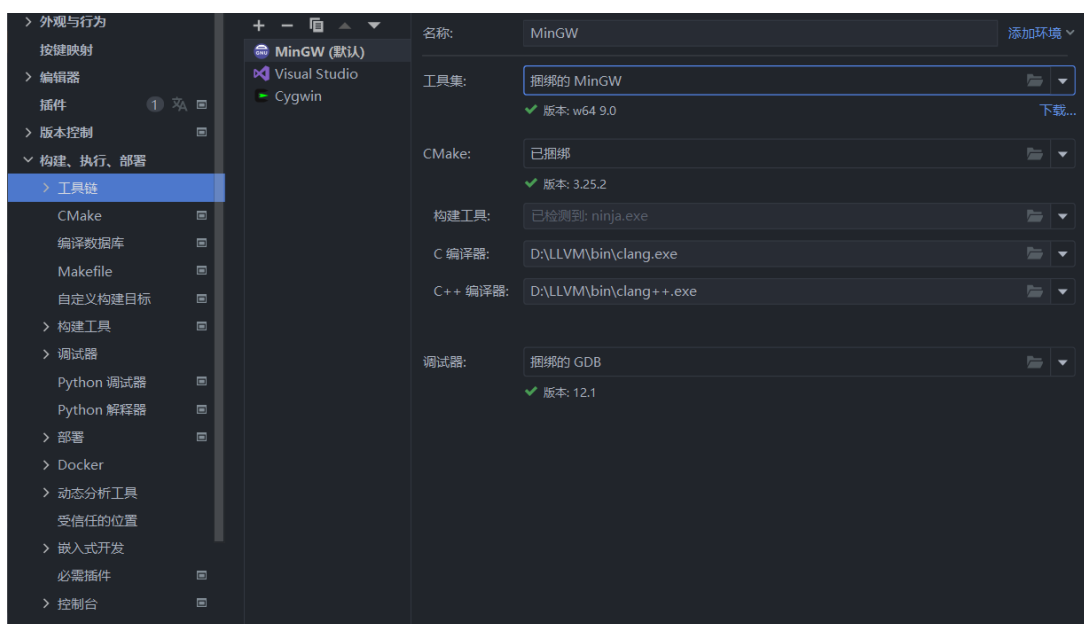
本学期并行计算实验使用的 IDE 为 CLion，下载链接如下 [CLion](#)。

### 下载 MinGW64

使用 MinGW64 作为工具集，下载链接如下 [MinGW64](#)。

### 配置 LLVM + clang

前往 [LLVM](#) 官网下载，c 编译器使用 LLVM clang，c++ 编译器使用 LLVM clang++，构建工具使用 Ninja(而非通常的 Make)。整体工具链配置完毕后如下：



### 配置 CMake

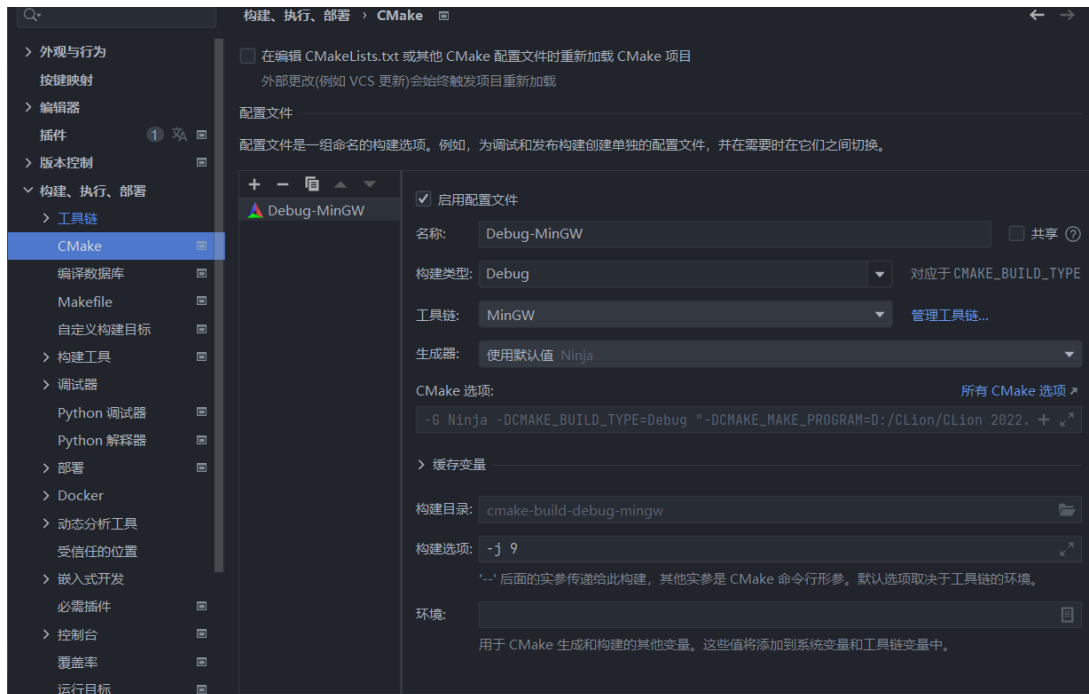
编写 CMakeLists.txt 如下：

- 使用 c++ 14 标准；
- 用 -fopenmp 选项开启 OpenMP 支持

```
cmake_minimum_required(VERSION 3.10)
project(sort)
set(CMAKE_CXX_STANDARD 14)
find_package(OpenMP REQUIRED)
set(SOURCE_FILES main.cpp)
```

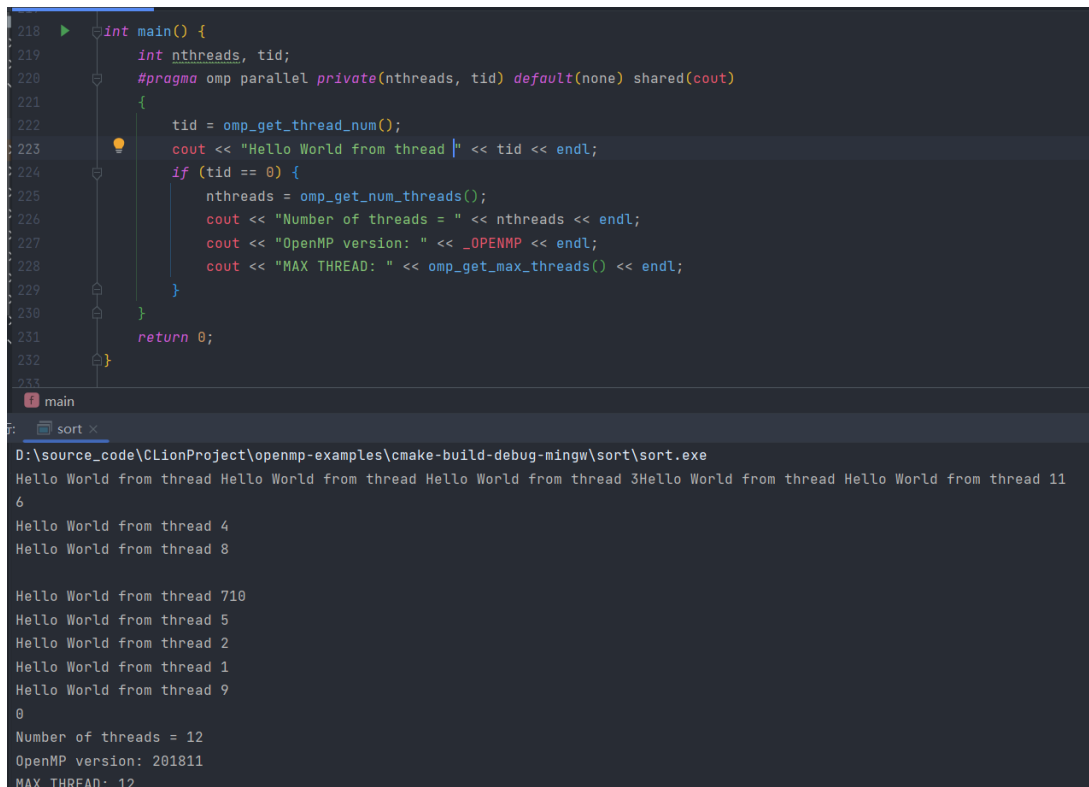
```
add_executable(sort ${SOURCE_FILES})
set(CMAKE_C_COMPILER "clang")
set(CMAKE_CXX_COMPILER "clang++")
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -fopenmp")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fopenmp")
```

Cmake 相关配置如下：



## 验证配置成功

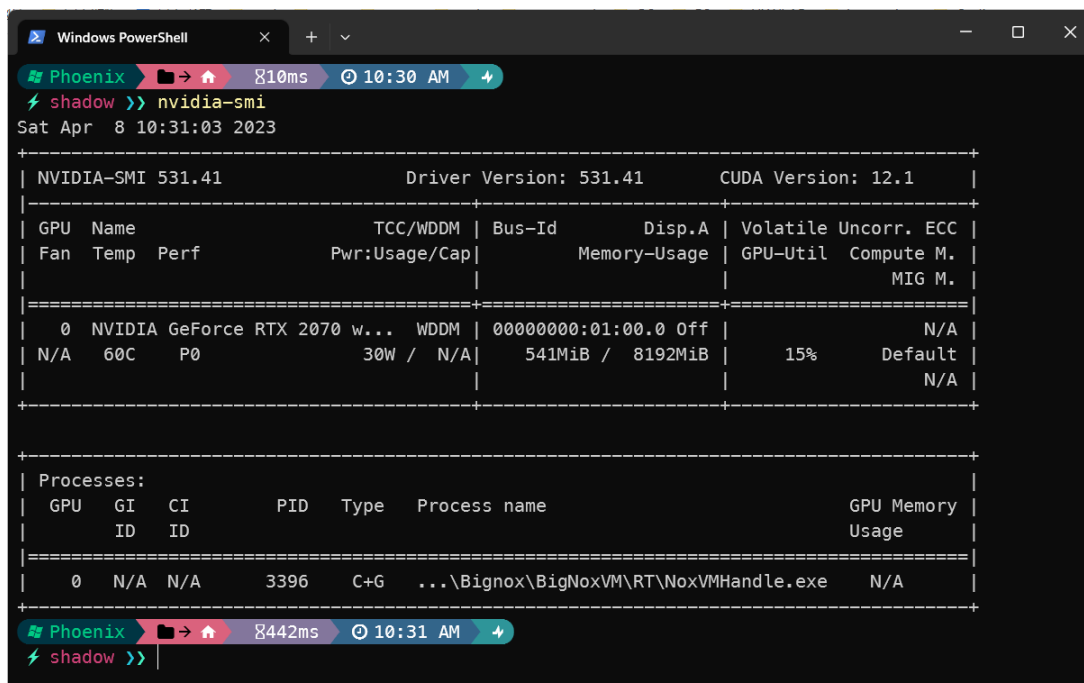
打印简单的 OpenMP 相关参数，来验证配置成功：



## CUDA 环境搭建

### NVIDIA 驱动的安装

依然使用 CLion 作为 IDE，配 OpenMP 时已经完成，不再赘述。下面检查驱动是否安装：在 Windows Terminal 中输入 nvidia-smi：



```
Windows PowerShell
Phoenix 810ms 10:30 AM
shadow >> nvidia-smi
Sat Apr 8 10:31:03 2023

+-----+
| NVIDIA-SMI 531.41                  Driver Version: 531.41      CUDA Version: 12.1     |
+-----+-----+
| GPU Name                               TCC/WDDM | Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf              Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                                           | MIG M.         |
+-----+-----+
|  0  NVIDIA GeForce RTX 2070 w...  WDDM  00000000:01:00.0 Off |                  N/A | |
| N/A   60C   P0              30W /  N/A |  541MiB /  8192MiB |      15%    Default  |
|                                           |                  N/A |
+-----+-----+

+-----+
| Processes:                         |
| GPU   GI    CI          PID    Type   Process name                      GPU Memory |
| ID   ID   ID              |          |                  | Usage     |
+-----+-----+
|  0   N/A  N/A         3396    C+G    ...\.BigNox\BigNoxVM\RT\NoxVMHandle.exe  N/A       |
+-----+

Phoenix 8442ms 10:31 AM
shadow >>
```

#### 说明

这里我是已经配好了 CUDA 环境，所以驱动和 CUDA 版本都有显示。

没有安装驱动的情况下，需要到 [英伟达官网](#) 选取对应显卡的版本下载安装。

### CUDA 安装

进入 [CUDA Toolkit 官网](#) 进行下载并安装，我选择的是 12.1 版本，同时在系统变量中加入 CUDA 路径：

CUDA_PATH	C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1
CUDA_PATH_V12_1	C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1

以及在 PATH 中加入 CUDA 的 BIN 路径：

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\bin
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\libnvvp

### 验证安装成功

下面通过运行 deviceQuery.exe、bandwidthTest.exe，result = PASS 说明安装成功：

```
Windows PowerShell
Phoenix C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\extras\demo_suite 80ms 10:55 AM
shadow >> .\deviceQuery.exe
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\extras\demo_suite\deviceQuery.exe Starting...

CUDA Device Query (Runtime API) version (CUDA static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce RTX 2070 with Max-Q Design"
  CUDA Driver Version / Runtime Version      12.1 / 12.1
  CUDA Capability Major/Minor version number: 7.5
  Total amount of global memory:             8192 MBytes (8589606912 bytes)
  (36) Multiprocessors, ( 64) CUDA Cores/MP: 2304 CUDA Cores
  GPU Max Clock rate:                       1125 MHz (1.13 GHz)
  Memory Clock rate:                        5501 Mhz
  Memory Bus Width:                         256-bit
  L2 Cache Size:                           4194304 bytes
  Maximum Texture Dimension Size (x,y,z)    1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(32768, 32768), 2048 layers
  Total amount of constant memory:           32 MBytes
  Total amount of shared memory per block:   96 KBytes
  Total number of registers available per block: 65536
  Warp size:                               32
  Maximum number of threads per multiprocessor: 1024
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                     32 MBytes
  Texture alignment:                        32 MBytes
  Concurrent copy and kernel execution:      Yes with 2 copy engine(s)
  Run time limit on kernels:                 Yes
  Integrated GPU sharing Host Memory:         No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                    Disabled
```

```
Windows PowerShell
Phoenix C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\extras\demo_suite 8356ms 10:55 AM
shadow >> .\bandwidthTest.exe
[CUDA Bandwidth Test] - Starting...
Running on...

Device 0: NVIDIA GeForce RTX 2070 with Max-Q Design
Quick Mode

Host to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  12418.1

Device to Host Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  11766.2

Device to Device Bandwidth, 1 Device(s)
PINNED Memory Transfers
  Transfer Size (Bytes)      Bandwidth(MB/s)
  33554432                  295113.6

Result = PASS
```

## 配置 CMake

在 CLion 中，选择项目为 CUDA 可执行文件，然后编写 CMakeLists.txt 如下：

```
cmake_minimum_required(VERSION 3.10)
project(cuda_examples CUDA)
set(CMAKE_CUDA_STANDARD 14)
```

```
find_package(CUDA REQUIRED)
add_executable(cuda_examples main.cu)
```

这里以老师 PPT 上的乘积求和程序为例，输出结果如下，也验证了 CUDA 配置成功：

```
57     cudaMemcpy( dst: d_a, src: a, count: size, kind: cudaMemcpyHostToDevice);
58     cudaMemcpy( dst: d_b, src: b, count: size, kind: cudaMemcpyHostToDevice);
59
60     Dot <<< gridDim: 1, blockDim: N >>>( a: d_a, b: d_b, c: d_c); // single block multi threads
61     cudaMemcpy( dst: c, src: d_c, count: sizeof(int), kind: cudaMemcpyDeviceToHost);
62
63     int sumHost = 0;
64     for (int i = 0; i < N; i++) {
65         sumHost += a[i] * b[i];
66     }
67     printf( Format: "sum Calculated on Host = %d\n", sumHost);
68     printf( Format: "Device to Host: a * b = %d\n", *c);
69
70     free( Block: a);
71     free( Block: b);
```

main

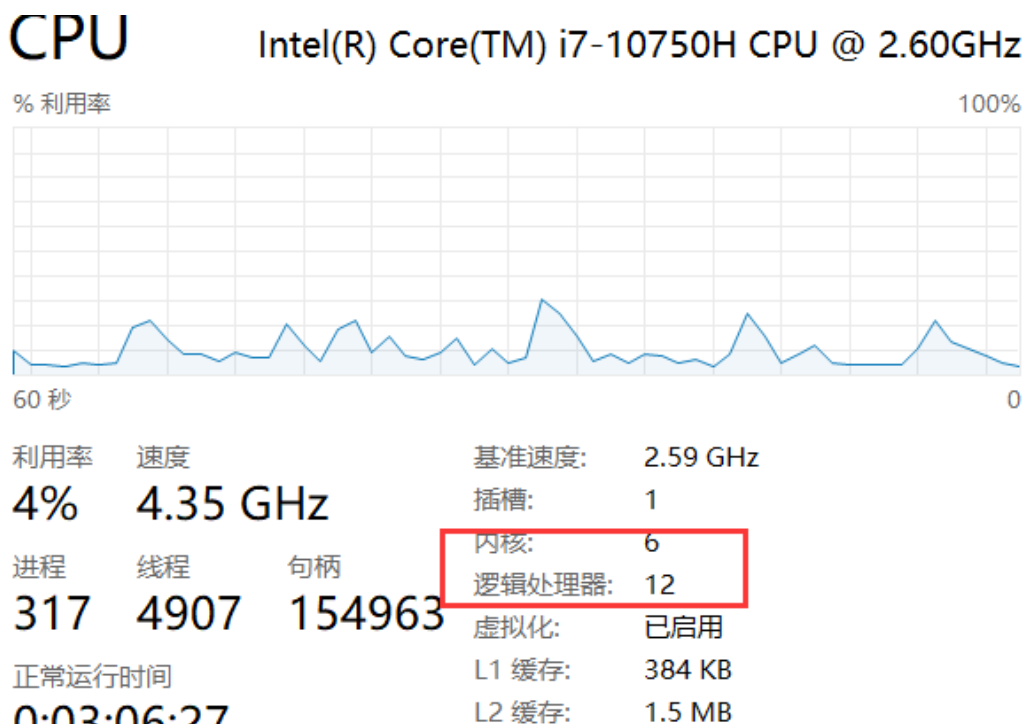
运行: cuda\_examples x

Array a[N]:  
1 7 4 0 9 4 8 8 2 4  
Array b[N]:  
5 5 1 7 1 1 5 2 7 6  
sum Calculated on device 151  
sum Calculated on Host = 151  
Device to Host: a \* b = 151  
进程已结束,退出代码0

## 其它设备参数

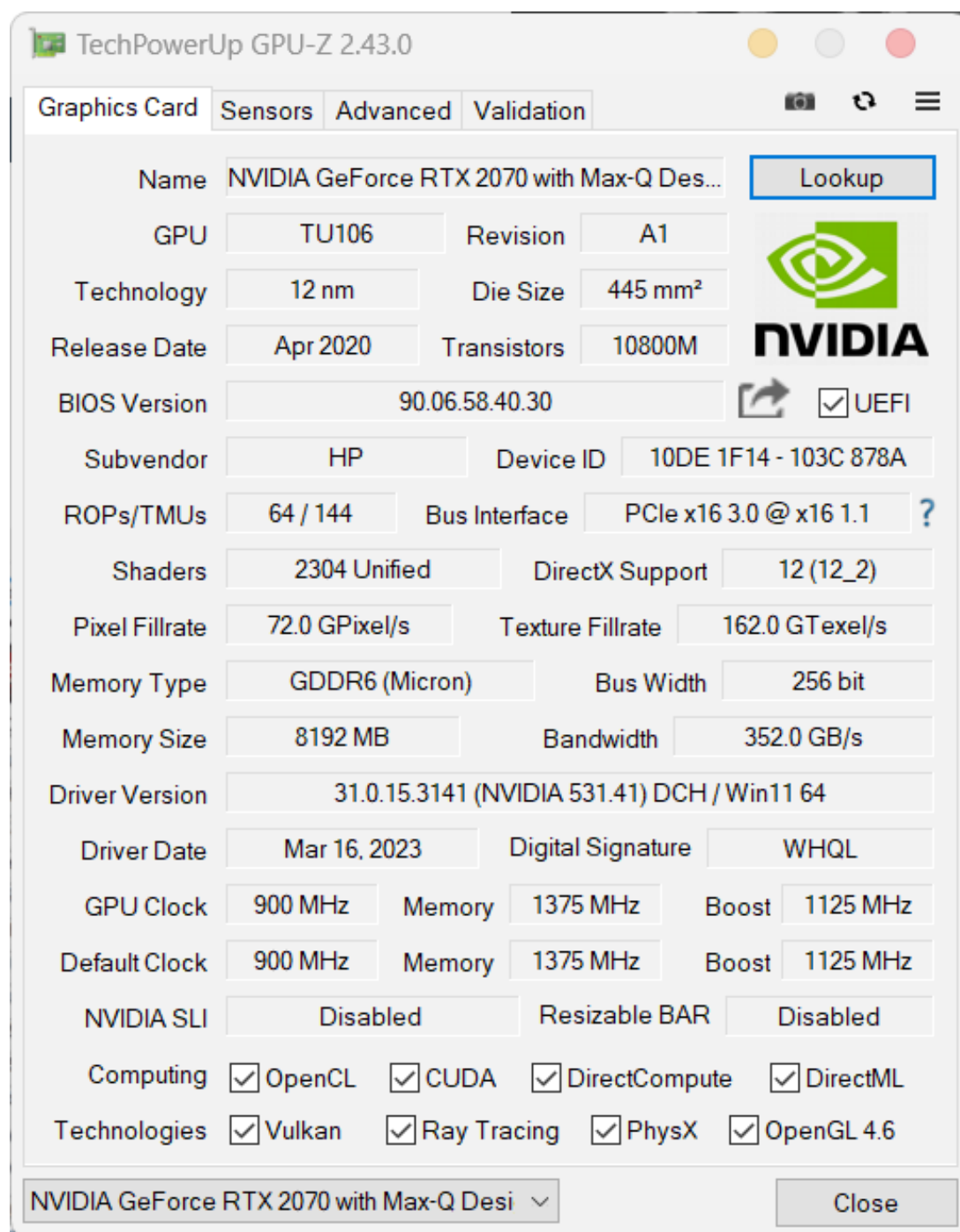
### CPU 参数

通过任务管理器来查看， 我的电脑是 6 核 12 线程的：



## GPU 参数

利用课程群中提供的 GPU-Z 查看 GPU 参数如下：



### 说明

实际上，在先前配置 CUDA 时候通过 deviceQuery.exe 也可以来获取相关参数。

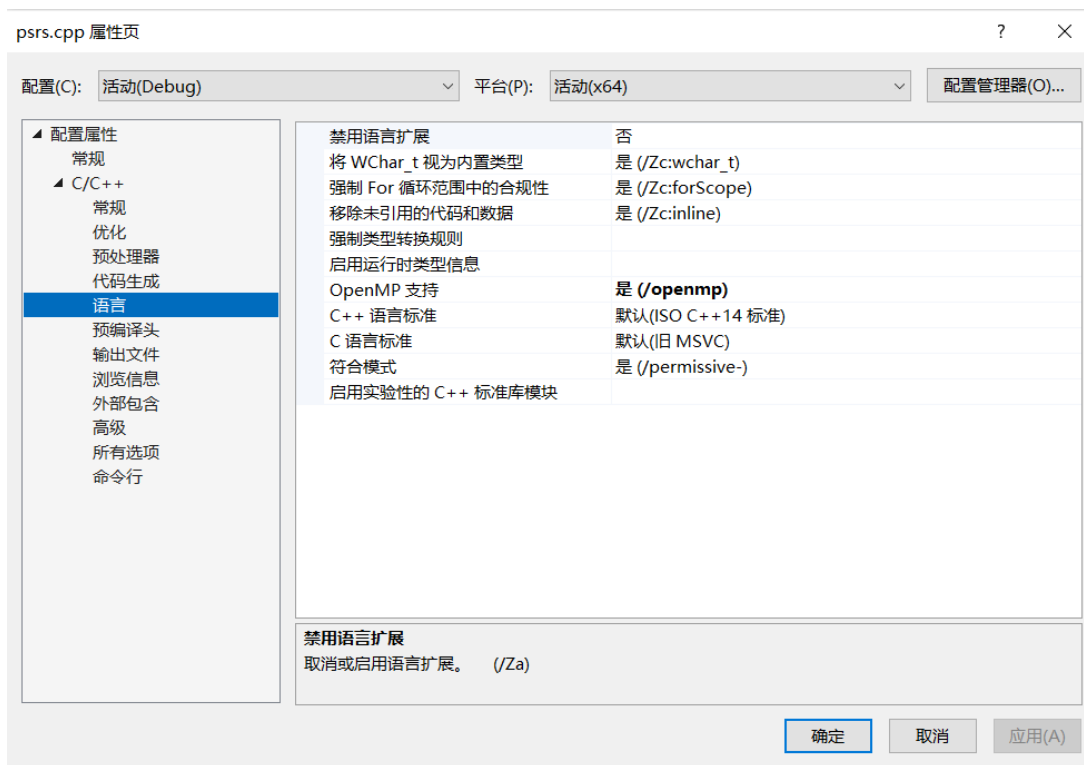
### 说明

以下为 PB20111689 蓝俊玮的实验环境。

## OpenmMP 环境搭建

通过 Visual Studio 2022 进行环境搭建，在 Visual Studio 2022 可以通过配置 OpenMP Support 开启 OpenMP 2.0 的拓展支持。

可以在 Visual Studio 2022 内右键点击创建的项目，或者上方任务栏中的项目，选择属性。在打开属性页之后，点击 C/C++ -> 语言，更改如下配置如图所示，然后点应用，确定。



测试效果如下：

```
Microsoft Visual Studio 调试控制台
Serial Pi: 3.14159
Serial Running Time: 0.0008724 seconds

Parallel Pi: 3.14166
Parallel Running Time: 0.0060322 seconds

Parallel Pi: 3.14159
Parallel Running Time: 0.0008547 seconds

Parallel Pi: 3.14159
Parallel Running Time: 7.74e-05 seconds

Parallel Pi: 3.14159
Parallel Running Time: 7.95e-05 seconds

D:\vs2022_code\parallel-computing-labs\x64\Debug\parallel-computing-labs.exe (进程 8668) 已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。 . . .
```

## MPI 环境搭建

通过 Visual Studio 2022 进行环境搭建，在 [Microsoft MPI v10.0](#) 下载 MPI，将 mspmissetup.exe 和 mspmisdsk.msi 都安装下来。

接下来在 Visual Studio 2022 的项目内，打开项目属性：

1. VC++目录 -> 包含目录 -> 编辑：

添加 D:\mpisdk\Include (这个与个人安装路径有关)

2. VC++目录 -> 库目录 -> 编辑:

添加 D:\mpisdk\Lib\x64 (x64 与 Visual Studio 2022 的平台活动一致)

3. C/C++-> 预处理器 -> 预处理器定义:

添加: MPICH\_SKIP\_MPICXX

4. C/C++ -> 代码生成 -> 运行库:

选择: 多线程调试 (/MTd)

5. 链接器 -> 输入 -> 附加依赖项:

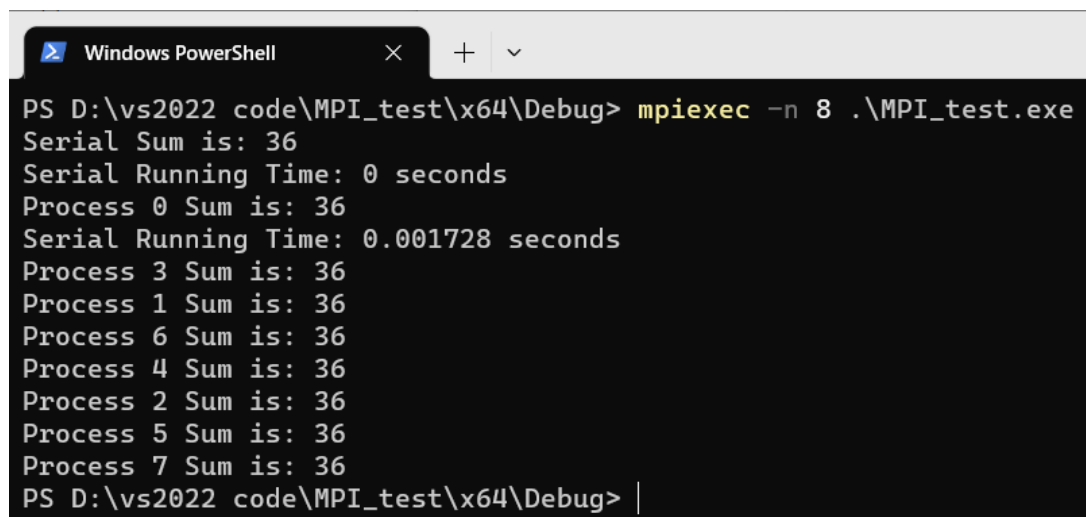
添加: msmapi.lib, msmtpifec.lib, msmtpifmc.lib (即 Lib/X64 下的 lib 文件)

编译和运行时通过 Visual Studio 2022 的菜单栏中的“生成”的“生成解决方案”可以生成可执行文件, 然后使用 Windows Terminal, 在生成的可执行程序的目录下运行它, 运行命令为: mpiexec -n 6 test.exe(其中 -n 后面跟着的是进程数量)

### 说明

参考链接: [VS2022配置MPI环境](#)

测试效果如下:



```
Windows PowerShell
PS D:\vs2022 code\MPI_test\x64\Debug> mpiexec -n 8 .\MPI_test.exe
Serial Sum is: 36
Serial Running Time: 0 seconds
Process 0 Sum is: 36
Serial Running Time: 0.001728 seconds
Process 3 Sum is: 36
Process 1 Sum is: 36
Process 6 Sum is: 36
Process 4 Sum is: 36
Process 2 Sum is: 36
Process 5 Sum is: 36
Process 7 Sum is: 36
PS D:\vs2022 code\MPI_test\x64\Debug> |
```

## OpenCV 4.7.0 安装

使用 GitCode.net 的 OpenCV 镜像安装 OpenCV

```
git clone https://github.com/opencv/opencv
```

```
cmake -B opencv-build -D OPENCV_GENERATE_PKGCONFIG=ON opencv
```

### 注意

-D OPENCV\_GENERATE\_PKGCONFIG=ON 这个选项十分重要, 控制是否生成 pkg\_config, opencv4 中如果不加这个命令, 就不会生成 pkgconfig, 就会导致安装后找不到 opencv4 文件。

cmake 结束后执行 make -j8 指令, 然后执行 make install, 由于我使用的是服务器, 没有足够的权限, 所以修改 cmake\_install.cmake 中的 CMAKE\_INSTALL\_PREFIX 为 /amax/home/junwei/opencv\_usr\_local (原来为 /usr/local)



安装后，添加环境变量，在 `/amax/home/junwei/.bashrc` 中添加如下：

```
export PKG_CONFIG_PATH=/amax/home/junwei/opencv_usr_local/lib/pkgconfig
export LD_LIBRARY_PATH=/amax/home/junwei/opencv_usr_local/lib
```

添加完成后：

```
source ~/.bashrc
```

验证是否安装成功，查看opencv的版本：

```
pkg-config --modversion opencv4
```

当使用 opencv 时，可以采用下面的方法编译运行：

```
nvcc test.cu -o test.out -I/amax/home/junwei/opencv_usr_local/include/opencv4 -L/amax/home/junwei/opencv_usr_local/lib `pkg-config --cflags --libs opencv4`
```

其中 `-I` 表示了添加额外的搜索库的路径，`-L` 表示了加载了额外的库

#### 说明

参考链接：[Linux下安装opencv\(root身份和非root普通用户安装\)](#)

参考链接：[下载不再卡顿，OpenCV 中国镜像仓库正式启用](#)

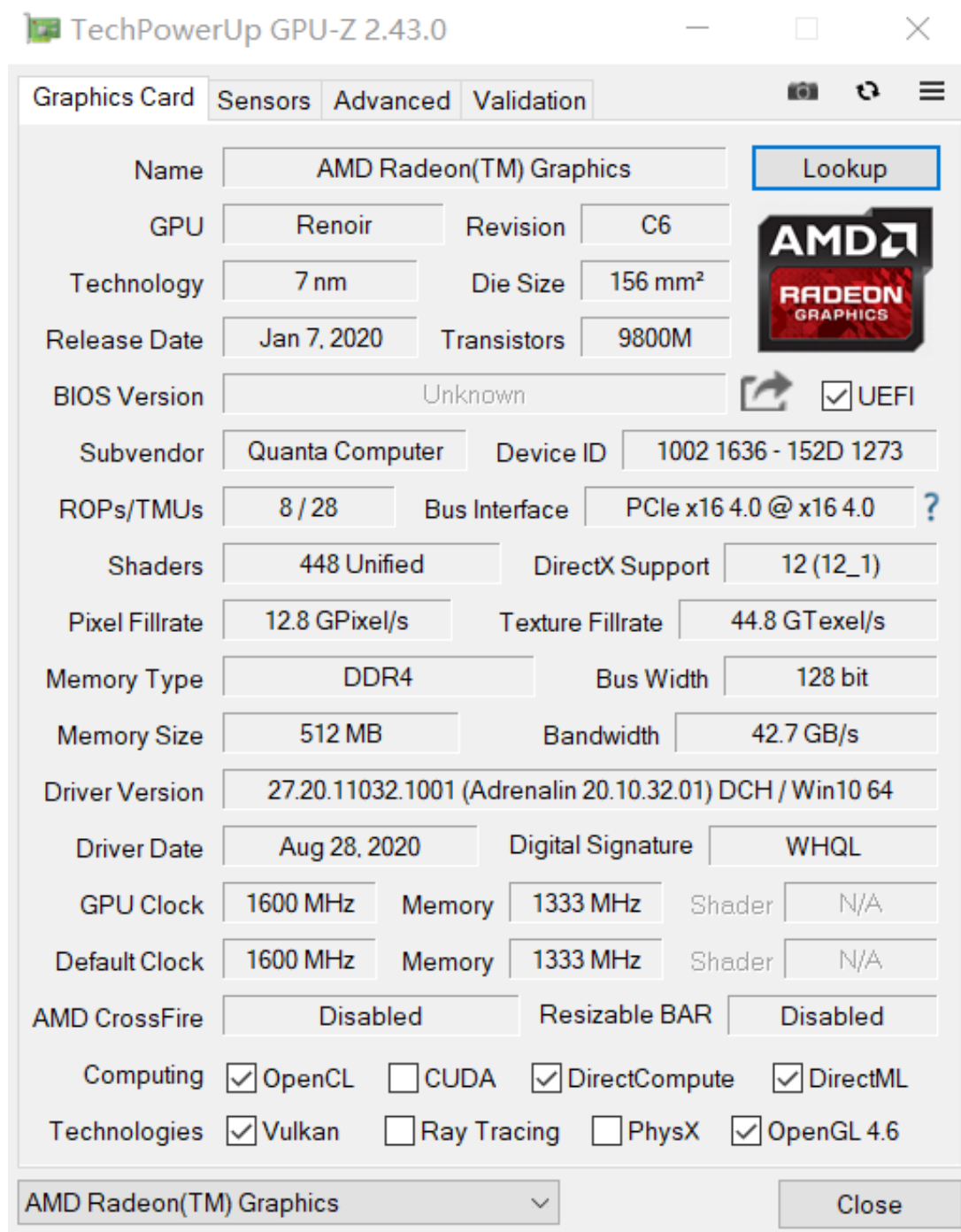
参考链接：[GCC中 -I、-L、-l 选项的作用](#)

## Cuda 环境

由于我使用的是实验室的服务器，因此 Cuda 的驱动都是已经安装好的，这里就不多说。

## 设备参数

我的个人电脑 CPU(AMD Ryzen 7 4800H with Radeon Graphics 2.90 GHz) 是 8 核 16 线程的，下面展示我个人电脑的 CPU 设备性能（没有 GPU）



下面展示服务器 GPU 设备的性能：

```
(base) junwei@admin:~$ nvidia-smi -L
GPU 0: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-6f80ca5b-9f54-8ff3-255e-a90e4408a9f2)
GPU 1: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-0ce8fe06-281e-d601-f7dd-f0a29b067a61)
GPU 2: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-83bfe182-5cfc-1d22-1348-276c30e11fa0)
GPU 3: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-1a59df34-bbee-e22b-13eb-adcb4d453e7e)
GPU 4: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-20f69bc9-ea5d-0848-308f-13d85d555e00)
GPU 5: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-023f01c2-aab0-17fe-d5f4-a4cf43c018d2)
GPU 6: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-c109c908-2f37-a223-fb94-61cb8217e51a)
GPU 7: NVIDIA GeForce RTX 2080 Ti (UUID: GPU-c7cded5f-e2d4-34ec-e196-35949c18997b)
(base) junwei@admin:~$
```

```
(base) junwei@admin:~$ ./test.out
Device Name: NVIDIA GeForce RTX 2080 Ti
totalGlobalMem: 11020 MBytes---11554848768 Bytes
sharedMemPerBlock: 49152
regsPerBlock: 65536
warpSize: 32
memPitch: 2147483647
maxThreadsPerBlock: 1024
maxThreadsDim[0-2]: 1024 1024 64
maxGridSize[0-2]: 2147483647 65535 65535
totalConstMem: 65536
major.minor: 7.5
clockRate: 1545000
textureAlignment: 512deviceOverlap: 1
multiProcessorCount: 68
```