基于 OpenCL 的并行方腔流加速性能分析*

李 森[1], 李新亮[2], 王 龙[1], 陆忠华[1], 迟学斌[1]
(1. 中国科学院 计算机网络信息中心超级计算中心, 北京 100190; 2. 中国科学院 力学研究所, 北京 100190)

摘 要: 提出了一种使用 OpenCL 技术对方腔流问题进行加速计算的方法。在计算方腔流问题时,将其转换为 N-S 方程通过空间有限差分和龙格库塔时间差分求解,并使用局部缓存等技术进行 GPU 优化。实验在 NVIDIA 和 ATI 平台对所给算法进行评测。结果显示,OpenCL 相对其串行版本加速约 30 倍左右。

关键词: 显卡通用计算; 计算流体力学; 方腔流; 有限差分计算
中图分类号: O242 **文献标志码**: A **文章编号**: 1001-3695(2011)04-1401-03
doi: 10.3969/j.issn.1001-3695.2011.04.057

Performance analysis of OpenCL parallel acceleration on cavity flow problem

LI Sen[1], LI Xin-liang[2], WANG Long[1], LU Zhong-hua[1], CHI Xue-bin[1]
(1. *Supercomputing Center, Computer Network Information Center, Chinese Academy of Sciences, Beijing* 100190, *China*; 2. *Institute of Mechanics, Chinese Academy of Sciences, Beijing* 100190, *China*)

Abstract: This paper proposed a new approach to accelerate cavity flow problem using OpenCL technology. This paper transformed the cavity flow problem into N-S equation and solved it by finite difference method. During this procedure, used local memory and other techniques to accelerate the code. This paper ran the program on both NVIDIA and ATI platforms. The experiment shows a 30x times speed up.

Key words: GPGPU(general purpose GPU); computational fluid dynamics; cavity flow; finite difference method

0 引言

近年来,GPU(graphic processing unit)通用计算(GPGPU)异军突起,逐渐成为计算机科学、应用科学的热点研究领域之一。尤其是在高性能计算方向,GPU 通用计算硬件性能的不断完善以及编程复杂性的不断降低,使得越来越多的理论工作者、实验科学家得以利用 GPU 强大的并行计算资源解决因计算速度限制而无法完成的复杂科学计算难题[1,2]。事实上,在 21 世纪初,GPU 作为通用计算处理器已经出现在科学计算领域。但当时其基于图形 API 接口进行设计,编程模式过于困难,缺乏灵活性,存在带宽瓶颈等问题,难以充分发挥 GPU 的计算能力。

英伟达(NVIDIA)公司于 2007 年推出了 GPU 通用计算的并行编程架构(compute unified device architecture, CUDA),令 GPU 通用计算的发展实现了新的飞跃。CUDA 架构包含了 CUDA 指令集(ISA)以及 GPU 内部的并行计算引擎,其采用 C 语言作为编程语言提供大量高性能计算指令开发能力,使得开发者能够在 GPU 强大计算能力的基础上建立一种更高效的密集数据计算解决方案。但 CUDA 架构的一个最主要问题是通用性,人们只能使用 NVIDIA 系列产品对 GPU 进行开发。

为了进一步完善异构架构模型,在 2008 年 6 月 WWDC 大会上,苹果公司提出了 OpenCL 规范,旨在提供一个通用的开放 API,便于开发人员在此基础上更好地开发异构通用计算软件。2008 年 12 月份,Khronos 工作组顺利完成了该标准的定稿工作。而在 2009 年的 6 月和 8 月,NVIDIA 和 AMD 也分别发布了支持 OpenCL 1.0 通用计算规范的驱动程序,使得人们可以真正使用 OpenCL 技术释放 GPU 和 CPU 强大的并行计算能力。OpenCL 允许开发人员把同样的代码移植到任何支持该标准的平台上运行,大大降低了程序移植性的复杂度。而开发人员面临的最重要的挑战之一就是如何充分利用本地多处理器资源对数据进行并行化处理。

本文使用 OpenCL 对方腔流问题进行加速,主要出于以下两个目的: a) 作为新推出的第一款通用计算编程标准,现有通过 OpenCL 进行加速的算例不多,其性能缺乏评判。通过一个经典案例,使用 OpenCL 技术分别在 NVIDIA 和 ATI 卡上进行实现,并将其结果在其他并行编程环境如 Openmp、CUDA 等进行对比,来对该技术性能进行性能评测。通过在不同平台的对比,软件人员可以更好地审视其性能,便于在今后开发中进行平台选择。b) 流体动力学作为一门计算学科,其庞大的计算任务和计算方法是高性能关注的重点方向之一。方腔流作为流体动力学中非常经典的算例,其分析方法在微流动研究中具有相当的普遍性和实用意义。本文也试图提出基于 OpenCL 的方腔流解决方案,为流体中其他问题在 OpenCL 平台的实现提供参考。

1 OpenCL 编程规范

1.1 OpenCL 运行框架和内存模型

OpenCL 运行架构运行时使用主机端程序对所有支持

收稿日期: 2010-08-28; **修回日期**: 2010-10-15 **基金项目**: CNIC 主任基金资助项目(CNIC_ZR_09005)

作者简介: 李森(1984-),男,研究生,主要研究方向为高性能计算、GPU 通用计算(lisen@sccas.cn); 李新亮,男,研究员,主要研究方向为非线性力学; 王龙,男,主要研究方向为大规模数值模拟、GPU 通用计算; 陆忠华,女,博导,主要研究方向为并行计算、高性能计算应用研究; 迟学斌,男,博导,主要研究方向为并行计算、高性能计算.

OpenCL 设备进行统一管理。一个主机程序(host) 管理一个或多个计算设备(compute device) 。对于每个支持 OpenCL 规范的计算设备,其在内部又进一步划分为多个计算单元(compute unit) ,每个单元有多个运行单元(processing element) ,每个运行单元按照 SIMD 或 SPMD 的方式运行。OpenCL 程序运行发生在两个地方: 每个内核函数(kernel) 在一个或多个 OpenCL 计算设备上运行,主机端程序在主机上运行。每个主程序定义 OpenCL 上下文(context) 对核函数运行进行统一管理。

每当主机端提交一个核函数到计算设备时,OpenCL 自动定义一个索引空间。在索引空间上的每一个点,内核函数用一个工作集(work-item) 来表示。通过全局标志符对每个工作集进行定位,就可以清楚地知道自己所需要完成的任务分工。事实上,每个工作集会运行完全相同的代码,但根据标志的位置不同,它们可能完成不同条件分支的代码。在索引空间中,多个工作集会组织成为一个工作组(work-group) ,它提供了对索引空间更粗一级的划分。同样地,每个工作组也有自己的全局索引标志符。在一个工作组中的多个工作集会在不同的运行单元上运行,而在同一个工作组中的工作集在同一个计算单元上运行(图 1) 。图 1 中,整个索引空间被分为多个工作组,每个工作组又进一步划分为多个工作集。

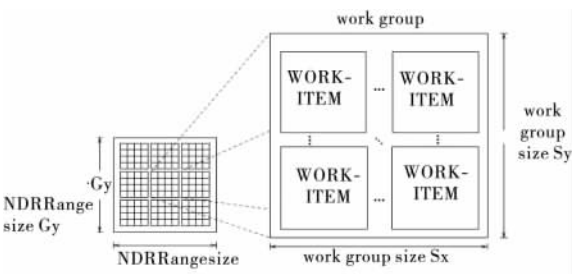


图1 OpenCL索引空间示例

每一个工作集可以访问的内存对象有四种形式:

a) 全局内存(global memory) 。对所有工作集可见,既可读,又可写。

b) 常量内存(constant memory) 。全局内存一部分,在整个运行阶段保持不变。

c) 局部内存(local memory) 。只对一个工作组内的所有工作集可见。

d) 私有内存(private memory) 。只对单独的一个工作集可见(图 2) 。

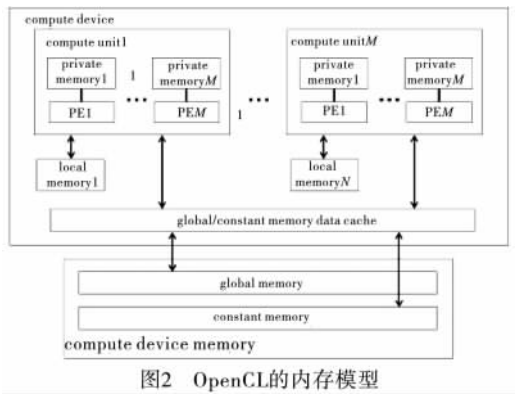


图2 OpenCL的内存模型

工作集对不同范围的存储器进行访问,速度也各不相同。事实上,编程人员可以将数据存放在指定的存储器。内存的显式操作给编程人员带来方便,同样也给编程者带来挑战。内存使用方式是否得当直接决定性能的高低。编程者应灵活地使用存储结构,以达到最优性能。

1.2 支持 OpenCL 的平台

作为 OpenCL 的提出者,苹果公司已经在其 Snow Leopard 产品线中加入了 OpenCL 支持。2009 年 6 月 NVIDIA 首家发布了支持 OpenCL 1.0 通用计算规范的驱动程序,支持 Windows 和 Linux 操作系统。同年 8 月,AMD 发布了可支持 X86 处理器和 ATI 显卡的 OpenCL SDK—ATI Stream SDK v2.0Beta。开发人员可以同时针对这些硬件进行编程与测试。

2 方腔流问题描述

本文使用方腔流作为实验案例,如图 3 所示。有一个正方形腔室,无量纲密度为 ρ,里面充满静止的不可压缩粘性流体。方腔内有初始时刻的压力和密度值。周围壁面固定不动,上壁面以一无量纲初始速度沿上壁面方向运动。

解决该问题时,物理上首先通过动量及能量守恒建立 N-S 方程,表示为

$$\frac{\partial}{\partial t}U+\frac{\partial}{\partial x}f_1+\frac{\partial}{\partial y}f_2=\frac{\partial}{\partial x}V_1+\frac{\partial}{\partial y}V_2 \tag{1}$$

其中: U 是密度、速度、能量组成的向量,f 是无粘性项,V 是粘性项。一般地,N-S 方程属于非线性的微分方程,无法直接通过解析式求解,转而采用数值方法。对方腔流问题,一般地可在时间步上采用 3 阶 Runge-Kutta 方法进行求解,即将式(1) 转换为

$$U^1=U^n+\Delta tL_h(U^n) \tag{2}$$

$$U^2=\frac{3}{4}U^n+\frac{1}{4}[U^1+\Delta tL_h(U^1)] \tag{3}$$

$$U^{n+1}=\frac{1}{3}U^n+\frac{2}{3}[U^2+\Delta tL_h(U^2)] \tag{4}$$

其中: L_h 为空间离散算子。而空间域上对整个计算区域进行网格划分。对于无粘项,通过流通矢量分裂,将原先的无粘项分解为正通量和负通量; 然后再分别针对正、负通量采用相应的差分离散,如式(5) 所示。

$$f_j'=a_1f_{j-4}+a_2f_{j-3}+a_3f_{j-2}+a_4f_{j-1}+a_5f_j+a_6f_{j+1}+a_7f_{j+2}+a_8f_{j+3} \tag{5}$$

其中:

$$a_1=7.14\times10^{-3}\quad a_2=-6.67\times10^{-2}\quad a_3=0.3\quad a_4=-1$$
$$a_5=0.25\quad a_6=0.6\quad a_7=-0.1\quad a_8=-9.52\times10^{-3}$$

与不进行矢量分裂,直接离散对流项(通常采用中心格式或谱方法等) 相比,矢量分裂后采用迎风差分格式具有更好的稳定性,同时能有效抑制混淆误差[3]。

3 基于 OpenCL 的方腔流实现

3.1 主程序

由运行模型可知,OpenCL 程序分为两部分,主程序进行任务调度分配,内核程序负责具体计算。

对上面方腔流问题分析,本文确定主程序的工作流程。在进行实际计算之前,主程序应先建立 OpenCL 平台上下文和指令队列,对设备进行初始化设置。在实际计算过程中,建立变量 U_n 为每步所需计算变量的矢量形式。本文采用式(2) ~(4) 中提到的 3 阶 R-K 方法,每个时间步又分为三次顺序子计算过程,分别计算出 U^1、U^2 和 U^{n+1} 后进行下一步时间迭代。对此 2 维方腔流问题,采用流通矢量分裂的方法进行无粘项的

计算。配制其系数 Jocabian 矩阵,通过其特征值得出正通量和负通量,然后通过 7 阶差分得出式(2) ~(4) 中空间离散算子的影响。对粘性项采用其定义直接通过相关散度计算获得。在以上这些过程中,主程序实际上只负责任务分配,而每一项具体计算过程,如通量分裂、差分计算等都由主程序配制好参数后将待完成的计算任务提交到 OpenCL 指令队列,等待 OpenCL 设备进行计算(图 4) 。主程序建立适当的同步机制以保证计算过程依次完成。

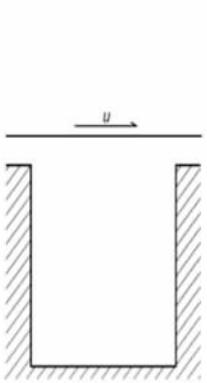


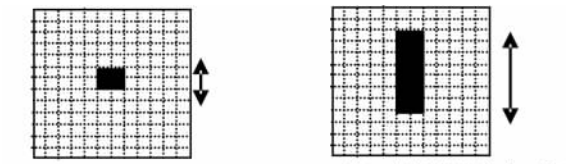
图3　方腔流

图4　程序流程图

对于上述方法,主程序需注意几个问题: a) 由于每个 OpenCL 程序可能管理多个设备,在创建上下文时根据需要进行选择,每个上下文可以关联到一个或多个设备,多个上下文也可以关联到同一设备; b) OpenCL 在设计时充分考虑到移植性,所以实际在编写程序时假定并不知道底层平台到底需要在主程序中动态进行的编译和创建(build) 操作; c) 在 OpenCL 规范里面明确要求,对设备和内核函数的访问都要通过队列来进行。在此,不能通过直接的函数调用进行内存的拷贝操作或启动核函数,而需要将所有这些信息加入到指令队列中(command queue) ,通过 OpenCL API 进行统一调度。事实上使用指令队列有一个非常重要的好处即允许程序员进行任务级并行。举例来讲,不同方向的差分就可以通过队列的乱序执行模式同时进行操作,以达到进一步加速的目的。

3.2　内核函数设计

在程序中涉及的流通矢量分裂、迎风差分计算和 R-K 时间步差分等方法,通过不同内核函数来实现。内核函数的参数由主函数指定后提交到指令队列后由设备内核函数计算完成。本文将整个二维计算网格映射到 OpenCL 全局索引空间。计算网格上每个点映射到一个工作集,从而每个工作集记录并计算网格上该点所有需要计算的信息。分析可知,例如流通矢量分解、R-K 差分等过程,其网格点的信息可以准确地映射到索引空间直接计算,而差分式(5) 的计算则存在一些难点。如式(5) ,在计算一个点时需要其周围多个点的状态信息。这是许多流体计算中非常常见的情况。这种依赖性给划分索引空间尤其是使用局部内存带来了困难。如果将所有信息放在全局内存里所有的工作集都可读可写,由于依赖关系,在更新一个点时会多次反复地读取全局内存。这样频繁地访问全局内存操作必定会带来性能上的损失。由于以上所描述的依赖关系,实际上每个工作组所申请的局部内存比该工作组所需要计算的面积大。对式(5) ,考虑 Y 方向的差分,若一个工作组计算 width × height 大小区域,则申请局部缓存大小为 width × (height + 4 + 3) 大小(图 5) 。X 方向可同理实现。



(a)一个工作组计算网格大小　　(b)一工作组申请的局部内存面积,比计算网格大

图5　一个工作组计算网格的大小及其局部内存大小

4　实验

4.1　实验环境

对 OpenCL 程序的性能对比建立在不同设备的程序运行上。本文将 OpenCL 与传统的串行程序、Openmp 加速后的程序、OpenCL 分别在 NVIDIA 和 AMD 硬件上的设备,以及单纯使用 NVIDIA 进行 CUDA 加速后进行对比,以尽可能评估 OpenCL 在这类问题上的性能。

在串行方面,使用的 CPU 为 Intel® Xeon® 主频为 3.00 GHz,一级缓存 32 KB,二级缓存 6 MB 的硬件环境。串行的 Fortran 程序使用 ifort 编译器进行 level 2 级优化。对应的 Openmp 使用了 4 个核的相同设备。在 GPU 方面,使用 NVIDIA 的显卡型号为 Tesla 1060,显存 4 GB,AMD 显卡方面,使用 9270 进行评测。

在进行方腔流数值实验过程中,通过 R-K 方法在时间步上进行差分,并使用 Steger-Warming 进行流通矢量分裂,然后采用 7 阶迎风差分的方式对所给问题进行求解。在表 1 中显示的是不同规模方腔流问题在不同平台上的时间比较。本文截取了 100 步以进行输出,时间以 s 给出。

表 1　不同规模 OpenCL 方腔流与串行、Openmp、CUDA 的性能对比(运行时间按 s 计)

scale size	serial	Openmp(4 cores)	OpenCL(NVIDIA)	OpenCL(ATI)	CUDA
512 × 512	13.9844	9.406	2.975	7.588	1.34
1024 × 1024	70.83	52.2614	9.213	24.785	4.81
2048 × 2048	281.4307	217.9449	32.651	91.246	17.54
4096 × 4096	3823.8038	1515.912	124.742	XX	66.91
5120 × 5120	6843.9784	2378.9607	196.625	XX	108.3499

4.2　结果分析

从表 1 中的数据可看出,当运算规模较小时,Openmp、OpenCL、CUDA 都不能完全达到性能提升的目的。由阿姆达尔定律(Amdahl's law) 可知,并行系统性能取决于并行与串行部分所占比例。当计算数据规模较小时,计算部分所占用时间与程序总时间的比例较小。程序很多时间花费在数据初始化、线程开销和内存显存传递上面。这种情况并不利于发挥 GPU 大规模计算的优势。而当数据量上升时,实际需要计算的数据达到几个 G,计算部分所占时间比例逐步升高,计算部分也成为程序运行的瓶颈。此时使用 OpenCL 技术进行加速,加速比从原始的 10 倍左右上升到 30 倍左右,这说明 OpenCL 更适合大规模的并行计算要求。

本文使用了相同的内核函数分别在 NVIDIA 和 ATI 平台进行了比较。实验中,OpenCL 在 Tesla 1060 上的运算速度约为 ATI 9270 的 2.55 倍。这可能由以下原因引起: OpenCL 程序整体架构与 CUDA 非常类似。在计算所使用方腔流算例中该模型可以很好地映射到 CUDA 平台。而 ATI 采用流处理器架构,若需要充分发挥其性能应考虑其流处理的性质,更多采用矢量数据类型并进行流优化。在本算例中,(下转第 1421 页)

3.3　软件成分映射关系的映射实现

事务操作映射操作页面如图 5 所示,以收费事务分解为例,分解为五个操作。

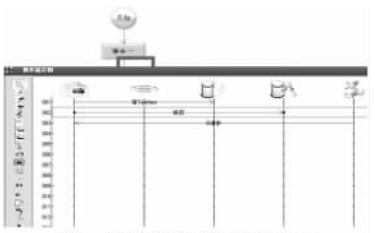


图5　事务操作映射定义界面

1) 页面加载操作　对已经保存到业务数据库的信息进行查询,加载到页面中,把与文档操作有关的信息,利用相关计算,加载到页面相应域中,并对表单数据进行合法性检查。

2) 保存操作　保存页面信息到相应的数据库中。

3) 引用操作　对相关信息进行引用,查询数据库,并显示在相应视图中。

4) 上传事件操作　保存页面信息,进行决策值判断,决定下一流程往哪个页面跳转,并进行下一活动主题请求,看是否是最后一个办理人,然后决定页面跳转。

5) 计算操作　查询相关数据,进行收费计算,加载到页面。

4　结束语

本文将流程类业务过程分解为流程、操作、主体及单证四要素。在此基础上,使用 KAOS 方法构建业务流程领域对象、目标、操作、主体四个元模型,用于支持面向目标业务流程领域的形式化建模。通过扩展语义约束,规范了数据的形式与范围,帮助用户理解和参与,引导用户全面描述现实系统;通过引入组织本体,由于组织本体涉及组织的多个方面,采集的数据是多维的,支持 KAOS 方法的多角度分析,进一步保证需求获取的完整性和一致性。依据 PKAOS 方法建立的形式化规则构建了一个图形化的需求定制与开发平台 RCDP,完成一个需求的定制过程。实践证明,该方法能够帮助领域专家构建流程类业务领域的需求模型,提高需求模型的构建质量。

参考文献:

[1] 陆汝钤,金芝,陈刚. 面向本体的需求分析[J]. 软件学报, 2000, 11(8): 1009-1012.

[2] 金芝. 基于本体的需求自动获取[J]. 计算机学报,2000,23(5): 486-487.

[3] 张文燚,怀进鹏. 领域分析与建模研究[J]. 北京航空航天大学学报,2004,30(12): 1225-1230.

[4] 陈昊,应时. 一种用目标驱动技术扩充的用例分析方法[J]. 计算机工程与应用,2004,40(6): 66-68.

[5] 金芝,陆汝钤. 多范例自动需求建模和分析: 一种基于本体的方法[J]. 中国科学 E 辑: 技术科学,2003,33(4): 297-299.

[6] HARUHIKO K, MOTOSHI S. Using domain ontology as domain knowledge for requirements elicitation[C]//Proc of the 14th IEEE International Requirements Engineering Conference. Mineeapolis: IEEE Computer Society,2006: 642-644.

[7] 吴越,王智学,陈彬. 需求模型中目标的关系及其发现方法[J]. 计算机工程, 2008,34(14): 35-36.

(上接第 1403 页) 为方便比较,本文采用了相同的内核函数,未经过优化的程序可能导致不同平台有不同的运行时间。对于同在 NVIDIA 平台上的 OpenCL 和 CUDA,使用相同的内核函数时 CUDA 的效率约为 OpenCL 的两倍。这其中包括 OpenCL 在设计成良好移植性时所带来的性能损失,也可能来自于驱动程序的支持不足造成。

对程序的分析除加速比外还应该考虑更多重要内容。其中很重要的一点是精度。对于 OpenCL 规范来说,其在标准中只支持单精度类型。多精度作为扩展内容给出,根据不同厂家设计而选择性实现。在本次实验中,本文只使用了单精度计算。对于精度要求高、迭代循环中涉及精度的问题,期待 OpenCL 标准在未来有新的补充和完善。

5　结束语

从实验来看,利用 OpenCL 和硬件提供的强大计算能力可以大大提高像方腔流或类似问题的计算速度。在实验中,也给出了多平台的性能对比,开发人员可以通过比较分析选择适合自己的平台。OpenCL 的优势在于其拥有优秀的加速比的同时也具有良好的跨平台移植性。但开发人员若要获得硬件最好的性能,需要尝试针对不同硬件特点使用更有针对性的开发环境或算法。

对于现在不同厂家所实现的驱动, OpenCL 无法达到真正的统一。目前 OpenCL 的规范只能单独运行在 AMD 或 NVIDIA 产品上。开发人员可以使用 AMD CPU 加 ATI 显卡的策略,但无法使用 ATI 加 CUDA 进行统一的 OpenCL 编程。这给真正异构编程带来很多困难。笔者期待该标准的进一步完善。

也可以使用 MPI 加 OpenCL 实践多节点间的并行策略,其具体操作和理念同 MPI 加 CUDA 完全相同。对此,已经有很多文献进行过讨论,如文献[1],在此不再多述。在使用多个设备节点时,如何平衡各个设备节点间的计算任务,是今后工作的重点。

参考文献:

[1] 李博,李曦鹏,张云,等. 耦合 NVIDIA/AMD 两个 GPU 的格子玻尔兹曼模拟[N]. 科学通报,2009-12-31.

[2] DONG Ting-xing, LI Xin-liang, LI Sen, *et al*. Acceleration of computational fluid dynamics codes on GPU[C]//Proc of the 8th Asian Computational Fluid Dynamics Conference. 2010.

[3] 李新亮,马延文,傅德薰. 迎风紧致格式的混淆误差分析及其同谱方法的比较[J]. 计算物理,2002,19(4): 17-23.

[4] 李新亮,傅德薰,马延文. 复杂流体直接数值模拟软件 OpenCFD 理论手册[P]. 北京: 中国科学院力学研究所 LNM,2007.

[5] 张舒,褚艳利. GPU 高性能运算之 CUDA[M]. 北京: 水利水电出版社,2009.

[6] MUNSHI A. The OpenCL specification[S]. America: Khronos OpenCL Working Group,2009.

[7] AMD Corporation. ATI stream SDK OpenCL programming guide[K]. America: AMD,2010.

[8] NVIDIA Corporation. NVIDIA OpenCL JumpStart guide[K]. America: NVIDIA,2009.