

# CPU+多 GPU 异构协同计算的体系结构研究

李龙飞, 贺占庄, 徐丹妮

(西安微电子技术研究所, 陕西 西安 710054)

**摘 要:** 以 CUDA 架构为例, 对传统的 CPU+单 GPU 架构进行了分析, 提出了一种 CPU+多 GPU 异构协同计算的系统方案, 对关键的 CPU 对多 GPU 的管理及多 GPU 间数据通信等问题做了重点讨论, 从理论上进行了可行性分析, 并提出了相应的优化方法。

**关键词:** 图形处理器; 统一计算设备架构; 多 GPU; 异构协同计算

**中图分类号:** TN402

**文献标识码:** A

**文章编号:** 1000-7180(2014)03-0098-04

## Research of Architecture of CPU + Multi-GPU Heterogeneous Collaborative Computing

LI Long-fei, HE Zhan-zhuang, XU Dan-ni

(Xi'an Microelectronics Technology Institute, Xi'an 710054, China)

**Abstract:** In this paper, we analyzed the traditional CPU + single GPU architecture, and put forward a CPU + multi-GPU heterogeneous collaborative computing system. We focused on the CPU management for multi-GPU and data communication between multi-GPU, then theoretically analyzed the feasibility, finally gave the corresponding optimization methods.

**Key words:** GPU; CUDA; Multi-GPU; heterogeneous collaborative computing

### 1 引言

近些年来, GPU 的发展速度远远高于 CPU 的发展速度, 相对于 CPU 来说, GPU 具有更高的计算能力和带宽. CUDA (Compute Unified Device Architecture) 是一种基于新的并行编程模型和指令集架构的通用计算架构, 是一个完整的软硬件体系<sup>[1]</sup>. 随着微电子技术的不断发展, CUDA 的功能也在不断完善. 与 CPU 相比, CUDA 能够利用 Nvidia 公司的 GPU 更高效地解决许多复杂计算任务.

由于单个 GPU 对于通用计算性能的巨大提升, 越来越多的人开始把注意力放在了多 GPU 通用计算的研究上. 相对于 CPU+单 GPU 架构, CPU+多 GPU 架构会有更高的计算能力和更大的存储带宽, 但如何实现 CPU 对多 GPU 的管理, 以及多 GPU 设备间如何实现数据通信, 都成为了在该模式

下需要更深入研究的问题.

本文以 CUDA 架构为例, 提出了一种 CPU+多 GPU 异构协同计算的系统方案, 并给出了若干优化方法.

### 2 CUDA 体系架构

#### 2.1 CUDA 编程模型

在 CUDA 编程模型中, 根据功能的不同, 把 CPU 作为主机 (Host), GPU 作为设备 (Device). 在一个系统中, 设备可以有若干个, 而主机只有一个. 根据 GPU 数目的不同, 可以分为单 GPU 架构和多 GPU 架构. 这里首先介绍传统的单 GPU 架构.

在 CUDA 架构中, CPU 和 GPU 上运行的程序共同组成了一个完整的 CUDA 程序. 如图 1 所示, 主机端的程序主要由串行的处理步骤组成, 而设备端的 Kernel 函数为并行的处理过程. 从图 1 中也可

收稿日期: 2013-05-12; 修回日期: 2013-06-30

基金项目: 国产化指挥平台应用技术研究 (204030202)

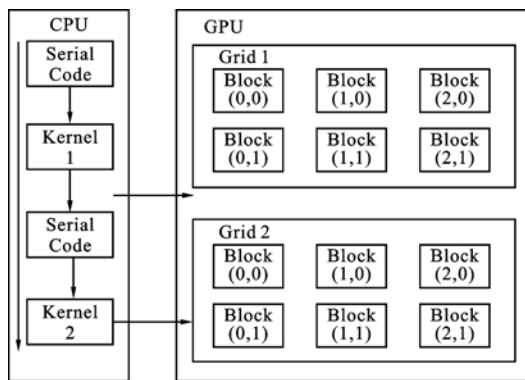


图1 CUDA编程模型

可以看出,Kernel函数的并行可以分为两个层次,分别为Grid中的Block间并行和Block中的线程间并行<sup>[2]</sup>。不同层次上的并行,是CUDA的一大特点。

CUDA同时也指定了存储器模型。如图2所示,每个线程都会有一个本地存储器和一个私有的寄存器组。存储器模型中还有两个只读地址空间,分别是常量存储器和纹理存储器,常量存储器的存储量相对较小(64 kB),被用来存储随机接入的数据,然而纹理存储器的存储量非常大。全局存储器可由GPU或CPU控制,并可对其任意地进行读写操作。

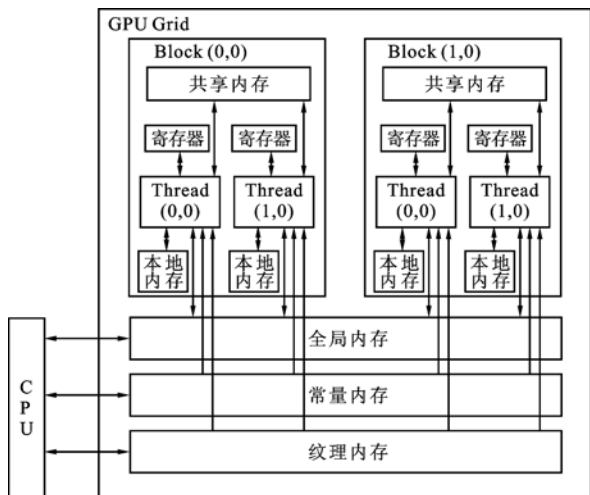


图2 CUDA存储器模型

相对于传统GPU来说,CUDA可以实现线程间的通信,这是传统GPU所不能达到的。为了克服传统GPU的这一不足,CUDA中属于同一Block的线程不仅能够并行执行,而且能够通过共享存储器和栅栏同步实现Block内的线程间通信<sup>[3]</sup>。采用这样的方式,就实现了同一Block内不同线程之间的通信,这称之为细粒度并行;而同一Grid内的不同Block之间的并行,称之为粗粒度并行。这两种不同层次的并行机制,将线程按照两种层次进行组织,在较低的层次实现线程通信,这是CUDA的关键

特性。

## 2.2 运行模式

基于GPU的通用计算,无论是哪一种架构,都遵循着同一个计算框架,那就是CPU+GPU异构协同计算。因此,在该环境中运行的程序就可以分为两个部分,分别为在CPU上运行的程序和GPU上运行的程序。正如前文提到的,因为CPU在异构协同计算中主要完成复杂逻辑的处理和对事务管理的计算,因此GPU上运行的程序都是由CPU来分配和管理的。这样一来,我们称运行在CPU上的程序为宿主程序(Host),称运行在GPU上的程序为核心程序(Kernel)。由于核心程序需要按宿主程序分配的任务来工作,因此其不能自我启动和输入输出,必须依靠宿主端进行环境设置和数据、代码载入等初始化工作,所以,GPU只能作为CPU的协处理器。虽然是协处理器,但是GPU要完成大量的并行数据处理工作。根据算法步骤划分和数据交换的需求,将运行模式分为同步和异步两种方式。这里主要讨论同步运行方式。具体来说,同步运行的整个计算过程可以分为三个步骤:(1)宿主程序把数据从内存复制到GPU的外部存储器;(2)核心程序读取数据,开始运算;(3)宿主程序再把数据从GPU的外部存储器复制回内存。

## 3 多GPU体系结构

多GPU计算模式目前主要有CPU+多GPU异构计算模式和GPU集群计算模式。GPU集群类似CPU集群,是将多台CPU+GPU的计算机组成GPU集群<sup>[4]</sup>。本文主要讨论CPU+多GPU异构协同计算的体系结构。

### 3.1 体系结构

与CUDA架构类似,CPU+多GPU架构也由主机CPU和协处理器GPU组成,不同的只是这里有多GPU。当多个GPU挂载到总线上后,总线的传输压力增大,这直接会引起数据传输延时增大。数据传输延时的问题在整个系统中是一个关键的问题,特别是对那些要在CPU和GPU之间频繁进行数据交换的算法,这个问题的影响将会更为严重。Dana Schaa与David Kaeli对共享总线结构的单GPU和多GPU的传输速率做过测试,结果表明在某些情况下,单GPU的吞吐率甚至会高于多GPU<sup>[5]</sup>。

图3所示为多GPU如何共享总线的系统架构。本文以两个GPU为例,如图3所示,在两个

GPU 与总线的接口处设置了一个开关,这个开关用来控制当前状态下哪一个 GPU 可以使用总线的 16 个通道,或者是在这两个 GPU 之间分配这 16 个通道。

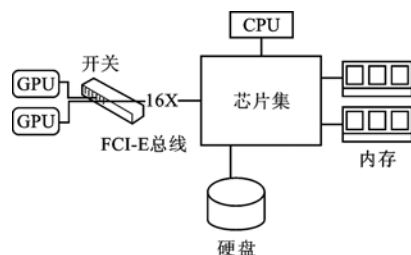


图 3 多 GPU 系统架构

图 3 所示的开关部分是由相应的算法来控制的,在整个系统架构中占据非常重要的地位。在 CUDA 体系中,除了直接使用操作系统提供的 API 管理多设备外,还可以把 CUDA 和 OpenMP 一起使用。合理地分配 GPU 及硬件资源,是提高系统计算能力、效率和降低功耗的关键因素。

### 3.2 多 GPU 间数据通信

根据 CUDA 架构,一个宿主线程只能在一个 GPU 设备上执行设备代码,因此,宿主线程与 GPU 设备是唯一对应的。一个宿主线程在执行时所创建的 CUDA 资源无法直接被其他宿主线程所使用,因此多个 GPU 要进行数据通信时,则需借助内存进行数据传输,也就是必须先将数据传送到内存上,再从内存传送到另一块 GPU,即按照 GPU-CPU-GPU 的顺序进行<sup>[6]</sup>。

在多 GPU 间数据通信的过程中,还涉及线程的同步问题。当多个线程同时读写内存资源的时候,可能会引起冲突。这时候,需要引入线程同步机制,即各线程之间要有个先来后到,按顺序依次进行操作。相关的同步机制主要有同步锁、信号量模型等。同步锁的主要思想是在共享资源上加一把“锁”,哪个线程获得了“钥匙”,才有权限访问该共享资源;信号量模型的主要工作方式是线程在运行的过程中,可以主动停下来,等待某个信号量的通知,这时候,该线程就进入到该信号量的等待队列当中,直到等到通知之后,再继续运行。因此,采用线程同步机制,实现了多 GPU 间数据通信的同步控制,保证了多 GPU 间数据通信的正常进行。

## 4 系统优化

### 4.1 任务分配方法

数据的相关性是左右 GPU 执行效率的关键因

素。在多 GPU 系统中,每个 GPU 上运行的数据相互独立,这样 GPU 的运算效率才能达到最大化,否则,在 GPU 上可能会出现等待或冲突等现象,这将会严重影响到整个系统的运算效率。

因此按照算法确定数据和任务的划分方式成为了 CUDA 编程中的关键部分。具体说来,在任务的分配上,应当将有数据相关性的任务尽量分配在同一个 GPU 上运行。除此之外,当任务在调度时,出现了负载不均衡,就应当考虑将较忙的 GPU 上与其他任务最不相关的任务迁移,以达到数据的冲突量小。

对 GPU 而言,在不考虑数据相关性的情况下,平均分配处理数据是最合理、最高效的。不过与 CPU 不同的是,GPU 是以 Block 为单位处理数据的。这里假设数据总量为  $M$ ,GPU 的数目为  $N$ ,Block 大小为  $B$ ,在数据处理时,一般令  $M$  是  $B$  的整数倍,这样一来,每个 GPU 上处理的 Block 数  $b = (M/B)/N$ ,当然这是理想的能除尽的情况。式(1)给出了通用计算公式:

$$b_i = (M/B) \times i/N - (M/B) \times (i-1)/N, 1 \leq i \leq N \quad (1)$$

由上式可看出,无论能否除尽,第  $i$  个 GPU 处理的 Block 数为  $b_i$ ,式中  $(M/B) \times (i-1)/N$  即为第  $i$  个 GPU 所要处理数据的偏移量。

### 4.2 固定内存与重叠方法

因为多 GPU 间的数据通信要通过内存来实现,因此内存会不定期的、反复的被使用。每一次 GPU 间的数据通信,系统都会给其分配内存,直到数据通信完成,再把分配的内存释放。这固然不是一个好方法。因为系统内存是有限的,并不能保证每一次都能在第一时间给 GPU 的数据通信分配出足够的内存空间;其次,内存的动态分配会耗费很多的资源,尤其是反复的申请、释放内存,耗费的资源更大。

因此,在多 GPU 系统中,可以采用 CUDA MallocHost 函数,在主内存中固定一块内存。所谓固定内存,就是指在内存中固定一块大小的空间,只为 GPU 与 CPU 数据通信时所使用。采用这种方法的好处是避免了内存的反复申请、释放,加快了读写速度,尤其是在 GPU 间数据通信频繁的任务中,更能体现出固定内存的必要性。不过如果固定太大空间的内存,将会影响到操作系统对内存的管理,可能会降低系统的效率。

图 4 分别示出了数据并行通信方式时序图和计算-通信重叠方式时序图。如图 4(a) 所示,以两个

GPU 并行为例,根据数据的相关性,将任务  $T_1$  拆分为  $T_{1-1}$  和  $T_{1-2}$  两部分,分别在 GPU-0 和 GPU-1 两个设备上执行.当并行执行到任务  $T_3$  后,两个 GPU 间要进行数据通信,GPU 暂停工作,直到数据交换完成, $T_4$  开始继续执行.很明显,在  $t_{TX}$  这一断时间内,两个 GPU 都处于空闲状态.

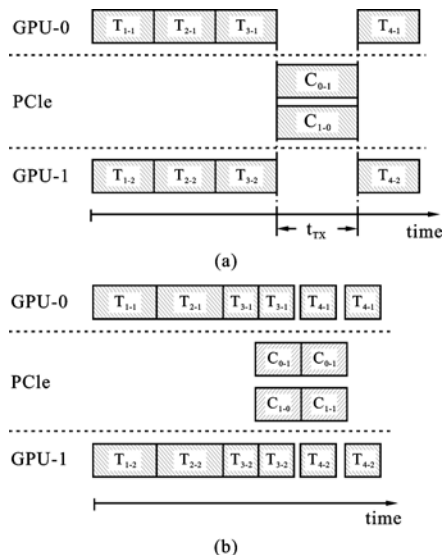


图4 数据通信时序图

采用计算—通信重叠方法,就可以将数据通信所花费的时间最小化.具体说来,如图4(b)所示,将  $T_3$  任务再分为两部分,当第一部分执行完成时,开始这一部分的数据通信,并继续执行第二部分;任务  $T_4$  同样被分为两部分,这样一来,当  $T_4$  第一部分任务要开始执行时,所需数据已经传送完成,PCI-E 总线上开始传送第二部分所需要的数据.

在图4(b)中,由于计算时间比数据通信时间短,因此 GPU 还是会有一定的空闲时间,但是与(a)相比,空闲时间已经小了很多.所以,最好的情况是当计算时间大于通信时间,这样一来,通信时间就会完全被隐藏.

表1中给出了在相同的硬件条件下,执行 CFD (Computational Fluid Dynamics, 计算流体力学) 应用时,数据并行模式与计算—通信重叠模式的时间对比.可以看出,采用计算—通信重叠方式,不仅减少了总时间,而且平衡了计算与通信时间,这样使 GPU 和内存保持在了一个平稳的状态,提高了系统的稳定性.

表1 数据并行与计算—通信重叠执行时间

| 模式   | 计算时间    |     | 通信时间    |     | 总时间      |
|------|---------|-----|---------|-----|----------|
| 数据并行 | 43.0 ms | 38% | 71.5 ms | 62% | 114.5 ms |
| 重叠模式 | 47.6 ms | 52% | 44.5 ms | 48% | 92.1 ms  |

## 5 结束语

本文提出了一种 CPU+多 GPU 异构协同计算的系统方案.在 CPU+多 GPU 系统中,CPU 对多 GPU 的管理模式,以及多 GPU 设备间的数据通信是该系统架构下需要重点研究的问题,本文提出了相应的解决方案,但是要具体解决这些问题,还需要更深入的研究,这也是未来工作的主要研究方向.

## 参考文献:

- [1] Nvidia. NVIDIA CUDA Compute Unified Device Architecture—Programming Guide[EB/OL]. [2013-04-20]. <http://developer.download.nvidia.com/compute/cuda,2008>.
- [2] 张舒,褚艳利. GPU 高性能计算之 CUDA[M]. 北京:中国水利水电出版社,2009.
- [3] DR David Kanter. NVIDIA's GT200: Inside a Parallel Processor[R]. Real world technologies,2008.
- [4] Fan Z, Qiu F, Kaufman A, et al. GPU Cluster for High Performance Computing[C]//Proceedings of the IEEE International Conference on Cluster Computing and Workshops. New Orleans, LA: IEEE,2009:1-8.
- [5] Schaa D, Kaeli D. Exploring the multiple-GPU design space[C]//Proceedings of the IEEE International Symposium on Parallel Distributed. Rome: IEEE, 2009:1-12.
- [6] Ryoo S, Rodrigues C, Stone S, et al. Program optimization study on a 128-core GPU[C]//The First Workshop on General Purpose Processing on Graphics Processing Units. [s. l.]: IEEE,2007:30-39.

## 作者简介:

李龙飞 男,(1988—),硕士研究生,CCF 学生会员.研究方向为计算机系统结构及其应用.

贺占庄 男,(1962—),博士,研究员,博士生导师.研究方向为计算机系统结构、SoC 和 IP 核设计.

徐丹妮 女,(1989—),硕士研究生.研究方向为计算机应用技术.