写在前面:材料来源

https://github.com/KhronosGroup/OpenCL-SDK/releases

可以从此处获取关键库文件（.h & .lib） (演示时使用了最新发布的 x64 版本)

注意，从此处获取的 OpenCL.dll 未必可用 (大小仅 55+KB)

显卡厂商发布的驱动往往也随带 OpenCL.dll，

对于 Nvidia RTX2060，其大小约 1.4MB，确实可用。

右者是在笔者本机 C:\Windows\System32 下找到的，也是演示时真正被使用的。

https://github.com/niXman/mingw-builds-binaries/releases

该渠道主要发布构建完毕的 MinGW-w64 组件。

笔者自用的是 Release 12.2.0-rt_v10-rev1 版本的 MinGW-w64

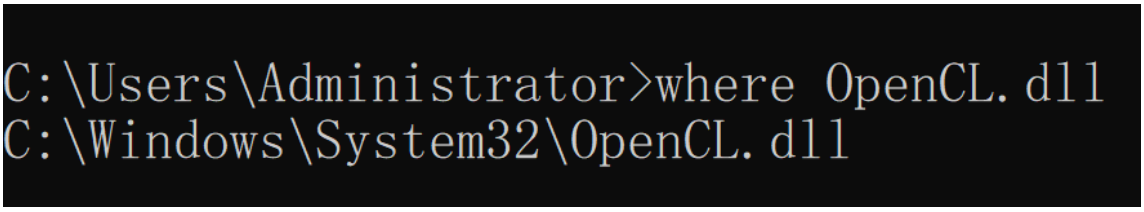
真.使用方法

这里的"mingw64"由前述材料的组装得到，在 Intel i7 10875H + Nvidia RTX 2060 配置的 Win10 OS 机器中测试通过。如果你使用 Apple Mac 或者安装了 Linux 发行版操作系统，请自寻出路。

0. 为你的 GPU 安装厂商发布的稳定可靠的正版驱动， (一般而言，这是装机自带的)

   至少要能在 C:\Windows\System32 下找到可用的 OpenCL.dll。

   可以在控制台用 where 命令行查找该文件。

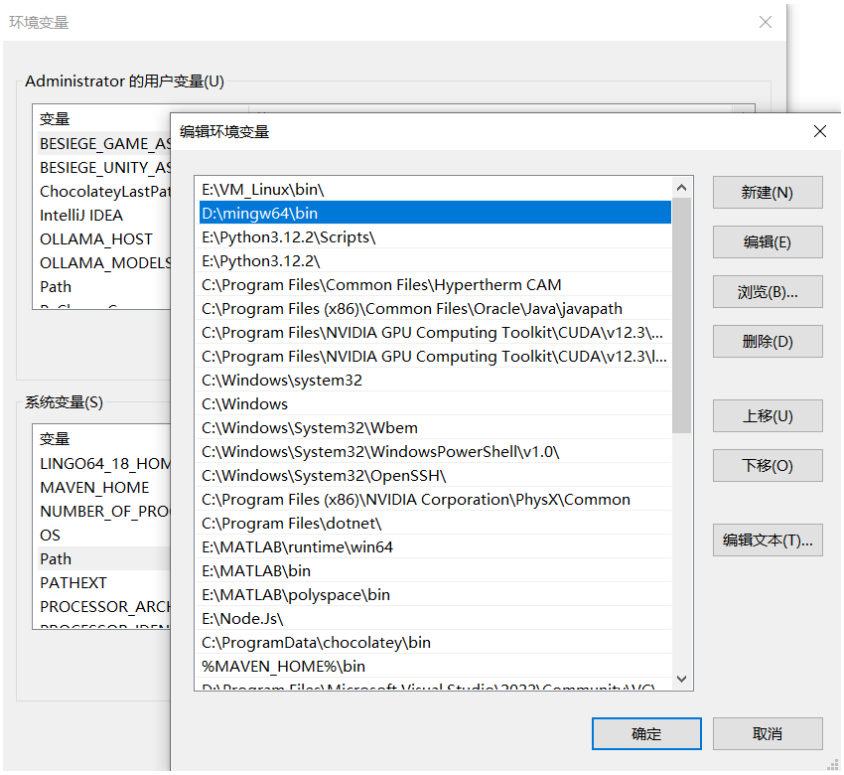


↑ 这样就行 ↑

1. 以你喜欢的方式解压下载所得的 mingw64.zip。

2. 添加系统环境变量，指向你解压所得的 (内层) mingw64 目录下的 bin 目录。

   (就像配置 python 解释器一样)



→

→

"mingw64\bin"之前的部分

根据你的实际路径作修改

如果系统环境很复杂，请注意 gcc 的版本是否为 12.2.0，自行增删调整环境变量

3. 双击 mingw64\OpenCL_Lab 目录下的 cmd.bat，打开一个控制台窗口。输入 clinfo 后回车，应当能查询到可用的 OpenCL Platform，否则可能需要返回第 0 步。

```
D:\mingw64\OpenCL_Lab>clinfo
Number of platforms                               1
  Platform Name                                   NVIDIA CUDA
  Platform Vendor                                 NVIDIA Corporation
  Platform Version                                OpenCL 3.0 CUDA 12.3.68
  Platform Profile                                FULL_PROFILE
  Platform Extensions                             cl_khr_global_int32_base_atomics cl_khr_global_
ions cl_nv_device_attribute_query cl_nv_pragma_unroll cl_nv_d3d10_sharing cl_khr_d3d10_sharing c
r_external_semaphore_win32 cl_khr_external_memory_win32
  Platform Extensions with Version                cl_khr_global_int32_base_atomics
                                                  cl_khr_global_int32_extended_atomics
                                                  cl_khr_local_int32_base_atomics
                                                  cl_khr_local_int32_extended_atomics
                                                  cl_khr_fp64
                                                  cl_khr_3d_image_writes
                                                  cl_khr_byte_addressable_store
                                                  cl_khr_icd
                                                  cl_khr_gl_sharing
                                                  cl_nv_compiler_options
                                                  cl_nv_device_attribute_query
                                                  cl_nv_pragma_unroll
                                                  cl_nv_d3d10_sharing
                                                  cl_khr_d3d10_sharing
                                                  cl_nv_d3d11_sharing
                                                  cl_nv_copy_opts
                                                  cl_nv_create_buffer
```

↑ 正常情况下，你会看到类似的输出 ↑

4. 如果一切顺利，继续输入 mingw32-make 后回车，让该工具根据 makefile 中的语句编译 test.cpp，如果编译成功，则继续输入 test 后回车，看到正常的算术运算结果表明 OpenCL 已经可用。

```
D:\mingw64\OpenCL_Lab>mingw32-make
g++ -I"..\include" -I"..\x86_64-w64-mingw32\include" test.cpp -o test -L"..\lib" -L"..\x86_64-w64-mingw32\lib" -lopencl
In file included from ..\include/CL/cl.h:20,
                 from test.cpp:15:
..\include/CL/cl_version.h:22:104: note: '#pragma message: cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaul
   22 |  #pragma message("cl_version.h: CL_TARGET_OPENCL_VERSION is not defined. Defaulting to 300 (OpenCL 3.0)")
      |                                                                                                        ^
```

↑ 正常的编译结果，应该不包含过多的 error 或者 warning 信息 ↑

test.cpp 源码所表达的关键逻辑是，读取 test.cl 中的源码至字符串变量，利用 OpenCL 扩展实现运行时编译，从而将（并没有实际意义的）计算任务提交至 GPU 完成。

test.cpp 中使用的函数 clCreateCommandQueueWithProperties 要求计算设备支持 OpenCL 2.0 标准，如果出现相关报错，让 DeepSeek 师傅帮你改用 clCreateCommandQueue 即可。

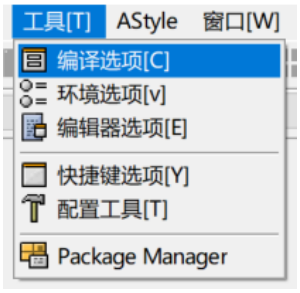
```
kernel execution cost: 0.450267s
host timer finds cost: 0.452s

Received From GPU
0.966332
0.967203
0.968073
0.968945
0.969817
0.970689
0.971562
0.972433
0.973307
0.974181
0.975053
0.975927
0.976802
0.977675
0.97855
0.979425
0.9803
0.981175
0.982051
0.982927
```

↑ 典型的正常结果，其中的耗时则和设备的具体情况有关 ↑

Build log 后紧跟空行是正常的，这代表 kernel 代码被顺利编译

5. 充分理解第 4 步的各个环节，你就可以尝试借助 IDE (比如 Visual Studio) 为自己减负了。

下面提供 Dev-C++ 5.11 的部分配置参考，成功后可以在 IDE 界面编译并运行，享受语法高亮、参数提示、变量名替换、单步调试 (gdb 仅对于 host 端代码可用) 等一切现代 IDE 应有的功能。



→

→

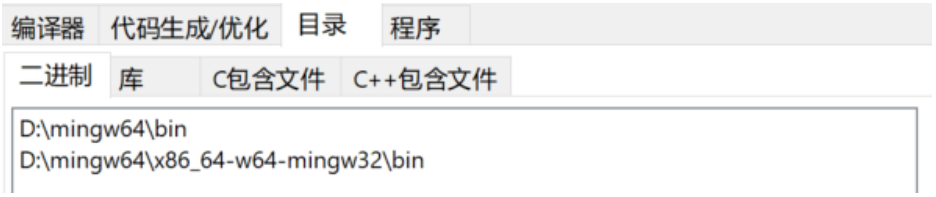
这里只有 –lopencl 是必需的

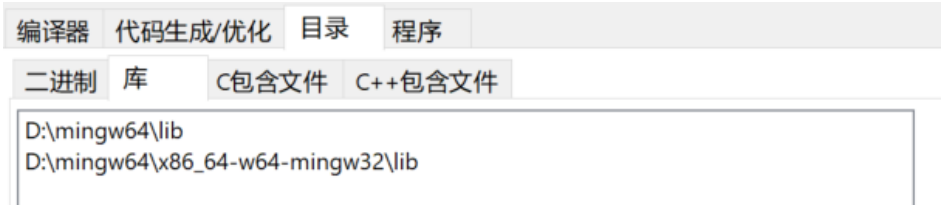
→

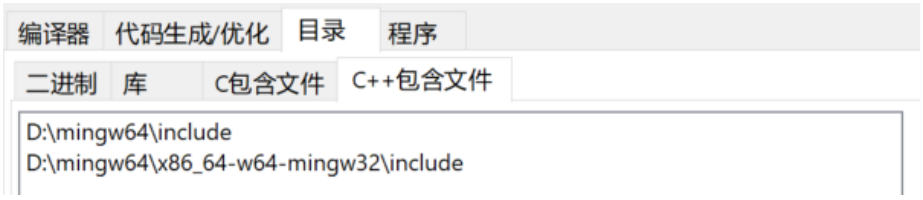
如果使用 gdb 进行单步调试

"产生调试信息"一项请选 Yes

其它均可为默认的 No







→

→

→

"程序"一栏，最好都选择 mingw64 目录下找到的版本。