

qqllu\_did的专栏

目录视图摘要视图RSS 订阅

个人资料



qqllu\_did\_lq

关注发私信

访问：20345次

积分：352

等级：BLOG 2

排名：千里之外

原创：15篇

转载：0篇

译文：0篇

评论：6条

文章搜索

Q

文章分类

- 最优化解法

(1)
- 聚类

(3)
- 矩阵

(2)
- 机器学习

(3)
- cuda学习笔记

(4)
- c++学习

(2)

文章存档

- 2015年07月

(1)
- 2015年06月

(1)
- 2015年05月

(3)
- 2015年04月

(1)
- 2015年01月

(3)

展开

异步赠书:9月重磅新书升级, 本本经典SDCC 2017之区块链技术实战线上峰会程序员9月书讯每周荐书:Java, Web, Python极客编程 (评论送书)

Cuda中Global memory中coalescing例程解释

标签：cuda 并行计算 实例

2015-05-20 03:031295人阅读评论(0)收藏举报

分类：cuda学习笔记(3)

版权声明：本文为博主原创文章，未经博主允许不得转载。

Global memory是cuda中最常见的存储类型,又叫做Device memory, 位于Host主机区域上, 它的生命周期是在整个Grid里面, 大约具有500个cycle latency。在cuda并行程序中, 尽量用Coalesing accessing的策略来最大化带宽bandwidth。什么是Coalesing accessing呢? 如图所示:



当半个Warp的16个threads在一次memory transaction中coalesced时, Global memory中的带宽得到了最大的利用。其中, 需要注意的是, Device在一次transaction中, 从global memory中可以一次读取32-bit, 64-bit, 128-bit, 例如

- 64 bytes - each thread reads a word: int, float, ...
- 128 bytes - each thread reads a double-word: int2, float2, ...
- 32 bytes (compute capability 1.2+) - each thread reads a short int.

下面有两个实例来说明Global memory中的coalescing问题:

```
1) float3型Uncoalesced
__global__ void accessFloat3(float3 *d_in,
```

阅读排行	
Caffe下自己的数据训练和测试	(7819)
MATLAB2014b下运行cuda...	(2706)
Cuda中Global memory中c...	(1294)
Jacobi迭代与SOR迭代求解...	(1210)
深度学习值得关注的75篇文章	(1171)
聚类之isodata算法	(883)
聚类之hierachical clusteri...	(793)
Share memory中bank con...	(709)
基于adaboost的人脸检测方法	(703)
Dogleg“狗腿”最优化算法	(601)
评论排行	
Caffe下自己的数据训练和测试	(5)
深度学习值得关注的75篇文章	(1)
Jacobi迭代与SOR迭代求解...	(0)
Ubuntu 14.04 64bit + CU...	(0)
MATLAB2014b下运行cuda...	(0)
基于adaboost的人脸检测方法	(0)
聚类之K-means均值聚类	(0)
聚类之hierachical clusteri...	(0)
矩阵之LU分解	(0)
聚类之isodata算法	(0)
推荐文章	
* CSDN日报20170828——《4个方法快速打造你的阅读清单》	
* Android检查更新下载安装	
* 动手打造史上最简单的 Recycleview 侧滑菜单	
* TCP网络通讯如何解决分包粘包问题	
* SDCC 2017之区块链技术实战线上峰会	
* 快速集成一个视频直播功能	
最新评论	
Caffe下自己的数据训练和测试 weifei4838 : 楼主有在windows下微调后画loss曲线吗, 求帮助	
Caffe下自己的数据训练和测试 jialangzhui4789 : 你的test.txt中没有给图像加标签, 那在测试时, 怎么得到准确率。	
Caffe下自己的数据训练和测试 元气少女缘结神 : 你好 可以帮忙看下我在caffe画accuracy曲线时出来的曲线图不对 loss曲线倒是的 ht...	
Caffe下自己的数据训练和测试 w20ss08 : 请问迭代4500000次(90期), 每1000次迭代, 我们测试学习网络验证数据, 我们设置初始的学习率...	
Caffe下自己的数据训练和测试 zjsyhs1 : 能共享下你的数据集吗	
深度学习值得关注的75篇文章 chensheng312 : 学习了~	

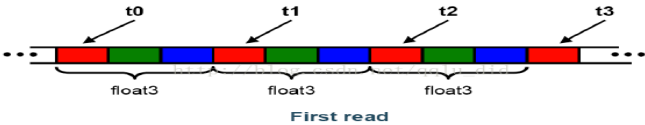
```
float3* d_out)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;

    float3 a = d_in[index];

    a.x += 2;
    a.y += 2;
    a.z += 2;

    d_out[index] = a;
}
```

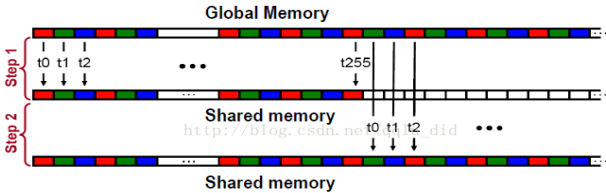
在这段代码中, float3有12个bytes, 不等于要求的4,8,16 bytes, 半个warp读取3个Global memory中非连续区域, 如图:



有三种方法可以解决这个问题

1:使用shared memory, 也叫做3-step approach

假如每个block中使用256个threads, 这样一个thread block需要 sizeof(float3)\*256 bytes的share memory空间, 每个thread读取3个单独的float型, 这实质上是指讲输入定义为float型, 在核函数数里面讲读取在share memory中的float变量转换为float3型并进行操作, 最后再转换成float型输出, 如图;



Similarly, Step3 starting at offset 512

代码如下:

关闭

```
__global__ void accessInt3Shared(float *g_in, float *g_out)
{
    int index = 3 * blockDim.x * blockDim.x + threadIdx.x;
    __shared__ float s_data[256*3];
    s_data[threadIdx.x] = g_in[index];
    s_data[threadIdx.x+256] = g_in[index+256];
    s_data[threadIdx.x+512] = g_in[index+512];
    __syncthreads();
    float3 a = ((float3*)s_data)[threadIdx.x];

    a.x += 2;
    a.y += 2;
    a.z += 2;

    ((float3*)s_data)[threadIdx.x] = a;
    __syncthreads();
    g_out[index] = s_data[threadIdx.x];
    g_out[index+256] = s_data[threadIdx.x+256];
    g_out[index+512] = s_data[threadIdx.x+512];
}
```

如果不好理解的话,假设我们的blockDim=4,取4个float3型变量,我们会发现,每-  
入操作(输出操作一样)为:

Thread 0:

S\_data[0]=g\_in[0]; S\_data[4]=g\_in[4]; S\_data[8]=g\_in[8];

Thread 1:

S\_data[1]=g\_in[1]; S\_data[5]=g\_in[5]; S\_data[9]=g\_in[9];

Thread 2:

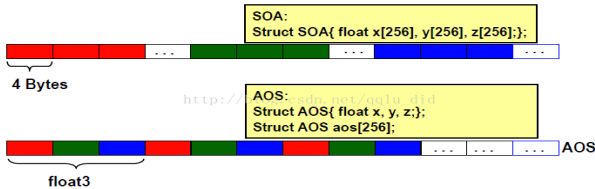
S\_data[2]=g\_in[2]; S\_data[6]=g\_in[6]; S\_data[10]=g\_in[10];

Thread 3:

S\_data[3]=g\_in[3]; S\_data[7]=g\_in[7]; S\_data[11]=g\_in[11];

可以看出,对于每个thread同一时刻(similar step)的数据读入,地址均是连续,这样就达到了coalescing。

2) 使用数组的结构体(SOA)来取代结构体的数组(AOS)



3) 使用alignment specifiers

\_\_align\_\_(X), where X = 4, 8, or 16

struct \_\_align\_\_(16) { float x; float y; float z;};

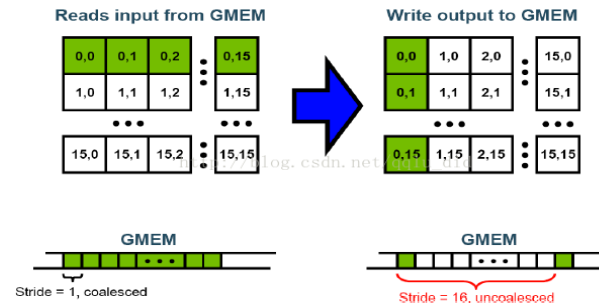
尽管这损失了比较多的空间:



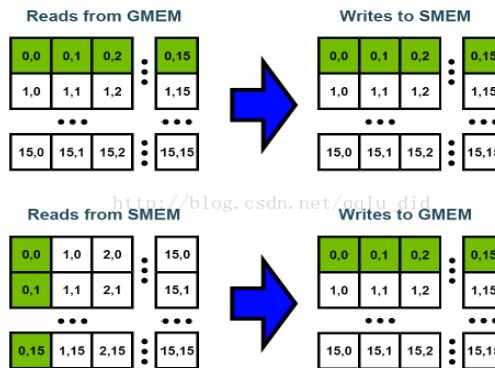
2) 第二个实例:矩阵转置 Matrix Transpose.

一般做法:Uncoalesced Transpose, GMEM为Global memory

关闭



我们发现一般的做法, 在写output时, 地址是不连续的, 即uncoalesced, 因此我们和memory存储输入数据, 根据转置的关系, 来实现coalescing, SMEM为shared m



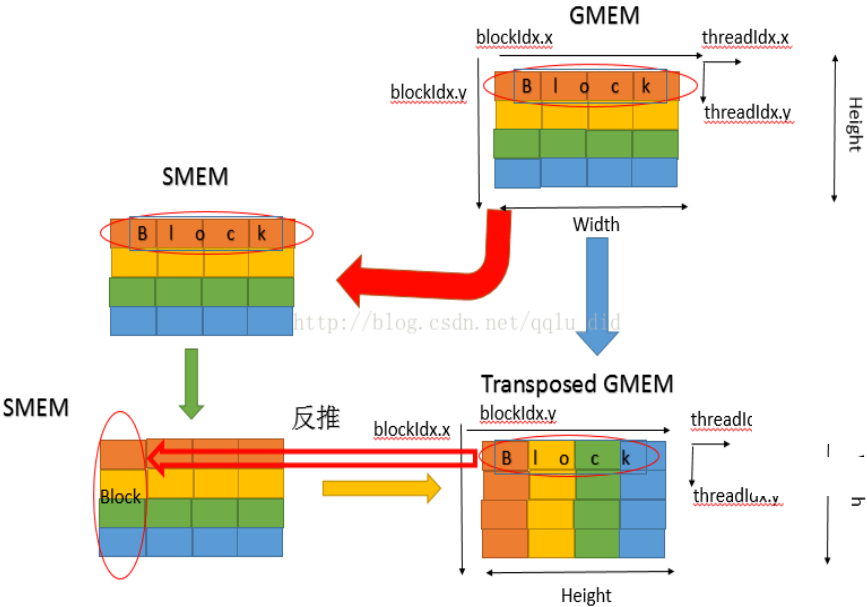
代码如下:

```
__global__ void transpose(float *odata, float *idata, int width, int height)
{
    __shared__ float block[BLOCK_DIM*BLOCK_DIM];
    unsigned int xBlock = blockDim.x * blockIdx.x;
    unsigned int yBlock = blockDim.y * blockIdx.y;
    unsigned int xIndex = xBlock + threadIdx.x;
    unsigned int yIndex = yBlock + threadIdx.y;
    unsigned int index_out, index_transpose;
    if (xIndex < width && yIndex < height)
    {
        unsigned int index_in = width * yIndex + xIndex;
        unsigned int index_block = threadIdx.y * BLOCK_DIM + threadIdx.x;
        block[index_block] = idata[index_in];
        index_transpose = threadIdx.x * BLOCK_DIM + threadIdx.y;
        index_out = height * (xBlock + threadIdx.y) + yBlock + threadIdx.x;
    }
    __syncthreads();
    if (xIndex < width && yIndex < height)
```

关闭

```
odata[index_out] = block[index_transpose];
```

程序的逻辑关系有时还挺绕的, 我们以一个4\*4矩阵为例, 将逻辑关系展示如下:



设dim3 gridDim(4,1), dim3 blockDim(1,4), 以橙色block为例, 如输入数据时, 将其放入到sharememory中, 代码体现在:

```
unsigned int index_in = width * yIndex + xIndex;
unsigned int index_block = threadIdx.y * BLOCK_DIM + threadIdx.x;
block[index_block] = idata[index_in];
```

接下来的代码实际上是将block的区域给换了, 如左下图所示, block换成了一列四种不同颜色的, 最终转置的矩阵如右下图所示, 从图示可以看出, 最终结果的坐标系Height、Width、blockIdx.x、blockIdx.y均对位变换了, 这时我们只需要找threadIdx.x'、threadIdx.y'与threadIdx.x、threadIdx.y之间的关系, 其实可以看出, 一个block里面的坐标系没有发生变换, 则threadIdx.x'=threadIdx.x, threadIdx.y'=threadIdx.y, 所以代码如下:

```
index_transpose = threadIdx.x * BLOCK_DIM + threadIdx.y;
index_out = height * (xBlock + threadIdx.y) + yBlock + threadIdx.x;
odata[index_out] = block[index_transpose];
```

总体来说, Global memory中coalescing就是保证其在数据访

储的变量尺寸为32、64、128 bit, 我们常常使用share memory来解决coalescing问题。

关闭

- 上一篇    Jacobi迭代与SOR迭代求解希伯特矩阵
- 下一篇    Share memory中bank conflict问题

相关文章推荐

- Cuda中Global memory中coalescing例程解释
- 大数据技术实战线上峰会--董西成
- CUDA编程——Memory Coalescing
- 30天系统掌握机器学习--唐宇迪
- 
- ORACLE数据库学习资料集锦
- 
- PHP从零开始实战篇

- 
- 玩转微信小程序第一篇
- 
- 深度学习案例分享—人脸检测
- MATLAB2014b下运行cuda6.5
- CUDA学习日志:线程协作与例程
- CUDA学习日志:入门例程和编程接口
-



[查看评论](#)

暂无评论


您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点,不代表CSDN网站的观点或立场

公司简介 | 招贤纳士 | 广告服务 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服    杂志客服    微博客服    webmaster@csdn.net    400-660-0108 | 北京创新乐知信息技术有限公司 版权所有 | 江苏知之为计算机有限公司 |

江苏乐知网络技术有限公司

京 ICP 证 09002463 号 | Copyright © 1999-2017, CSDN.NET, All Rights Reserved 

关闭