作业1 查询设备并编写向量计算程序

姚舜宇 1190202107

1. 查询设备

调用cudaGetDeviceCount()得到GPU设备的数量调用cudaGetDeviceProperties()函数得到GPU设备的属性结构体。

查看GPU数量的代码如下：

|  |
| --- |
| int Device\_count = 0;  cudaGetDeviceCount(&Device\_count);  printf("cuda Device Count: %d\n",Device\_count); |

查看设备属性的代码如下：

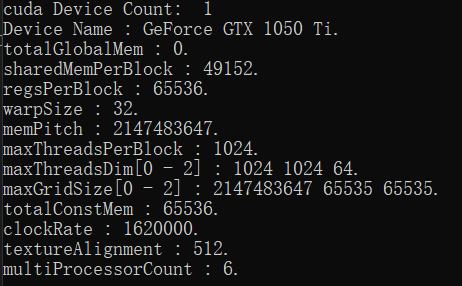
|  |
| --- |
| for (int i = 0; i < Device\_count; i++) {  cudaDeviceProp prop;  if (cudaGetDeviceProperties(&prop, i) == cudaSuccess) {  if (prop.major >= 1) {  printDeviceProp(prop);  break;  }  }  } |

其中调用自定义函数printDeviceProp打印结果。

函数编写如下：

|  |
| --- |
| void printDeviceProp(const cudaDeviceProp& prop)  {  printf("Device Name : %s.\n", prop.name);//设备名称  printf("totalGlobalMem : %d.\n", prop.totalGlobalMem);//全局储存器的大小  printf("sharedMemPerBlock : %d.\n", prop.sharedMemPerBlock);//共享储存器的大小  printf("regsPerBlock : %d.\n", prop.regsPerBlock);//寄存器的数目  printf("warpSize : %d.\n", prop.warpSize);//线程束中线程多少  printf("memPitch : %d.\n", prop.memPitch);//对显存访问时对齐时的pitch的最大值  printf("maxThreadsPerBlock : %d.\n", prop.maxThreadsPerBlock);//一个block中最多可以有的线程数  printf("maxThreadsDim[0 - 2] : %d %d %d.\n", prop.maxThreadsDim[0], prop.maxThreadsDim[1], prop.maxThreadsDim[2]);//block内3维度中各维度的最大值  printf("maxGridSize[0 - 2] : %d %d %d.\n", prop.maxGridSize[0], prop.maxGridSize[1], prop.maxGridSize[2]);//Grid内三维度中各维度的最大值  printf("totalConstMem : %d.\n", prop.totalConstMem);//常数储存器的大小  printf("clockRate : %d.\n", prop.clockRate);//显存的频率  printf("textureAlignment : %d.\n", prop.textureAlignment);//对纹理单元访问时对其参数的最大值  printf("multiProcessorCount : %d.\n", prop.multiProcessorCount);//设备中流多处理器(SM)的个数  } |

程序运行结果：



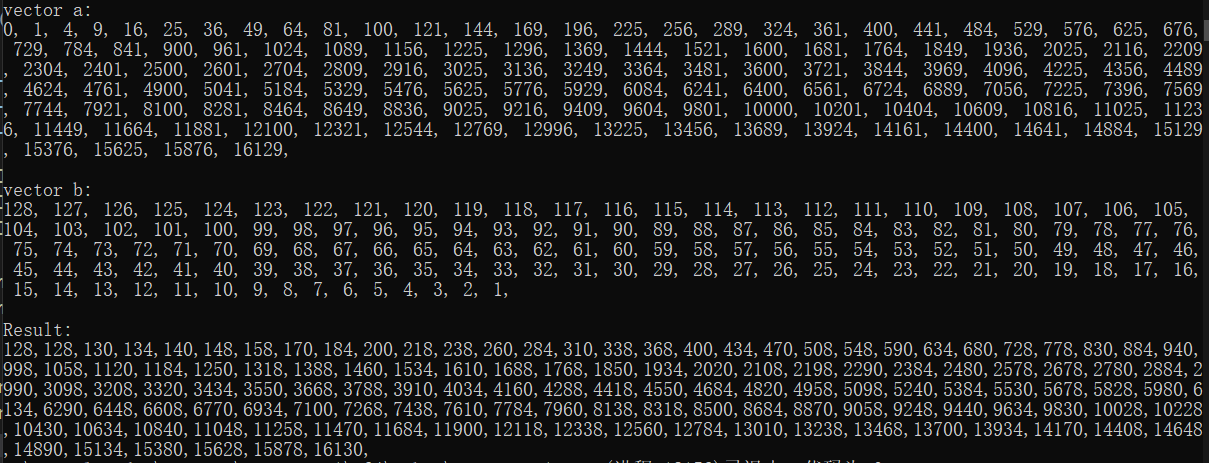
1. 向量计算程序

程序流程：

1. 分配CPU内存并初始化两个向量
2. 分配GPU内存并将数据从主机复制到GPU
3. 调用内核函数进行向量相加
4. 数据从GPU复制回主机
5. 输出结果
6. 释放CPU和GPU内存

向量a初始化为i\*i，向量b初始化为128-i ，i从0增加到127。

程序结果如下：



完整代码如下：

|  |
| --- |
| #include "cuda\_runtime.h"  #include "device\_launch\_parameters.h"  #include <stdio.h>  #include <math.h>  \_\_global\_\_ void add\_Kernel(int\* array\_a, int\* array\_b, int\* array\_c)  {  const unsigned int i = blockIdx.x \* blockDim.x + threadIdx.x;  array\_c[i] = array\_a[i] + array\_b[i];  }  int main()  {  int arraySize = 128;  int\* a, \* b;  int\* c ;  a = (int \*)malloc(arraySize \* sizeof(int));  b = (int \*)malloc(arraySize \* sizeof(int));  c = (int \*)malloc(arraySize \* sizeof(int));      for (int i = 0; i < arraySize; i++)  {  a[i] = i \* i;  b[i] = 128 - i;  }  printf("vector a:\n");  for (int i = 0; i < arraySize; i++) {  printf("%d, ", a[i]);  }  printf("\n\nvector b:\n");  for (int i = 0; i < arraySize; i++) {  printf("%d, ", b[i]);  }  printf("\n");  int\* a\_dev;  int\* b\_dev;  int\* c\_dev;  cudaMalloc((void\*\*)&a\_dev, arraySize \* sizeof(int));  cudaMalloc((void\*\*)&b\_dev, arraySize \* sizeof(int));  cudaMalloc((void\*\*)&c\_dev, arraySize \* sizeof(int));  cudaMemcpy(a\_dev, a, arraySize \* sizeof(int), cudaMemcpyHostToDevice);  cudaMemcpy(b\_dev, b, arraySize \* sizeof(int), cudaMemcpyHostToDevice);  add\_Kernel << <1, 128 >> > (a\_dev, b\_dev, c\_dev);  cudaThreadSynchronize();  cudaMemcpy(c , c\_dev, arraySize \* sizeof(int), cudaMemcpyDeviceToHost);  printf("\nResult:\n");  for (int i = 0; i < arraySize; i++) {  printf("%d,", c[i]);  }  cudaFree(a\_dev);  cudaFree(b\_dev);  cudaFree(c\_dev);  free(a);  free(b);  return 0;  } |