目录

[1 文档概述 2](#_Toc433360038)

[2 代码目录结构 2](#_Toc433360039)

[3 Common.h/cu 2](#_Toc433360040)

[3.1 全局变量 2](#_Toc433360041)

[3.2 枚举变量 3](#_Toc433360042)

[3.3 结构体和基础类 4](#_Toc433360043)

[4 Distribution.h/cu 4](#_Toc433360044)

[5 Case.h/cu 6](#_Toc433360045)

[6 main.cu 8](#_Toc433360046)

# 1 文档概述

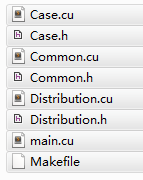
本文为“对GPU内存访存测试”实验代码的详细描述文档。

其中第2章为对代码目录结构，简单介绍了每个文件中的代码功能；

第3-5章详细介绍了每个文件中的代码；

第6章，main.cu文件其内为测试代码，在这一章介绍了测试demo以及一些注意问题。

# 2 代码目录结构



共有8个文件，三大部分（Case，Common，Distribution）、一个main文件和一个makefile文件。

Common.h/cu：本文件中包括全局变量、枚举变量、结构体和基础类的定义

Distribution.h/cu：本文件中包括生成各种分布数据生成函数的声明和定义。

Case.h/cu：核心测试类

Main.cu：主函数，内含demo

# 3 Common.h/cu

## 3.1 全局变量

全局变量包括：测试数据类型、测试数据范围（MIN、MAX）、PI以及e。

其中测试数据范围是与测试数据类型有关。

|  |
| --- |
| // 测试数据的类型  #define DATA\_TYPE unsigned char  // !!注意，这里的最大最小值跟 DATA\_TYPE 的类型有关  #define MIN 0  #define MAX 256  #ifndef PI  #define PI 3.141592654  #endif  #ifndef e  #define e 2.718281828459  #endif |

## 3.2 枚举变量

定义了数据内容分布类型、数据访问类型和数据组织形式。

其中数据内容分布类型有六种，数据访问类型有七种，数据组织形式有三种。详见注释。

|  |
| --- |
| // 数据内部分布  enum data\_content  **{**  dc\_random **=** 0**,** // 随机分布  dc\_standard\_normal**,** // 标准正态分布  dc\_poisson**,** // 泊松分布  dc\_uniform**,** // 均匀分布  dc\_geometric**,** // 几何分布  dc\_exponential // 指数分布  **};**  enum access\_mode  **{**  am\_sequential**=**0**,** // 顺序访问  am\_step**,** // step 访问  am\_random**,** // 随机访问  am\_standard\_normal**,** // 标准正态分布访问  am\_poisson**,** // 泊松分布访问  am\_geometric**,** // 几何分布访问  am\_exponential // 指数分布访问  **};**  // 数据组织形式  enum data\_form  **{**  df\_1D **=** 0**,**  df\_2D**,**  df\_tree  **};** |

## 3.3 结构体和基础类

包括一维模拟二维的结构体、树节点和树类。

其中Tree结构使用**完全二叉树**，存储方式选用**孩子表示法**。

|  |
| --- |
| /\* 树节点 \*/  class Node  **{**  public**:**  Node**\*** left**;**  Node**\*** right**;**  DATA\_TYPE data**;**  **};**  // 一维模拟二维  **typedef** struct data2D\_st  **{**  size\_t rows**;**  size\_t cols**;**  DATA\_TYPE **\***data**;**  size\_t pitchBytes**;**  **}**Data2D**;**  class Tree  **{**  public**:**  // 申请空间并设置父节点和子节点的关系  Tree**(**int \_num**);**  Tree**(){}**  **~**Tree**();**  Node **\***nodes**;**  int num**;** // 节点数量  **};** |

# 4 Distribution.h/cu

根据分布类型生成各种分布数据和根据。根据数据访问类型生成访问下标数据。

几种分布的公式如下：

（1）正态分布

http://h.hiphotos.baidu.com/baike/s%3D205/sign=2abf505a42166d223c77129473220945/342ac65c1038534384b650b09213b07eca808822.jpg

（2）泊松分布

http://c.hiphotos.baidu.com/baike/s%3D114/sign=bc59b0147dd98d1072d40830153eb807/574e9258d109b3de0e06c280cdbf6c81810a4c98.jpg

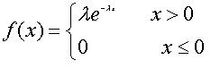
（3）均匀分布

Y = x

（4）几何分布

http://g.hiphotos.baidu.com/baike/s%3D219/sign=990b72acb68f8c54e7d3c22e03282dee/314e251f95cad1c891dc48657d3e6709c93d5108.jpg

（5）指数分布



在Distribution.h文件中，对每一种分布重载了三次，分别用于生成一维（或一维模拟二维）数据（DATA\_TYPE**\*** data）、访问下标数据（int**\*** data）和树节点数据（Node **\***nodes）。实现方式为传出min、max，分布参数、size和数据指针，在函数内生成数据并赋值。

|  |
| --- |
| int random**(**int min**,** int max**,** int size**,** DATA\_TYPE**\*** data**);**  int random**(**int min**,** int max**,** int**\*** data**);**  int random**(**int min**,** int max**,** int size**,** Node **\***nodes**);**  int standard\_normal**(**int min**,** int max**,** int miu**,** int sigma**,** int size**,** DATA\_TYPE**\*** data**);**  int standard\_normal**(**int min**,** int max**,** int miu**,** int sigma**,** int**\*** data**);**  int standard\_normal**(**int min**,** int max**,** int miu**,** int sigma**,** int size**,** Node **\***nodes**);**  int poisson**(**int min**,** int max**,** int Lambda**,** int size**,** DATA\_TYPE**\*** data**);**  int poisson**(**int min**,** int max**,** int Lambda**,** int**\*** data**);**  int poisson**(**int min**,** int max**,** int Lambda**,** int size**,** Node **\***nodes**);**  int uniform**(**int min**,** int max**,** int size**,** DATA\_TYPE**\*** data**);**  int uniform**(**int min**,** int max**,** int**\*** data**);**  int uniform**(**int min**,** int max**,** int size**,** Node **\***nodes**);**  int geometric**(**double probability**,** int min**,** int max**,** int size**,** DATA\_TYPE**\*** data**);**  int geometric**(**double probability**,** int min**,** int max**,** int**\*** data**);**  int geometric**(**double probability**,** int min**,** int max**,** int size**,** Node **\***nodes**);**  int exponential**(**double lambda**,** int min**,** int max**,** int size**,** DATA\_TYPE**\*** data**);**  int exponential**(**double lambda**,** int min**,** int max**,** int**\*** data**);**  int exponential**(**double lambda**,** int min**,** int max**,** int size**,** Node **\***nodes**);** |

在Distribution.cu文件中，参照三篇博客实现了生成各种分布数据。

// 代码实现参考

// http://blog.csdn.net/zhengnanlee/article/details/12619483

// http://m.blog.csdn.net/blog/asiaLIYAZHOU/45509047

// http://www.cnblogs.com/yeahgis/archive/2012/07/15/2592696.html

# 5 Case.h/cu

该类为核心组织类，Case.h内为Case类，代码如下：

|  |
| --- |
| class Case  **{**  public**:**  // 初始化默认参数  Case**()**;  **~**Case**();**  // 数据组织形式  // 一维数组，树等  data\_form df**;**  // 数据大小  int size**;**  int r**,**c**;** // 二维数组所需数据  // 数据内容形式  // 随机、各种分布  data\_content dc**;**  // 根据数据组织形式、数据大小和数据内容形式初始化数据  int initData**();**  // 线程数目  int thread\_num**;**  int block\_size**;** // 线程块大小  // 不同的访问方式  access\_mode am**;**  int step**;**  int am\_num**;** // 访问数据的数量  // 三个运行函数，即核心函数  // 分别得出在不同内存下的访问性能  // 每个函数内执行流程如下：  // (1)（申请空间，初始化数据需提前完成）  // (2)数据拷贝，执行核函数  // (3)释放数据  int global\_run**();**  int shared\_run**();**  int constant\_run**();**  private**:**  // 数据组织形式  DATA\_TYPE **\***data1D**;**  Data2D data2D**;**  Tree **\***tree**;**  **};** |

下面解释类成员变量。

|  |  |
| --- | --- |
| data\_form df; | 数据组织形式，一维、二维和树结构 |
| size | 当为一维数组和树结构时，该成员函数代表测试数据大小 |
| r**,**c | 当为二维数组时，r代表行数，c代表列数。Size = r\*c |
| data\_content dc; | 数据内容分布 |
| access\_mode am; | 不同分布的访问方式 |
| thread\_num | 线程数量，在这里，线程数量等于数据大小 |
| block\_size | 线程块大小 |
| step | Step访问时所需变量 |
| am\_num | 每个线程访问的数据数目 |

下面解释类成员函数

|  |  |
| --- | --- |
| Case() | 构造函数，初始化各种默认参数 |
| initData(); | 根据df（数据组织形式）和dc（数据内容形式），调用对应的分布函数生成分布数据。 |
| global\_run();  shared\_run();  constant\_run(); | 三个运行核心函数，分别得出在不同内存下的访问性能。函数内流程基本相同。  （1）根据数据组织形式（一维、二维、树结构），申请device空间并拷贝数据；  （2）根据不同的访问分布方式（分为三类，sequential访问、step访问、其他分布访问），执行不同的核函数。  在其他分布访问方式时，需要在核函数外根据不同的访问方式在host端生成访问下标，再拷贝到device端（虽然可以用curand在核函数外生成一些分布，但am\_geometric分布和am\_exponential分布无法生成，为了考虑性能测试统一，不使用curand）。 |

Case.cu内为constant memory定义，核函数和Case类成员函数实现。

Constant memory的大小必须在编译前确定，所以对于不同constant memory数据大小的测试只能一次次手动改。

三种内存（global、shared、constant），三种数据组织形式（一维、二维、树结构），三种访问分布（sequential、step、Common），共27个核函数。核函数名的格式为：**\_数据组织形式\_内存种类\_访问分布Ker()**.

Global memory核函数内流程为：（1）计算下标；（2）下标越界判断（3）读取am\_num个数据，并相加。

Shared memory核函数内流程为：（1）计算下标；（2）下标越界判断；（3）根据copy\_num\_per\_thread拷贝数据到shared memory中。copy\_num\_per\_thread是参数，事先在核函数外计算。（4）读取am\_num个数据，并相加。

Constant memory核函数内流程与global memory核函数相同。

# 6 main.cu

在该文件中，给出了测试demo。

代码思路为先设计好各个参数，比如Global memory数据大小、线程数、block大小等。然后再循环各个参数跑就可以了。下面对其中一个进行解释。

在循环中，（1）首先给Case类赋值各个参数；（2）初始化数据；（3）打印部分参数；（4）根据访问方式（原因是step访问需要设置step的大小）分为两种，step访问和其他。（5）然后计时运行函数。

|  |
| --- |
| // 1D Global  **for** **(**int gs **=** 0**;** gs **<** global\_size\_num**;** gs**++)**  **for** **(**int bs**=** 0**;** bs **<** block\_size\_num**;** bs**++)**  **for** **(**int \_dc**=** 0**;** \_dc **<** dc**;** \_dc**++)**  **for** **(**int \_am **=** 0**;** \_am **<** am**;** \_am**++)**  **for** **(**int an **=** 0**;** an **<** am\_num\_num**;** an**++)** **{**  Case c**;**  c**.**df **=** df\_1D**;**  c**.**size **=** global\_size**[**gs**];**  c**.**thread\_num **=** c**.**size**;** // 线程数与数据量大小相同  c**.**block\_size **=** block\_size**[**bs**];**  c**.**dc **=** **(**enum data\_content**)**\_dc**;**  c**.**am **=** **(**enum access\_mode**)**\_am**;**  c**.**am\_num **=** am\_num**[**an**];**  c**.**initData**();**  cout**<<**EnumToString**(**c**.**df**)**  **<<** " Global:"  **<<**" size="**<<** c**.**size  **<<**" block\_size="**<<**c**.**block\_size  **<<**" data\_content="**<<** EnumToString**(**c**.**dc**)**  **<<**" access\_mode="**<<**EnumToString**(**c**.**am**);**  float runTime**;**  // step access mode  **if** **(**\_am **==** 1**)** **{**  **for** **(**int s **=** 0**;** s **<** step\_num**;** **++**s**)** **{**  c**.**step **=** step**[**s**];**  cout **<<** " step=" **<<** step**[**s**];**  cudaEvent\_t start**,** stop**;**  cudaEventCreate**(&**start**);**  cudaEventCreate**(&**stop**);**  cudaEventRecord**(**start**,** 0**);**  **for** **(**int i **=** 0**;** i **<** LOOP**;** i**++)**  c**.**global\_run**();**  cudaEventRecord**(**stop**,** 0**);**  cudaEventSynchronize**(**stop**);**  cudaEventElapsedTime**(&**runTime**,** start**,** stop**);**  cout **<<** " runTime=" **<<** **(**runTime**)** **/** LOOP **<<** " ms" **<<** endl**;**  **}**  **}** **else** **{**  cudaEvent\_t start**,** stop**;**  cudaEventCreate**(&**start**);**  cudaEventCreate**(&**stop**);**  cudaEventRecord**(**start**,** 0**);**  **for** **(**int i **=** 0**;** i **<** LOOP**;** i**++)**  c**.**global\_run**();**  cudaEventRecord**(**stop**,** 0**);**  cudaEventSynchronize**(**stop**);**  cudaEventElapsedTime**(&**runTime**,** start**,** stop**);**  cout **<<** " runTime=" **<<** **(**runTime**)** **/** LOOP **<<** " ms" **<<** endl**;**  **}**      **}** |

下图是运行截图

