**作业有两问：**

* 1. 查询你机器上GPU设备的参数

首先，新建.cu文件调用cudaGetDeviceCount()得到GPU设备的数量；

其次，调用cudaGetDeviceProperties()函数得到GPU设备的属性结构体解释关键属性的含义，至少包括设备名称、计算能力为多少、设备可用全局内存、每线程块最大线程数、设备可用全局内存容量、每线程块可用共享内存容量、每线程块可用寄存器数量、每线程块最大线程数、每个处理器簇最大驻留线程数、设备中的处理器簇数量等可参考WILT 3.2节

**代码如下：**

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include <stdio.h>

int main()

{

int numDevices;

cudaGetDeviceCount(&numDevices);

printf("GPU数量:%d\n", numDevices);

for (int i = 0; i < numDevices; ++i)

{

printf("----------------------\n");

cudaDeviceProp cdp;

cudaGetDeviceProperties(&cdp, i);

printf("设备编号: %d\n", i+1);

printf("设备名称: %s\n", cdp.name);

printf("计算能力: %d.%d\n", cdp.major, cdp.minor);

printf("设备全局内存总量 : %lu bytes\n",cdp.totalGlobalMem);

printf("每线程块最大线程数: %d\n", cdp.maxThreadsPerBlock);

printf("每个线程块的共享内存大小：: %lu bytes \n",cdp.sharedMemPerBlock);

printf("设备上一个线程块（Block）种可用的32位寄存器数量%d:\n", cdp.regsPerBlock);

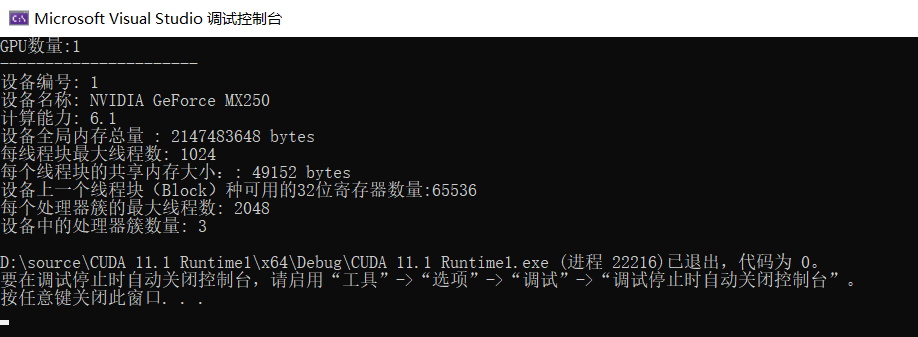
printf("每个处理器簇的最大线程数: %d\n", cdp.maxThreadsPerMultiProcessor);

printf("设备中的处理器簇数量: %d\n", cdp.multiProcessorCount);

}

}

**截图如下：**



* 1. 自己从头编写并运行VectorSum内核  
     新建.cu文件删掉模板参照课件代码片段写自己的向量加法程序（注意：课件代码故意留了小bug）向量大小设为128编译运行，截取结果，放入报告

代码如下：

#include <device\_launch\_parameters.h>

#include "cuda\_runtime.h"

#include <stdlib.h>

#include <iostream>

using namespace std;

\_\_global\_\_ void addKernel(int A[], int B[], int C[])

{

int i = threadIdx.x;

C[i] = A[i] + B[i];

}

int main()

{

int\* A, \* Ad, \* B, \* Bd, \* C, \* Cd;

int n = 128;

int size = n \* sizeof(int);

// CPU端分配内存

A = (int\*)malloc(size);

B = (int\*)malloc(size);

C = (int\*)malloc(size);

// 初始化数组

for (int i = 0; i < n; i++)

{

A[i] = 60;

B[i] = 70;

}

// GPU端分配内存

cudaMalloc((void\*\*)&Ad, size);

cudaMalloc((void\*\*)&Bd, size);

cudaMalloc((void\*\*)&Cd, size);

// CPU的数据拷贝到GPU端

cudaMemcpy(Ad, A, size, cudaMemcpyHostToDevice);

cudaMemcpy(Bd, B, size, cudaMemcpyHostToDevice);

cudaMemcpy(Bd, B, size, cudaMemcpyHostToDevice);

// 定义kernel执行配置，（128/128）个block，每个block里面有128个线程

dim3 dimBlock(128);

dim3 dimGrid(n / 128);

// 执行kernel

addKernel <<<dimGrid, dimBlock >>> (Ad, Bd, Cd);

// 将在GPU端计算好的结果拷贝回CPU端

cudaMemcpy(C, Cd, size, cudaMemcpyDeviceToHost);

cout << "A[i]=60,B[i]=70,C的结果为"<<endl;

for (int i = 0; i < n; i++)

{

printf("C[%d]=%d\t",i+1,C[i]);

if ((i+1)% 5 == 0)

{

cout << endl;

}

}

// 释放CPU端、GPU端的内存

free(A);

free(B);

free(C);

cudaFree(Ad);

cudaFree(Bd);

cudaFree(Cd);

return 0;

}

截图如下：

