

硕 士 研 究 生 读 书 报 告



题目 基于GPU的并行计算设计与分析

作者姓名 黄鑫

作者学号 21551047

指导教师 李启雷

学科专业 软件工程

所在学院 软件学院

提交日期 二○ 15 年 12 月

Design and Analysis of Parallel Computation Based on GPU

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Qilei Lee

By

Xin Huang

Zhejiang University, P.R. China

2009

摘要

本文探讨了如何通过GPU来实现并行计算，从而解决高性能计算机计算的问题。在现如今GPU的并行计算经过了很多年的发展，而且GPU设备的性能以及并行度也得到了很大的提高。由于GPU并行处理特性的优势，可以利用GPU的并行计算的技术来加速计算的速度，其广泛应用在图像处理、信号处理、全息计算等领域。OpenCL提供了统一的对不同并行计算平台的编程接口，通过使用OpenCL提供的接口，我们可以就利用GPU实现并行计算。混合OpenCL的技术可以提供跨平台的分布式的并行计算。本篇阐述了OpenCL以及混合OpenCL的设计与实现，并且分析了OpenCL利用GPU加速的性能。

**关键词**：GPU，OpenCL，混合OpenCL，并行计算

Abstract

This paper will introduce how to implement parallel computation based on GPU to solve the problem of high performance computation on computers. Nowadays, the parallel computation on GPU has been developed for many years. And the performance and parallelism of GPU devices have been greatly improved. Based on advantages of GPU parallel characteristic, the parallel acceleration technique based on GPU can be used to improve the performance of computation. It’s commonly used to areas like image processing, signal processing and holographic computation. OpenCL provides unified programming interface for various parallel computing platforms. The parallel computation on GPU can be implemented with the interfaces provided by OpenCL. The hybrid OpenCL can provide more various distributed and parallel computing platforms. In this paper, we introduce the design and implementation of OpenCL and hybrid OpenCL, and analyze the performance of OpenCL acceleration on GPU.

**Keywords**: GPU, OpenCL, hybrid OpenCL, parallel computation

1 引言

为了尽可能地利用并行处理器的性能，硬件厂商会提供特别地并行编程模型，比如NVIDIA GPU的CUDA，不同的设备有不同的编程模型。但是不同的编程模型提高了开发人员的编程难度，因此为了统一不同处理器的编程接口，Khronos组织提出了OpenCL标准。OpenCL是一个跨平台的计算框架，它统一了不同处理器的编程接口。一个OpenCL程序可以运行在不同平台设备上并且不需要修改任何程序。目前的流处理器平台支持OpenCL的编程，比如GPU、CPU、单元处理器、FPGA等。OpenCL的编译器会将程序编译到每个支持OpenCL的设备中。

混合OpenCL将OpenCL的设备连接到一个网络中，它将OpenCL的环境扩大了一个级别。通过使用混合OpenCL，运行OpenCL的程序能够在高性能云计算机上运行，从而提供分布式的计算。混合OpenCL包括了两个元素，一个提供了OpenCL实现的抽象以及一个连接不同运行OpenCL 的程序。

**2 OpenCL基础知识**

**2.1 OpenCL标准**

OpenCL标准由Khronos联盟制定，Khronos联盟所开发的OpenCL是一套适用于不同硬件架构且获得高性能的通用API，这意味着我们可以轻松地运用OpenCL API来发挥不同硬件平台并行处理的能力

OpenCL API的主机部分代码是由C和C++封装而成，而内核代码是用C99语言编写。

**2.2 OpenCL规范**

OpenCL的规范分作4个模型，分别是平台模型、执行模型、内存模型、编程模型。

1. 平台模型：定义了一个能够并行执行OpenCL代码的处理器(设备)的抽象硬件模型，供编程人员编写能够在这些设备上执行的OpenCL C(kernel)函数
2. 执行模型：定义了主机如何配置OpenCL环境以及如何在设备上执行kernel函数，提供了主机和设备的交互机制，提供了在设备上执行kernel的并发模型。
3. 内存模型：定义了kernel所用的抽象内存层次(memory hierarchy)，即主机和设备上的内存结构以及内存数据的传送等。
4. 编程模型：定义了如何将算法的并发模型映射到设备的物理硬件上。

**2.3 OpenCL平台模型**

OpenCL中有描述执行OpenCL代码的平台抽象出的对象，平台可以看成是某个厂商特定得OpenCL API的实现，即选用某个公司的平台后没办法与另外一个公司的平台上的设备进行通信，在OpenCL API中，用cl\_platform\_id表示设备的平台id号，利用这个id号，就可以调用OpenCL API相应地找到是用哪个平台执行OpenCL代码。

平台模型是一个抽象的设备架构，开发者可以直接以平台为目标编写，而厂商则负责将抽象架构映射到具体的硬件。平台模型还将一个设备定义为一系列的计算单元(Compute Unit)，每个计算单元都是独立工作的，计算单元又进一步划分成处理部件(Processing Unit)，图2-1为层次化的平台模型：

平台

设备

计算单元

PU

PU

计算单元

PU

PU

设备

计算单元

PU

PU

计算单元

PU

PU

图2-1 层次化的平台模型

**2.4 OpenCL执行模型**

OpenCL执行模型分为两个部分：第一部分是执行在一个或者多个设备上的内核kernel函数，第二部分是执行在主机上的那一部分代码。

kernel函数

在OpenCL中实际执行并行计算的那一部分代码称为kernel函数，kernel函数由C99标准编写，但其代码的区别在于kernel函数有自己的一个关键字集和OpenCL kernel实现的执行模型。

在OpenCL执行模型中，kernel是被OpenCL中的多个work-item并发执行，也就是说work-item就是OpenCL执行模型中系统执行的最小单位，在计算单元上运行的kernel程序的实例就是一个work-item，实际上一个work-item就是映射物理硬件上的一个处理单元(Process Unit)。

开发者可以通过一个n维索引空间(NDRange)来声明新建的work-item的数量，一个NDRange可以是work-item的一维、二维、3维甚至是多维的索引空间，这个索引空间就被映射到work-item从kernel函数获取的输入数据或者是输出数据的索引空间里。

此外OpenCL中还有一个work-group的概念，与work-item一样，work-group同样具有N个维度，一个work-group包含多个work-item，具体的数值通过程序去指定，同时不能超过硬件查询返回的最大的work-item数目。实际上一个work-group映射着物理硬件的一个计算单元(Compute Unit)。

通常一个N核心的CPU会有N个计算单元，而每个计算单元又包含着M个处理单元。

在程序中，Work-item、Work-group的大小可以通过程序去指定，在程序中会要求指定global\_work\_size、local\_work\_size变量的大小，间接地就确定了Work-item、Work-group的大小。

global\_work\_size是一个N维空间的数组，用于指定在各个维度下需要总共用多少个work-item去并行执行一个kernel。global\_work\_size也是一个N维空间的数组，用于指定在各个维度下一个work\_group的Work-item的数量。由于程序指定的global\_work\_size可能会大于实际硬件的Work-item的数量，于是多余的部分就会被分成许多个work-group去执行，为了程序的效率考虑，通常global\_work\_size为实际硬件work-item数量的整数倍。

local\_work\_size就是一个work\_group能容纳的最大的work\_item的大小，该大小不能超过实际硬件给定的最大的work\_item的大小，而且从性能上来说最好为实际硬件的最大work-item数量，这样可以尽可能地发挥一个CU的全部效能。

从以上分析可以得知，对于 Global\_work\_size为和一个Local\_work\_size为计算出得 Work-Group大小为：

而Work-item的大小为:

但实际上Work-group的大小还受到硬件的控制，不能超过实际硬件给定的最大的Work-group的大小。

在OpenCL内核实际的运行中，Global ID唯一地标识了一个work\_item，global ID是一个N维数组，从(0,0,...0)开始，直到(global\_work\_size[0], global\_work\_size[1],…, global\_work\_size[N-1])

同样，Local ID唯一地标识了work\_item在其work\_group中的位置，local ID是一个N维数组，从(0,0,...0)开始，直到(local\_work\_size[0], local\_work\_size[1],…, local\_work\_size[N-1])

Group ID唯一地标识了work\_item所在的work\_group，group ID是一个N维数组。

从上述定义可知，一个work\_item可用两种方式标识，一种是通过global ID标识，第二种是通过所在组数去乘以一个组所含的work\_item数再加上local ID。

也就是有如下关系：

(2-1)

以二维寻址空间为例，假设输入的work-item数为(Gx,Gy)，一个group的work\_item数为(Sx,Sy)，该work-item的global-ID为(gx,gy)，local-ID为(lx,ly)，其所在group的偏移量为(ox,oy)，则有：

(2-2)

那么组数(Wx,Wy)为：

(2-3)

图2-2为kernel输入数据寻址模型图：

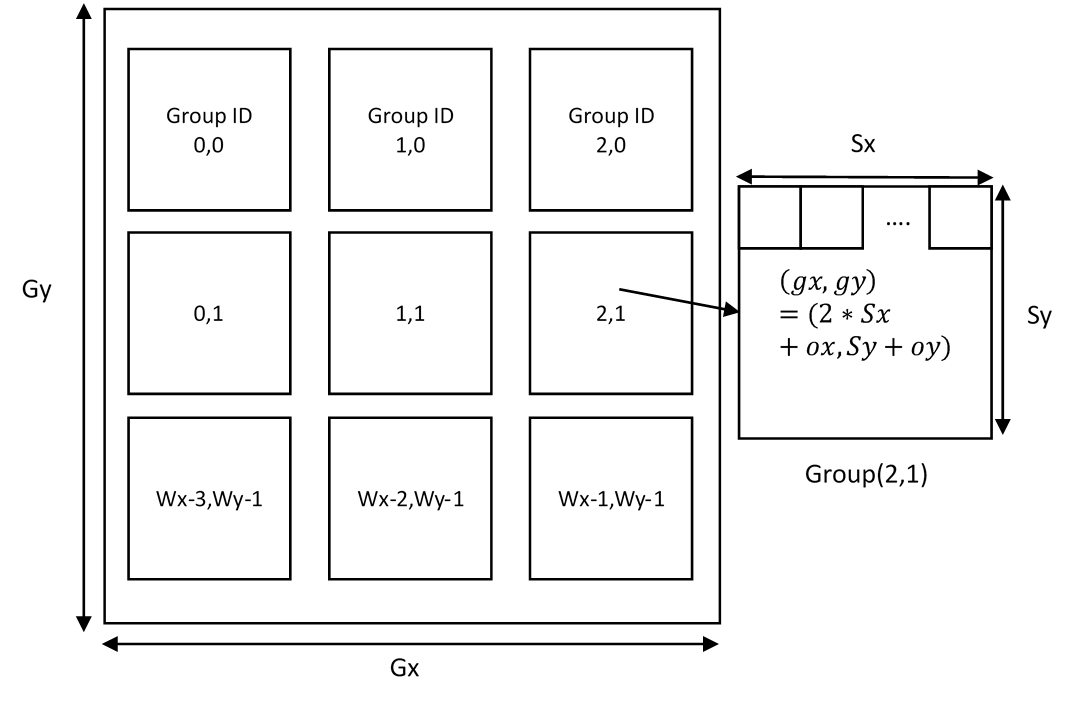


图2-2 kernel数据寻址模型

OpenCL允许有多个编程模型映射到该模型中。

**2.5 主机host执行模型**

主机是通过一个context对象用于执行内核函数，一个context对象包含如下4个资源：

1. 设备devices：host所用的设备
2. 内核kernels：被host设备所执行的OpenCL函数
3. 程序对象programs：运行kernel内核的程序
4. 内存对象memories：能够同时被kernel与主机访问的内存对象

主机host是通过一个命令队列command queue使设备执行内核的函数。主机首先将一个命令command入队命令队列，之后context中的设备就会执行命令。这些命令包括：

1. 内核执行命令：用于设备执行指定的内核程序
2. 内存命令：用于将主机与设备的内存交互，以及内存地址映射等。
3. 同步命令：用于命令之间的同步。

命令队列中命令的执行有2种方式：

1. 顺序执行：队列中的命令将会按照命令入队的顺序去执行，命令与命令之间没有关系，不需要同步处理。
2. 无序执行：队列中的命令将不按照入队的顺序执行，命令之间为并发执行，命令之间需要同步处理。

**2.6 OpenCL内存模型**

OpenCL定义了一个抽象的内存模型供开发者利用其模型编写代码，厂商则负责映射该模型到实际的内存硬件中。OpenCL的内存分为全局内存、本地内存、常量内存和私有内存四种内存，以下分别介绍。

1. 全局内存(global memory)

全局内存对于所有work-group中的work-items都可以进行读写，也就是说全局内存对于设备上所有计算单元都是可见的。主机端传送到设备端或是设备端传送到主机端的数据都在全局内存中完成。

在OpenCL中，关键字\_\_global用于指定数据指针所引用的数据驻留在全局内存中。

全局内存对应设备上的专有内存或者是共享内存，例如对于CPU来说，全局内存则是系统的内存，而对于集成显卡的GPU来说，全局内存对应的是GPU的共享内存，而对于独立显卡的GPU来说，全局内存对应的就是GPU的专有内存。

1. 本地内存(local memory)

本地内存只能由一个工作组的所有work-items进行读写，不同工作组的本地内存不同，也就是说本地内存只对设备上相同work-group的work-items所共享，但是不同work-group之间的本地内存无法进行交互。

本地内存映射硬件上的高速暂存存储器(scratchpad memory)，它的地址空间对于计算单元(Compute Unit)来说是独一无二的，通常可以用于相同work-group的work-item之间的数据同步。

相比于全局内存来说，本地内存的访问延迟更短，带宽够高，但缺点在于其大小受到硬件的限制，不能超过硬件给定的计算单元内存最大值。而且本地内存只能用于相同组之间的工作项的本地内存数据交互，不同的工作项之间的本地内存数据无法交互。

1. 常量内存(constant memory)

常量内存是一块在内核程序执行过程中恒定不变的全局内存，它便于所有work-items所访问，它无法写入，仅可以读取，常量内存就是全局内存中的一部分。

1. 私有内存(private memory)

私有内存是各个work-item所私有的内存区域，不同work-item有不同的私有内存，私有内存不能被不同work-item所共享。也就是说私有内存只对单个work-item可见。

对于没有关键字的非指针类型的kernel参数和局部变量默认都是私有内存。

私有内存通常会被映射到寄存器之中，因此私有内存是OpenCL内存交互中效率最高的。

图2-3为OpenCL内存与实际硬件内存的映射关系：



图2-3 OpenCL内存与实际硬件映射模型

**2.7 OpenCL 编程模型**

在OpenCL的执行模型中支持两种编程模型，分别为数据并行编程模型(data parallel programming model)与任务并行编程模型(task parallel programming model)，或者是两种混合的编程模型。

* 数据并行编程模型

数据并行编程模型定义了一系列指令同时应用在多个内存对象元素上的计算。OpenCL定义的寻址空间指定了数据是如何映射到work-item的。在严格的数据并行编程模型中，每一个work-item都必须至少有一个对应的内存对象元素能够在kernel内核下并行执行。当然在OpenCL中并不要求这种一一对应的关系。

OpenCL提供了两种方式供程序开发者并行处理数据，第一种方式开发者可以显式定义work-item的总数以及work-group中work-item的数量划分，也可以通过隐式定义work-item的总数，但不指定如何划分work-group中的work-item数量，而是交给OpenCL来自动管理。

在这种编程模型下，数据间的同步可以通过组间同步函数barrier()来实现。一旦调用barrier()那么就可以保证该work-item所在同一个work-group的所有work-item都已经执行完barrier()之前的函数。

* 任务并行编程模型

OpenCL的任务编程模型是每一个kernel由一个计算单元中的一个work-item去执行，在这种方式下，仅仅只是不同的计算单元之间并行，也就是不同任务之间的并行，不涉及到寻址空间的问题。

**3 混合OpenCL(Hybrid OpenCL)**

仅仅使用OpenCL有几个问题：(1) OpenCL的运行库与一个特定厂商的设备是紧密联系在一起的，因此它不能利用其它厂商的设备。(2) 而且它只能利用单机上连接的本地设备。开发混合OpenCL可以解决上面这两个问题，它运行在有比较大规模的OpenCL环境中，它能够使得OpenCL程序运行在高性能云计算机中。混合OpenCL可以提供更多种类的并行计算平台以及OpenCL程序的利用价值。

混合OpenCL将网络层与OpenCL的运行时连接起来。网络层将多个OpenCL运行时通过网络上的远程主机相连接。OpenCL的运行时加了一层网络层叫做混合OpenCL运行时(Hybrid OpenCL runtime)。在远程主机上，桥接程序与混合OpenCL运行时进行交互并且复制OpenCL应用的API与管理实际的OpenCL资源。

**3.1 混合OpenCL运行时(Hybrid OpenCL Runtime)**

混合OpenCL运行时用于抽象出在远程主机上被OpenCL应用程序使用的OpenCL设备。我们称远程主机上的OpenCL设备叫做远程设备，而程本地主机上的OpenCL设备叫做本地设备。混合OpenCL需要找出拥有在网络上的的OpenCL设备并且列出这些远程主机去管理远程设备。目前，通过枚举主机上的因特网协议地址从而将其写入一个静态的文本文件中。当OpenCL运行时被初始化的时候，这个文本文件将被加载从而运行时就可以配置网络主机。此外，OpenCL运行时将分配资源处理句柄给对应的新的OpenCL资源。因此混合OpenCL运行时必须将由每个桥接程序分发的资源句柄与传给OpenCL应用程序的资源句柄相连接。

**3.2 桥接程序(Bridge Program)**

桥接程序是用于执行来自混合OpenCL运行时的API调用，它可以返回结果给OpenCL运行时.桥接程序需要管理并且保存资源句柄。因为当桥接程序分配资源给OpenCL运行时的时候，资源句柄就已经存在于OpenCL运行时钟了。此外，桥接程序必须管理每一个OpenCL应用程序的连接资源。可以用C++来实现这种复杂的资源管理。

**4 小结**

通过对于基于GPU的OpenCL的研究以及对于混合OpenCL程序的分析，让我对在GPU上作并行计算有了比较明确的认识，大致有以下几点：

（1）首先了解了一种可以在GPU上作并行计算的标准与技术：OpenCL，我了解了OpenCL的标准、规范、平台模型、执行模型、内存模型以及编程模型。知道了OpenCL具体的标准以及规范，并且通过实际调用OpenCL API认识到了OpenCL的编程模型，知道了GPU并行计算的过程：即数据流首先从主机内存流向独立显卡的独显内存，再通过OpenCL的运行kernel在每个GPU的计算单元上运行并行计算程序，最后将并行计算程序的结果写回主存。这一系列的过程OpenCL将一个厂商的所有设备抽象为一个平台(platform)，将一个计算单元抽象为一个工作项(work-item)。并且OpenCL抽象出自己的内存模型以及内存寻址模型，使得主机内存与设备内存之间可以相互交互。

（2）进一步地了解了大规模OpenCL并行计算的混合OpenCL的实现。混合OpenCL通过加入网络层以及桥接程序将不同的运行OpenCL的主机连接起来，混合OpenCL实现了不同厂商的OpenCL设备并行计算的可能。也就是说混合OpenCL不仅可以实现一个GPU设备的并行计算，而且还可以实现不同平台的GPU设备的并行计算，为大规模并行计算缔造了可能。

（3）通过以上对基于GPU的并行计算的研究，提供了实现GPU并行计算的更明确的思路，为更高效地GPU并行计算以及更大规模的GPU并行计算提供了一个不错的思路。