**任务描述**

本关任务：编写使用CUDA计算斐波那契数列的C语言小程序，并按要求输出对应的斐波那契数列。

**相关知识**

为了完成本关任务，你需要掌握：

* 什么是CUDA
* 如何简单使用CUDA C

**什么是CUDA**

CUDA(Compute Unified Device Architecture)的简称，是由NVIDIA公司创立的基于他们公司生产的图形处理器GPUs(Graphics Processing Units)的一个并行计算平台和编程模型。

通过CUDA，GPUs可以很方便地被用来进行通用计算。在没有CUDA之前，GPUs一般只用来进行图形渲染（如通过OpenGL，DirectX）。开发人员可以通过调用CUDA的API，来进行并行编程，达到高性能计算目的。NVIDIA公司为了吸引更多的开发人员，对CUDA进行了编程语言扩展，如CUDA C/C++,CUDA Fortran语言。

在CUDA中通常将CPU及系统的内存称为**主机**，将GPU及GPU本身的显示内存称为**设备**。CUDA中能够操作的最基本单位叫**线程**，一般通过GPU的一个核进行处理。多个线程能够组成**线程块**，而这些线程块是并行执行的，线程块间无法通信，也没有执行顺序。多个线程块能够组成**网格**。

**如何简单使用CUDA C**

其实写CUDA C程序和写C语言的程序差不多，只不过在写代码的时候多了几个CUDA的概念而已。

首当其冲的就是主机与设备，由于CUDA的架构设计，主机内存的指针和设备内存的指针是不能直接赋值使用的。所以CUDA C提供了分配设备内存与内存拷贝的API。

想要分配设备内存的话就需要用到cudaMalloc函数。该函数的声明如下：

1. cudaError\_t cudaMalloc (void \*\*devPtr, size\_t size );

其中：

* devPtr: 分配设备内存的指针的地址
* size: 想要分配多大空间的设备内存

想要对设备内存中的数据进行拷贝的话，就需要用到cudaMemcpy函数。该函数的声明如下：

1. cudaError\_t cudaMemcpy ( void\* dst, const void\* src, size\_t count, cudaMemcpyKind kind );

其中：

* dst: 想要把数据拷贝到哪里
* src: 想要拷贝的数据来自哪里
* count: 拷贝多少内存
* kind: 拷贝的动作类型，比如想要将设备内存拷贝到主机内存，就传入cudaMemcpyDeviceToHost。

有分配内存的API，当然也有释放内存的API。cudaFree函数可以释放使用cudaMalloc分配的内存。

对内存有了一定的理解后，我们就需要知道怎样去使用这些内存了。在CUDA C中想要区分主机代码和设备代码很简单，就是看有没有特殊的前缀，例如\_\_global\_\_。在函数前面加上\_\_global\_\_表示这个函数是要在GPU上执行的。例如如下计算数组内所有数据的平方和的函数就是可以在GPU上执行的：

1. #define DATA\_SIZE 100
2. \_\_global\_\_ static void sumOfSquares(int \*num, int\* result)
3. {
4. int sum = 0;
5. int i;
6. for(i = 0; i < DATA\_SIZE; i++) {
7. sum += num[i] \* num[i];
8. }
9. \*result = sum;
10. }

那么怎样调用这个函数呢？也很简单，例如：

1. sumOfSquares<<<1, 1>>>(gpudata, result);

其中两个1分别代表使用多少个线程块，以及每个线程块使用多少个线程来执行sumOfSquares函数。很显然现在只用了一个线程去跑，相当于是一个串行程序。但是如果现在想要并行计算，首先就要增加线程数量。如果改为如下代码，就相当于有10个线程在并发执行。

1. sumOfSquares<<<1, 10>>>(gpudata, result);

但是仅仅做这样的修改，还是不能实现10个线程并发计算数组中所有数字的平方和。那还需要做什么样的修改，才能实现并发呢？我们可以使用threadIdx。threadIdx是一个CUDA的内建的变量，表示目前的thread是第几个thread(由0开始计算)。

此时能够很容易的想到，将100个数字分配给0到9号线程，每个线程将自己的那部分数据的平方和算出来，然后存到一个长度为10的数组中。代码如下：

1. #define DATA\_SIZE 100
2. #define THREAD\_NUM 10
3. \_\_global\_\_ static void sumOfSquares(int \*num, int\* result)
4. {
5. //result为长度为10的数组，存放每个线程的平方和
6. int tid = threadIdx.x;
7. const int size = DATA\_SIZE/THREAD\_NUM;
8. int sum = 0;
9. int i;
10. for(i = tid\*size; i < (tid+1)\*size; i++) {
11. sum += num[i] \* num[i];
12. }
13. result[tid] = sum;
14. }

然后将设备中的数据拷贝到主机内存，最后再由主机累加所有的平方和。代码如下：

1. sumOfSquares<<<1,THREAD\_NUM>>>(gpudata,result);
2. int sum[THREAD\_NUM];
3. cudaMemcpy(&sum,result,sizeof(int)\*THREAD\_NUM,cudaMemcpyDeviceToHost);
4. //最终结果
5. int final\_sum = 0;
6. for(int i = 0; i < THREAD\_NUM; i++)
7. {
8. final\_sum += sum[i];
9. }
10. printf("%d\n", final\_sum);

这样就实现了10个线程并行计算平方和的功能。

**编程要求**

根据提示，在右侧编辑器补充代码，使用CUDA实现打印斐波那契额数列的功能。并且需要完成的输入与输出的格式如下：

* 输入: 需要输出的斐波那契额数列的长度，类型为int，如4
* 输出: 打印出斐波那契数列，如输入为4，输出为1 1 2 3

**PS：斐波那契数列中第n个值可以使用如下公式计算：**a\_n=\frac{1}{\sqrt 5}[(\frac{1+\sqrt 5}{2})^n - (\frac{1-\sqrt 5}{2})^n]*a*​*n*​​=​√​5​​​​​1​​[(​2​​1+√​5​​​​​)​*n*​​−(​2​​1−√​5​​​​​)​*n*​​]

**测试说明**

平台会对你编写的代码进行测试：

测试输入：  
1  
预期输出：  
1

测试输入：  
4  
预期输出：  
1 1 2 3

**注意：**打印完斐波那契数列之后要换行。