# CUDA 原子操作

和许多多线程并行问题一样，CUDA也存在互斥访问的问题，即当一个线程改变变量Ｘ,而另外一个线程在读取变量Ｘ的值，执行原子操作类似于有一个自旋锁，只有等Ｘ的变量在改变完成之后，才能执行读操作，这样可以保证每一次读取的都是最新的值.

在kernel 程序中，做统计累加，都需要使用原子操作：atomicAdd();

原子操作很明显的会影响程序性能，所以可以的话，尽可能避免原子操作．

## CUDA原子操作API:

atomicAdd()  
int atomicAdd(int\* address, int val);  
unsigned int atomicAdd(unsigned int\* address,  
                           unsigned int val);  
unsigned long long int atomicAdd(unsigned long long int\* address,  
                                        unsigned long long int val);  
读取位于全局或共享存储器中地址address 处的32 位或64 位字old，计算(old + val)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
只有全局存储器支持64 位字。  
  
C.1.2  atomicSub()  
int atomicSub(int\* address, int val);  
unsigned int atomicSub(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算(old - val)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。

C.1.3  atomicExch()  
int atomicExch(int\* address, int val);  
unsigned int atomicExch(unsigned int\* address,  
                            unsigned int val);  
unsigned long long int atomicExch(unsigned long long int\* address,  
                                        unsigned long long int val);  
float atomicExch(float\* address, float val);  
读取位于全局或共享存储器中地址address 处的32 位或64 位字old，并将val 存储在存储器的同一地址中。这两项操作在一次原子事务中执行。该函数将返回old。  
只有全局存储器支持64 位字。  
  
  
C.1.4  atomicMin()  
int atomicMin(int\* address, int val);  
unsigned int atomicMin(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算old 和val 的最小值，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.1.5  atomicMax()  
int atomicMax(int\* address, int val);  
unsigned int atomicMax(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算old 和val 的最大值，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.1.6  atomicInc()  
unsigned int atomicInc(unsigned int\* address,  
                       unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算 ((old >= val) ? 0 : (old+1))，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.1.7  atomicDec()  
unsigned int atomicDec(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算 (((old == 0) | (old > val)) ? val : (old-1))，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.1.8  atomicCAS()  
int atomicCAS(int\* address, int compare, int val);  
unsigned int atomicCAS(unsigned int\* address,  
                           unsigned int compare,  
                           unsigned int val);  
unsigned long long int atomicCAS(unsigned long long int\* address,  
                                       unsigned long long int compare,  
                                       unsigned long long int val);  
读取位于全局或共享存储器中地址address 处的32 位或64 位字old，计算 (old == compare ? val : old)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old（比较并交换）。  
只有全局存储器支持64 位字。

C.2位逻辑函数C.2.1  atomicAnd()  
int atomicAnd(int\* address, int val);  
unsigned int atomicAnd(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算 (old & val)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.2.2  atomicOr()  
int atomicOr(int\* address, int val);  
unsigned int atomicOr(unsigned int\* address,  
                          unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算 (old | val)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。  
  
  
C.2.3  atomicXor()  
int atomicXor(int\* address, int val);  
unsigned int atomicXor(unsigned int\* address,  
                           unsigned int val);  
读取位于全局或共享存储器中地址address 处的32 位字old，计算 (old ^ val)，并将结果存储在存储器的同一地址中。这三项操作在一次原子事务中执行。该函数将返回old。