CUDA编程——Mars：MapReduce on GPU

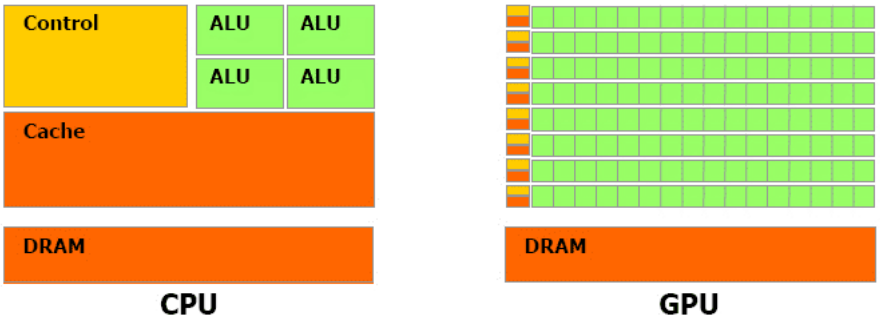
原创 ZhangJunior 2016-01-19 11:31:21 4027 收藏 1 版权
分类专栏： CUDA编程

CUDA编程——Mars：MapReduce on GPU

1 GPU加速机器学习

GPU是一种SIMT（单指令多线程）体系结构，即多个线程执行同一个指令，而每个线程操作的数据不同。这种结构令GPU天生具有大规模计算能力。GPU出色的浮点计算性能特别提高了深度学习两大关键活动：分类和卷积的性能，同时又达到所需的精准度。深度学习需要很高的内在并行度、大量的浮点计算能力以及矩阵预算，而GPU可以提供这些能力，并且在相同的精度下，相对传统CPU的方式，拥有更快的处理速度、更少的服务器投入和更低的功耗。NVIDIA介绍，TITAN X在工业标准模型AlexNet上，花了不到三天的时间、使用 120万个 ImageNet 图像数据集去训练模型，而使用16核心的 CPU 得花上四十多天。更震撼的是使用 NVIDIA推出的DIGITS DevBox [1]来训练 AlexNet 则只要13个小时就能完成。

然而，这种庞大的并行能力需要付出代价：必须编写专门的软件才能利用这样的优势，GPU编程相对于CPU编程需要更多程序员的付出。目前GPGPU的程序模型仍不成熟，将数据划分为不同粒度，送到GPU的每个流处理器（SP）运算，这些工作仍需要程序员手工完成。此外，由于GPU不具有分支预测等复杂的流程控制单元，对于高度分支的程序执行效率差。GPU核心是虚拟化的，线程调度由硬件完成，无法动态调度。程序员需要避免写有高度分支的程序。GPU由于没有足够大的cache，读写主存导致latency。程序员需要利用大量线程隐藏latency。另外不同厂商的GPU硬件架构不同，用户可以获得的细节有限。这些都导致在GPU上设计通用的计算框架，仍然具有很大挑战。



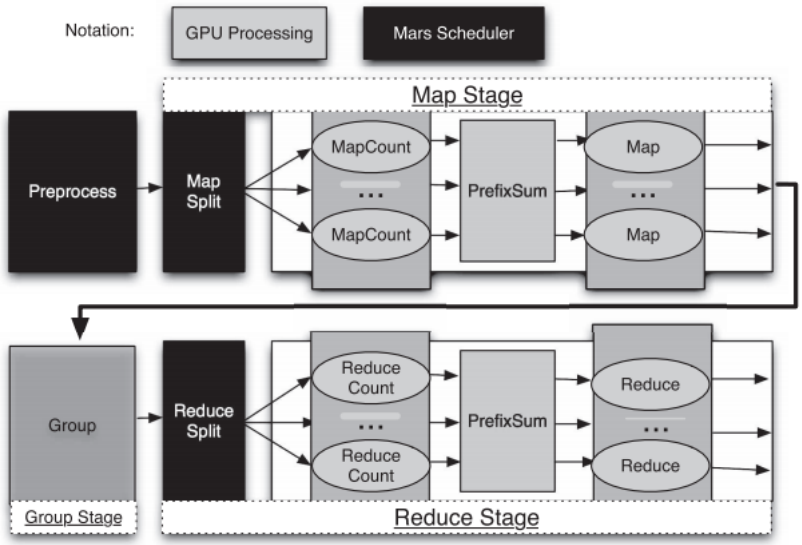
2 Mars

近来，一些GPGPU 编程框架被提出，如NVIDIA的CUDA和AMD的Brook+,这些框架大大提升了GPU可编程性。Wenbin Fang等认为这些编程语言的接口依赖与特定厂商，并且他们的硬件抽象不适合于开发复杂应用。所以提出了一个易于在GPU上编程的MapReduce框架[2]。Mars框架可以用在分布式环境中，如hadoop。Mars可以应用在多核CPUs，NVIDIA GPUs，AMD GPUs或者联合一个多核CPU和一个GPU的单机上。Mars解决了三个技术挑战：首先MapReduce根据数据分割任务，利用GPU执行大量并行线程时，负载不平衡是一个固有问题，特别是GPU的线程由硬件管理。其次，GPU缺乏有效的全局同步机制，Map或Reduce任务中的线程在输出缓存上常常发生写入冲突。尽管GPU现在已经支持原子操作，原子操作的缺陷却会伤害大量GPU线程的可扩展性[3]。Mars提出一个lock-free调度方法来减少GPU线程同步带来瓶颈。第三，MapReduce应用通常是数据密集，且结果的规模也是依数据
这两个特性导致GPU编程有以下需求：1）足够多线程隐藏内存延迟，充分利用设备内存的高带
预先在设备内存上分配输出缓冲区，利用DMA减少内

行运行，每个thread一次运算一个key/value pair，在Map阶段，框架平均分配key/value pairs到每个thread，Reduce阶段，Mars使用一种简单但高效的倾斜算法重新分配数据到Reduce任务，达到负载均衡。为了避免多线程写入冲突，Mars采用了一种lock-free策略保证并行程序的正确，仅付出很小的同步代价。

2.1 Mars 工作流程

Mars的工作流程如下图。



以Mars的word count为例，Mars读取文件，将文件切分为ceil（2048）大小的一块，这里的ceil（2048）是指≥2048字符长度的连续字符，即块以非空字符开始，结尾是偏移≥2048字符长度的第一个空字符的前一位。每一块分配给一个GPU thread，256个thread组成一个block，多个block组成一个grid，一次GPU内核函数调用执行一个grid。从调度和运行方式看，GPU上block概念和CPU上的进程很相似，一个进程占用一个CPU核运行，多个进程轮转调度；一个block占用一个GPU SM运行，多个block轮转调度。从这个角度看，GPU的SM很像GPU核。

2.2 MapSplit

假设原始数据放在磁盘上，Mars利用CPU程序从磁盘读取数据，将输入转换为key/value pairs保存在主存中，之后传输到GPU设备内存。MapSplit阶段，将输入分配给GPU thread，分配的方式是一种分段式扫描的方式。

分段扫描，就是对数据集进行有规律的扫描操作（最大值，最小值，总和等），并附带一个额外的数组，将原来的数组分成不同大小的块。每块分配一个或多个线程进行计算。由于附加的数组可以在运行时进行更新，因此如果分段扫描能保持在一个单独的线程块内执行，就可以减少调用多内核的需要。否则，则需要采用一种更简单的解决办法。分段式扫描能够在多数情况下正常工作，并且线程和线程块的数量能随着并行度增加或缩减灵活改变。

2.3 MapCount

MapCount用于计算Map输出的中间结果的大小，以便预先分配GPU内存，计算方式是通过求前缀和（Prefix Sum）获得输出大小和每个线程写入数据的位置。前缀和也叫累积和，一组数序列 $x_0, x_1, x_2, \ldots$ 的前缀和还是一组数序列 $y_0, y_1, y_2, \ldots$，计算方法如下：

$$y_0 = x_0$$
$$y_1 = x_0 + x_1$$
$$y_2 = x_0 + x_1 + x_2$$
$$\cdots$$

例如，自然数的前缀和是三角形数：

input numbers	1	2	3	4	5	6	...
prefix sums	1	3	6	10	15	21	...

2.4 Map&Group

通过前缀和，标记每个线程的输出的位置，提前分配GPU内存。最后GPU线程执行用户的Map函数，接着Map以lock-free方式获得每个线程输出结果的大小和写入位置，输出结果。
在Group阶段，按照key排序分组和hash分组都是可行的，Mars采用了排序分组，因为有些应用需要把输出排序，并且hash分组也必须为每个hash bucket进行排序。

2.5 Reduce

ReduceCount和MapCount相似，不再赘述。

Reduce阶段，把key相同的中间结果分配给一个GPU thread，由于不同key的记录数量不同，这可能造成线程负载不均衡。Mars采用了一种倾斜处理策略减缓负载不均衡问题，可以跨reduce workers分配负载，即使用户定义的Reduce操作之间是关联的。这个策略就是迭代运行两步：1）把数据分为M大小相同的块。2）对每个块执行Reduction，M个thread执行Reduce函数，计算单个块内的一组记录。注意：在每次迭代中，Reduction只在具有相同keys中间结果上执行。
接着Reduce以lock-free方式获得每个线程输出结果的大小和写入位置。最后把所有Reduce workers输出到一个缓存区域。

2.6 Lock-free方案

在GPU运算前，Mars已经在设备上以array格式分配好内存。然而，Map和Reduce输出的大小都是未知的，多线程在一个共享的array上写结果常常发生冲突。为了解决这两个问题，Mars提出了Lock-free方案。每个线程运行MapCount都会输出三个计数，如：中间结果的个数，中间结果keys的大小，和中间结果values的大小。根据中间结果key的大小，Mars计算prefix sum[6]，产生写入地址，该地址是一个输出array开始位置加偏移量。前缀和（prefix sum）计算在并行计算中很有用,因为在处理负载平衡问题时,经常需要将若干段数据重新平分,计算前缀和通常是一种有效的将数据平分的方法。

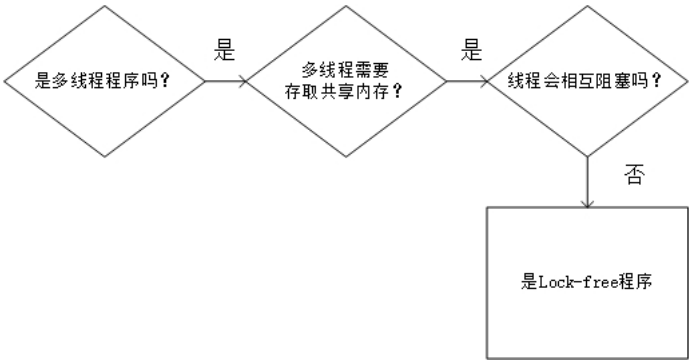
通过这些prefix sum，可以知道中间结果的准确大小，这样可以预先在设备上分配内存保存中间结果。由于每个Map有确定的和不重叠的结果缓存区，就可以避免写入冲突。Lock-free非常适合于大量线程并行运行的程序。

3 什么是lock-free?

众所周知，锁在解决并行过程中临界资源访问问题的同时可能会引入诸多新的问题，比如死锁(dead lock)，另外锁的申请/释放对性能也有不小的影响，当然最大的问题还在于使用锁的代码模块通常难以进行组合。
Lock-free的目标就是要消除锁对编程带来的不利影响。那么lock-free是什么？一个lock-free的解释是一个“锁无关”的程序能够确保执行它的所有线程中，如果某一个线程被挂起，那么其绝对不会阻止其他线程继续运行（Non-Blocking）[2]。

换句话说，各个线程不会互相阻塞，那么你的程序才能成为lock-free的。像我们平常用的互斥锁，当有线程获得锁，其他线程就被阻塞掉了，这里的问题就是如果获得锁的线程挂掉了，而且锁也没有释放，那么整个程序其实就被block在那了，而如果程序是lock-free的那么即使有线程挂掉，也不影响整个程序继续向下进行。所以，如果程序中的某一部分符合下面的条件判定描述，则我们称这部分程

合lock-free的。反过来说，如果某一部分程序不符合下面的条件描述，则称这部分程序是不符合 lock-free的。



是不是不用锁就是lock-free呢？举个例子：

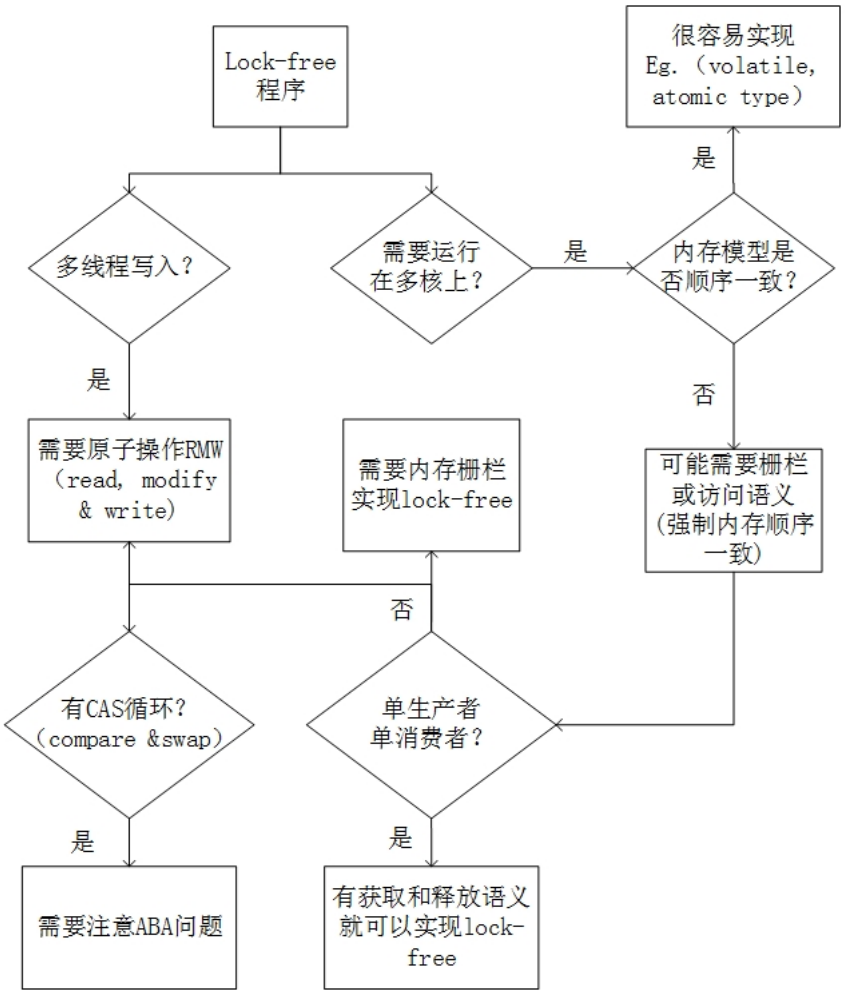
```
while (x == 0) {
    x = 1-x;
}
```

在这里如果两个线程同时执行，可能同时进入while循环，然后x两次改变值之后，依然是0，那么两个线程就会一直互相在这里阻塞掉了，所以这里虽然没有锁，依然不是lock-free的。

---

4 Lock-free的实现方式

当我们准备要满足 lock-free 编程中的非阻塞条件时，有一系列的技术和方法可供使用，如原子操作（Atomic Operations）、内存栅栏（Memory Barrier）、避免 ABA 问题（Avoiding ABA Problem）等。那么我们该如何抉择在何时使用哪种技术呢？可以根据下图中的引导来判断。



4.1 RMW

Read-modify-write是一类原子操作(such as test-and-set, fetch-and-add, and compare-and-swap)，即同时读取一个内存位置和写入一个新值，不论写入的新值是一个全新的值或是前一个值的函数。所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何线程切换。原子操作也大量用于非阻塞同步。

4.2 CAS

Compare-and-swap 比较内存中一个位置的内容和给定值，只有两个值相同时，用新的值更新内存中那个位置的内容。CAS由一个原子操作完成。原子性保证新的值是基于最新的信息计算的。如果那个值在这个过程中被其它线程更新过，则会发生写入失败。CAS的返回值表示操作是否成功，如可以返回一个bool值，这种CAS变体称为compare-and-set，也可以返回从内存中读到的值（不是被写入的值）。

```
function cas(p : pointer to int, old : int, new : int) returns bool {
    if *p ≠ old {
        return false
    }
    *p ← new
    return true
}
```

4.3 ABA problem

下面是 ABA 问题发生的过程：

1. T1 线程从共享的内存地址读取A；
2. T1 线程被抢占，线程 T2 开始运行；
3. T2 线程将共享的内存地址中的值由A修改成B，然后又修改回A；
4. T1 线程继续执行，读取共享的内存地址中的值仍为A，认为没有改变然后继续执行。
   如果同步机制通过值相同来判断“没有改变”，如CAS，就可能产生错误。因为在读两次值期间，其它线程可能执行了，甚至其它线程修改了第一个线程的运行假设，第一个线程被欺骗，以为“什么都没发生”，继续以旧的假设运行，这样就会造成错误。

4.4 Memory barrier

内存栅栏也叫内存屏障，是一类同步屏障指令，是CPU或编译器在对内存随机访问的操作中的一个同步点，使得此点之前的所有读写操作都执行后才可以开始执行此点之后的操作。

大多数现代计算机为了提高性能而采取乱序执行，这使得内存屏障成为必须。语义上，内存屏障之前的所有写操作都要写入内存；内存屏障之后的读操作都可以获得同步屏障之前的写操作的结果。因此，对于敏感的程序块，写操作之后、读操作之前可以插入内存屏障。

---

5 是lock-free 还是 wait-free?

在lock-free程序中，任何特定的线程可能会被其他线程阻塞，当给定线程被挂起时，其绝对不会阻止其他线程继续运行。CPUs可以继续执行其它线程中。那么lock-free算法提高系统的整体吞吐量，并且仅仅只增加特定事务的延时。

Wait-free算法确保CPUs持续做有用的工作，明确保证没有线程会被另一个线程阻塞[5]。相对于lock-free，wait-free算法更强力地保证高吞吐量， Linux内核的lockless page cache就是一个 wait-free例子。

我们回过头来看Mars中的设计，Mars中多线程读写的是共享内存吗？虽然名字上是共享内存，整个内存对于任一线程都是可存取的，但是每个线程只会读写属于自己的局部内存区，任何一个线程都不会被其他线程阻塞。是不是更应该是一个wait-free算法？

[1] https://developer.nvidia.com/digits

[2] http://preshing.com/20120612/an-introduction-to-lock-free-programming/

[3] Wenbin Fang, Bingsheng He, Qiong Luo, Naga K. Govindaraju: Mars: Accelerating MapReduce with Graphics Processors. IEEE Trans. Parallel Distrib. Syst. 22(4): 608-620 (2011)

[4] CUDA—Tutorial 5—Performance of Atomics. http://supercomputingblog.com/cuda/cuda-tutorial-5-performance-of-atomics

[5] Alistarh, Dan, Keren Censor-Hillel, and Nir Shavit. “Are lock-free concurrent algorithms practically wait-free?.” Proceedings of the 46th Annual ACM Symposium on Theory of Computing. ACM, 2014.

[6] https://en.wikipedia.org/wiki/Prefix_sum

[7] https://en.wikipedia.org/wiki/Memory_barrier

[8] https://en.wikipedia.org/wiki/Prefix_sum

在做分布式作业时，我们组想用分布式系统来实现一个图形学应用。图形学有很多需要并行计算的地方，比如渲染…

抢沙发

优质评论可以帮助作者获得更高权重

评论

浅析GPU计算——cuda编程 方亮的专栏 3万+

在《浅析GPU计算——CPU和GPU的选择》一文中，我们分析了在遇到什么瓶颈时需要考虑使用GPU去进行计…

GPU高效通信算法-Ring Allreduce 李博Garvin的专栏 7873

今天介绍一种新的GPU多卡计算的通信优化算法—Ring Allreduce。先来讲一下常规的GPU多卡分布式计算的原理。…

【CUDA开发】 CUDA Thrust 规约求和 - ZhangPY的专栏 - CSDN博客 10-30

CUDA编程——Mars:MapReduce on GPU 3134 CUDA编程——Mars:MapReduce on GPUGPU加速机器学习 GPU…

GPU编程 - xmdxcsj的专栏 - CSDN博客 10-30

cuda是英伟达公司的并行计算平台和编程模型,利用GPU加速计算。 linux使用的命令 nvcc:CUDA编译器驱动程序,类…

CUDA系列学习（五）GPU基础算法: Reduce, Scan, Histogram Rachel Zhang的专栏 3万+

喵~不知不觉到了CUDA系列学习第五讲，前几讲中我们主要介绍了基础GPU中的软硬件结构，内存管理，task类型…

摩尔的预言 唯有CUDA才是终极的CPU eloudy的专栏 1270

作者：小熊在线-宁道奇 . 标题：一二三四五六七八九十一二三四五六七八九十 标题：摩尔的预言 唯有CUDA才是终…

GPU通用计算调研报告_数学屌丝走在it的道路上 12-31

GPU在这十多年的演变过程中,我们看到GPU从最初帮助CPU分担几何吞吐量,到Shader(着色器)单元初具规模,然后…

GPU通用计算调研报告_痞子龙3D编程 2-1

GPU在这十多年的演变过程中,我们看到GPU从最初帮助CPU分担几何吞吐量,到Shader(着色器)单元初具规模,然后…

Mars: A MapReduce Framework on Graphics Processors Dedug调试 qq632544991p的专栏 529

Mars是一个运行在gpu上的map/reduce框架，笔者最近最近想使用这个框架来做一些图处理的工作。 然而mars运行…

CUDA学习——CUDA代码常用编写技巧(转) 渐行渐远 2893

1. 声明 __shared__ 变量或数组:__shared__ float sh_farr[256];__shared__ int a;2.结构体指针成员的分配设备内存:…

GPU上实现Map Reduce的好文章 03-12

GPU上实现Map-Reduce的框架 可以进一步实现各种机器学习

MPI和MapReduce对比 oraclestudyroad的博客 1579

Hadoop认证教程：MPI和MapReduce对比，在当前最流行的高性能并行体系结构中比较常用的并行编程环境分为两…

搞搞GPU 非●梵高 671

昨儿intel给拿过来一台服务器，想

CUDA——GPU并行计算框架 Q1368089323的博客 54

简介 CUDA（Compute Unified Device Architecture），是显卡厂商NVIDIA推出的运算平台。 CUDA™是一种由NVI…

CUDA 编程 之 invalid device function huyumars的专栏 1326

最近使用cuda编程经常遇到 invalid device function错误 核函数不能

Hadoop+GPU强强联手的性能探索 数学屌丝走在it的道路上 1984

Hadoop并行处理可以成倍地提高性能。现在的问题是如果将一部分计算工作从CPU迁移到GPU会怎么样？能否更快…

数据流技术在GPU和大数据处理中的应用 weixin_45585364的博客 339

点击上方蓝字关注我们 数据流技术在GPU和大数据处理中的应用苏华友,梅松竹,李荣春,窦勇国防科技大学计算机学…

GPU通用计算调研报告 weixin_33814685的博客 65

摘要：NVIDIA公司在1999年发布GeForce256时首先提出GPU（图形处理器）的概念，随后大量复杂的应用需求促…

五 浅谈CPU 并行编程和 GPU 并行编程的区别 weixin_30922589的博客 78

前言 CPU 的并行编程技术，也是高性能计算中的热点，也是今后要努力学习的方向。那么它和 GPU 并行编程…

【CUDA自带实例学习】2.锁页内存 dgh_dean的博客 848

#include "cuda_runtime.h" #include "device_launch_parameters.h" #include #include using namespace std; #defin…

ZhangJunior
码龄5年 暂无认证

12	21万+	9万+	14万+	
原创	周排名	总排名	访问	等级

1089	61	79	39	114
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

分类专栏

最新评论

CUDA编程——GPU架构，由sp，sm，t...
YaoJiawei329: 少了一张关键的图，能不能麻烦博主补上

缺少Python27_d.lib的解决方法
鸟儿先生: 非常感谢

CUDA编程——GPU架构，由sp，sm，t...
YaoJiawei329: 楼主我想请教一下，980显卡一个SM中是不是包括128个SP

CUDA编程——GPU架构，由sp，sm，t...
Elylicery 回复 HHCC001: SP应该是streaming processor

CUDA编程——GPU架构，由sp，sm，t...
Elylicery: 您好，想问一个cuda c行一个thread吗？还是可以执行

最新文章

缺少Python27_d.lib的解决方法

CUDA编程中 extern "c"用法解析

从FCN到DeepLab

2016年 12篇

目录