GPGPU 技术研究与发展

Research and Development of the General-Purpose Computation on GPUs

林一松,唐玉华,唐 滔

LIN Yi-song, TANG Yu-hua, TANG Tao

(国防科学技术大学计算机学院,湖南 长沙 410073)

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

摘 要:半导体工艺的发展使得芯片上集成的晶体管数目不断增加,图形处理器的存储和计算能力也越来越强大。目前,GPU 的峰值运算能力已经远远超出主流的 CPU,它在非图形计算领域,特别是高性能计算领域的潜力已经引起越来越多研究者的关注。本文介绍了 GPU 用于通用计算的原理以及目前学术界和产业界关于 GPGPU 体系结构和编程模型方面的最新研究成果。

Abstract: With the development of the semiconductor technology, the number of transistors integrated on a chip keeps increasing. Consequently, the computation and memory capacity of graphics processing units improve rapidly. So far, the floating-point computing capacity of GPUs has greatly exceeded that of CPUs, and the potential of GPUs in the non-graphic computing field, especially in high performance computing, has attracted more and more researchers' attention. This paper gives an introduction to the principles of the general purpose computation on GPUs and the latest research results about architecture and the programming model of GPGPU from both the research community and industry.

关键词:GPGPU;体系结构;编程模型

Key words: GPGPU; architecture; programming model

doi: 10.3969/j.issn.1007-130X.2011.10.016

中图分类号:TP303 **文献标识码**:A

1 引言

随着半导体工艺的发展,芯片上集成的晶体管数目按照摩尔定律增加,各种处理器的存储和计算能力也因此不断提高。作为最常用的两类处理器芯片,CPU 和 GPU 的性能发展呈现不同的规律。CPU 是计算机的主控处理部件,需要应对各种复杂的控制过程,追求的标量性能,注重通用性,因此在其芯片体系结构设计时,不得不将大量的芯片资源用于分支预测、乱序执行等复杂的控制逻辑,从而限制了其计算性能的提升。而 GPU 原本只负责加速图形计算,功能相对比较单一,芯片的控制逻辑比较简单,主要的芯片资源都用于提升计算性能,因此其计算能力发展更为迅速。例如,AMD 公司 2009 年第一季度发布的专业级流处理 GPU FireStream 9270[1]的单精度浮点峰值性能达到 1.2 TFlops,而 Intel 公司顶级的 Core2 Extreme 9650

在计算性能方面,GPU 要高出 CPU 整整一个数量级。计算性能增长的同时,GPU 的存储性能也在不断提升,2～4 GB 的存储容量、256～512 位宽的存储总线都为 GPU 发挥出强大的计算能力提供了充分的保证。

除了计算性能外,从性/价比和性能/功耗比上看,GPU 相对于通用 CPU 同样拥有明显的优势。这些特性都使得 GPU 成为搭建超级计算机的良好选择,可以在有限的系统规模和金钱开销的限制下达到非常高的峰值计算性能,因此越来越受到高性能计算领域的关注。

计算性能不断提升的同时,限制 GPU 广泛应用的另一制约因素——可编程性,也在不断提高。早期的 GPU 仅负责图形加速,流水线的功能固定,只能执行有限的几个操作以完成图像的生成。硬件工艺的进步使得 GPU 内处理单元的结构越来越复杂,功能也越来越灵活,其流水线具备了执行用户自定义程序的能力,从而具备了初步的可编程性。从最初的底层图形 API 发展到现在高度抽象的 CUDA[2]、Brook +[3] 以及 OpenCL[4],GPU 的可编程性得到了极大的提升,已经具备了开发非图形领域应用的基本能力,GPU 上的通用计算(General Purpose computation on GPUs,简称 GPGPU[5])也成为体系结构、编程及编译等领域热点的研究方向之一。

2 GPU 通用计算原理

现代 GPU 渲染管线的结构如图 1 所示。

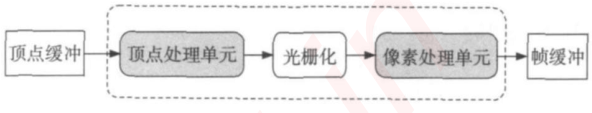

图 1 现代 GPU 渲染流水线

现代 GPU 渲染流水线主要包括顶点处理单元、光栅化单元和像素处理单元三个部分。进入可编程时代之前,进行 3D 图像渲染时,用户需要提供渲染对象在三维空间中的顶点坐标、法线、纹理坐标等参数,再定义好光源,就可以自动得到渲染出来的图像,渲染的过程由硬件自动完成,程序员无法精确控制。然而,这种黑盒的渲染模式越来越难以适应日益增长的图形应用,比如虚拟现实、大型 3D 游戏等,因此可编程流水线逐步发展起来。在图 1 所示的渲染流水线中,可编程的部分集中在顶点处理单元和像素处理单元,光栅化单元由于功运行在顶点处理单元和像素处理单元上的程序分别称为 vertex shader 和 pixel shader,程序员可以通过编程自由控制顶点变换、光照、像素的颜色计算以及纹理的采样过程,极大地提高了渲染流水线的渲染效率。利用 GPU 进行通用计算时,通用程序将被转换成 vertex shader 或者 pixel shader,映射到顶点处理单元或像素处理单元上运行。通常情况下,GPU 上的通用计算更偏向使用像素处理单元[6]。这是因为:首先,GPU 上的像素处理器一般要比顶点处理器的个数多;其次,像素处理单元在渲染流水线中位次靠后,和存储器的接合比顶点处理单元更为紧密,从它输出的数据可以直接写入存储器,并可以当作下一次执行的纹理数据使用。而顶点处理单元输出的数据则要通过光栅化以及像素处理单元才能写入存储器。

由于 GPU 是依靠大量运算单元并行执行来获取高吞吐率和计算性能,因此被映射到 GPU 上执行的程序通常是可并行化执行的循环结构,而其访问的数据的一般形式是数组。循环被编译器转换为各类 shader 并映射到 GPU 上运行,而其访问的数组则被当成纹理数据,存放在 GPU 的纹理存储器中。

由于不同的图形应用所需要的顶点处理和像素处理的计算量不同,在某些情况下,图 1 所示的顶点处理单元和像素处理单元分离的渲染架构会出现负载不平衡的现象,造成流水线的阻塞和停顿。从 DirectX 10 开始,业界提出了统一渲染架构模型,将顶点处理器和像素处理器合二为一,成为统一的流处理器,vertex shader 和 pixel shader 统称为流程序,都运行在这个流处理器上。统一渲染结构不但简化了渲染流水线的硬件构造,也避免了流水线各段的负载不平衡,极大地提高了资源利用率。

3 GPGPU 体系结构

目前可以生产支持通用计算的 GPU 厂家主要有 AMD、NVIDIA 和 Intel。本节我们将简要介绍这三个厂商目前主流 GPU 的核心体系结构。

3.1 AMD:R700

图 2 所示为 AMD R700 系列处理器体系结构框图。一般来说,GPU 的存储系统包括两部分,一部分是驻于内存中的系统空间,另一部分则是位于 GPU 上的显存空间,GPU 上的 DMA 引擎可以在

统中包含的数据包括命令队列、指令、常数以及输入、输出流,其中命令队列指明了 GPU 需要处理的任务,指令给出了执行部件的具体工作,常数、输入和输出流则给出了计算所需要的数据。这几个部分要素构成了 GPU 运行的基本要素。R700 芯片上的部分包括一个存储控制器(内含 DMA 引擎)、一个命令处理器、一个线程分派器以及一个数据并行处理器阵列(Data-Parallel Processor Array)。命令处理器负责读取由主处理器写入到 GPU 控制寄存器的命令,控制 GPU 的执行,当计算完成时,向主处理器发送硬件中断。线程分派器则根据控制命令指定的计算区间派生线程并分配到数据并行处理器阵列上去执行。数据并行处理器阵列是 R700 的核心部件,它由一系列互相独立的 SIMD 流水线组成,它们可以对流中的元素进行并行处理,或者通过存控对存储系统进行读写。R700 通过在 SIMD 流水线中维护数以百计的线程同时执行以及重叠访存和计算来隐藏访存延迟。

3.2 AMD:APU

为进一步提升处理器的工作效率,AMD 公司将 CPU 和 GPU 的核心进行融合(Fusion),推出了 APU[7](Accelerated Processing Units)系列处理器,以加速多媒体和向量处理能力。业界有观点认为,这种融合的体系结构是未来处理器的发展方向。

从图 3 可以看出,APU 处理器将多核的 x86 速总线以及存控融合在一块单一芯片上,可以实现数据在不同处理核心间的高速传输,大大缓解了目前 CPU 和 GPU 之间的通信受限于外部总线的问题。类似的融合体系结构还有 Intel 公司的 Sandy bridge 架构。

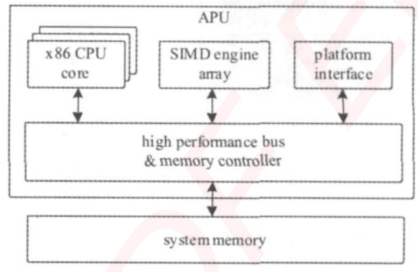

图 3 AMD APU 结构示意图

3.3 NVIDIA:GT200

GT200 是 NVIDIA 公司在 G80 架构的基础上推出的第二代基于 DX10 统一架构的 GPU 核心,是 NVIDIA 生产的 GeForce GTX280、GTX295 等显卡所采用的显示芯片。

从图 4 可以看出,GT200 核心总共拥有 10 个线程处理簇,每个线程处理簇拥有 24 个流处理核心,即总共拥有 240 个流处理核心。每个流处理核心主要包含一个整数处理单元和浮点处理单元。线程处理簇内每 8 个流处理核心为一组,构成一个流多处理器(Streaming Multiprocessor,简称 SM),共享一块大小为 16 KB 的局部存储器。每个

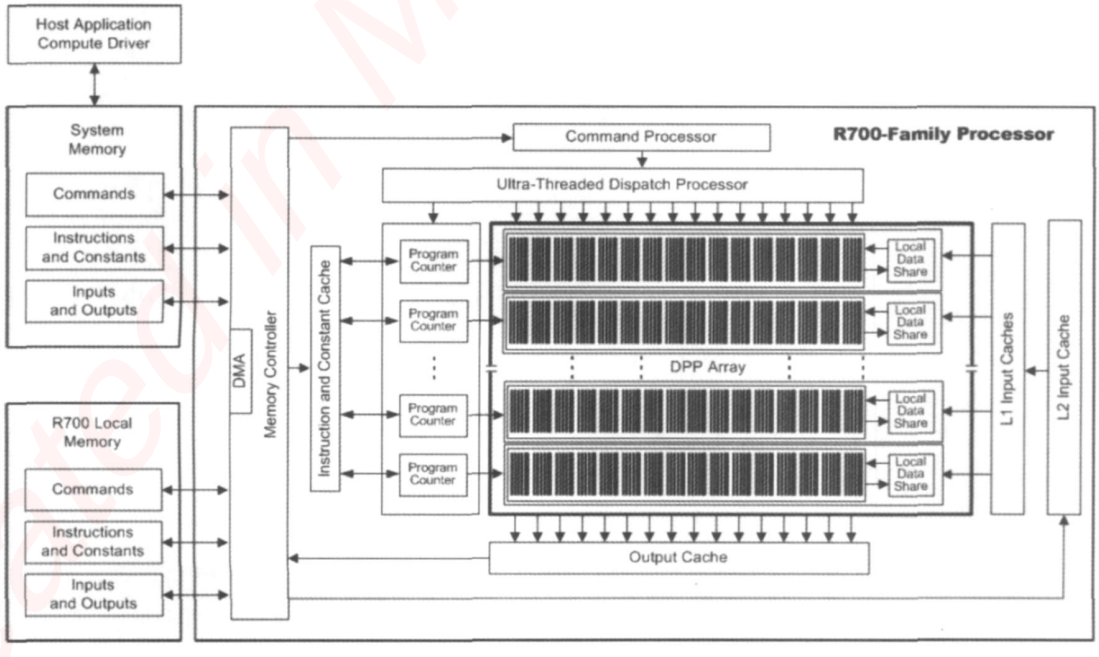

图 2 AMD R700 体系结构示意图

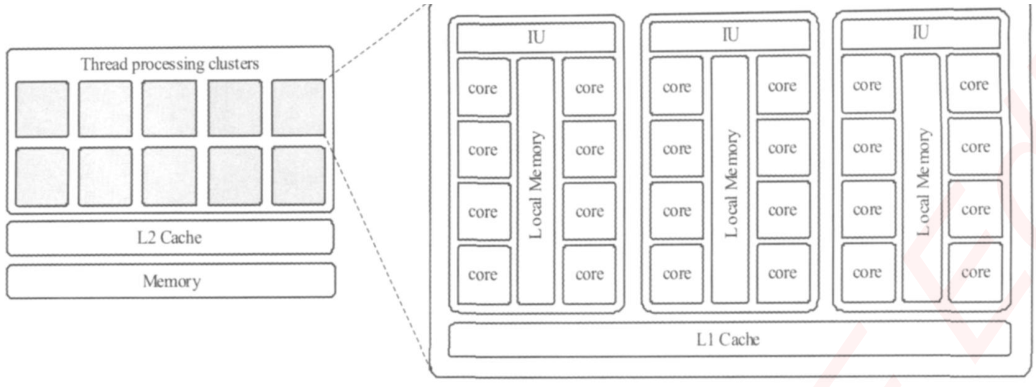

图 4　NVIDIA GT200 核心结构示意图

流多处理器拥有 16 个 load/ store 单元,每拍可以为 16 个线程计算访存地址。每个流多处理器还拥有 2 个特殊功能单元(Special Function Unit),负责计算超越函数指令,如正、余弦以及方根计算。特殊功能单元指令的分派与其他单元是独立的,因此它们可以并行执行。整个线程处理簇拥有一块 L1 Cache。L2 Cache 则为所有的线程处理簇共享。

3.4　NVIDIA:Fermi

Fermi[8]是 NVIDIA 公司推出的最新一代 GPU,提供了强大的双精度浮点计算能力。Fermi 较之以往 GPU 的不同之处在于:首先,为面向对数据精确度要求较高的应用,Fermi 首次在 GPU 的存储器中引入了 ECC 校验的功能;其次,Fermi 在保持了主流 GPU 中使用的 Shared Memory 作为片上存储层次外,还首次引入了通用的数据 Cache,以更好地支持通用计算。Fermi 中的 Cache 包含被 SM 私有的 L1 Cache 和被所有 SM 共享的 L2 Cache。此外,Fermi 中的 Shared Memory 和 Cache 可以通过配置互相转化。最后,Fermi 支持多 Kernel 同时执行。

图 5 给出了 Fermi 的体系结构框图。图中包含 16 个 SM,分布在共享的 L2 Cache 周围,每个 SM 均包含 L1 Cache、寄存器文件等私有存储资源,32 个流处理核心构成的计算阵列和线程调度单元。处理器的外围则分布着 DRAM、Host 和线程调度等接口。值得注意的是,Fermi 之前的 GPU 中 SM 内大多包含 8 个流处理核心,如 GT200,因此需要 4 拍才能发射一个 warp,而 Fermi 中将流处理核心数提升至 32 个,而且采用双发射技术,使得每拍可以发射 2 个 warp,极大提升了线程发射效率。

3.5　Intel:Larrabee

Larrabee[9]是 Intel 公司推出的第一款高性能

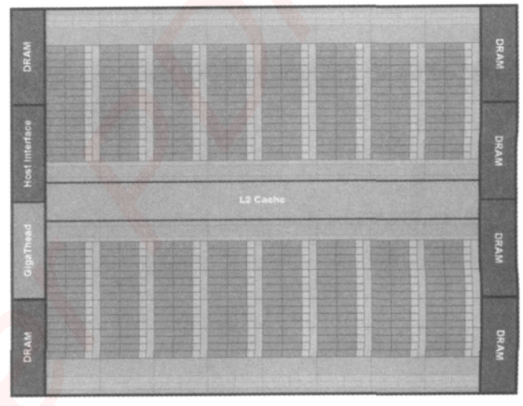

图 5　NVIDIA Fermi 体系结构示意图

独立显卡,主要面向高端通用计算平台。它拥有 2 GB 的 GDDR5 显存,128 GB/ s 的存储带宽,至少 16 个处理核心,每个核心的双精度峰值计算性能可以达到 14～40 GFLOPS。

Larrabee 基于传统的 x86 指令集结构设计,支持子函数调用以及页故障的处理,面向通用计算的潜力比一般的 GPU 更大。从图 6 可以看出,它采用顺序执行的 CPU 核心,通过处理器内的互联网络连接起来,可以执行标量指令或者宽向量指令。但是,为了提高并行度,容纳更多核心,Larrabee 采用较以往多核 CPU 更简单的核心,不支持 SSE 或 MMX 扩展指令。二级缓存被设计成分布式的,每个 CPU 核心对应其中一块,这样提高了 Cache 的访问效率,简化了数据同步和共享。

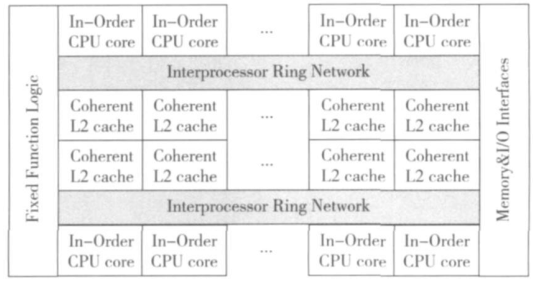

图 6　Larrabee 结构框图

从体系结构的角度,我们可以大致将上述几种 GPGPU 分为三类:

第一类由传统 GPU 演化而来,包括专业显卡供应商 AMD/ATI 和 NVIDIA 推出的一系列显卡。这类结构的显著特征是包含大量结构简单的处理核心构成的阵列,以高度并行化的方式批量处理数据,这些带有向量特征的处理阵列是由传统 GPU 中的多条并行的渲染管线发展而来。此外,这类 GPGPU 中仍或多或少存在一些专用于图像处理的部件,如纹理 Cache、帧缓冲等。随着面向通用计算的发展趋势,GPGPU 越来越专注于通用计算能力而渐渐弱化其作为显卡的功能。

第二类是由传统的多核 CPU 发展而来的,典型的代表是 Intel 的 Larrabee。这类 GPGPU 的主要特征是对传统的 CPU 核心进行裁剪,得到相对轻量级的处理核心,组成其计算部件,因此可以兼容部分传统 CPU 的指令集。相对于第一类 GPGPU 中的细粒度处理单元,这类处理核心仍然比较复杂,因此核心的集成度远不及第一类 GPGPU。

最后一类是将 CPU 和 GPU 体系结构相融合,其代表为 AMD 的 APU。这类体系结构的特点是将 GPGPU 中的处理阵列直接作为 CPU 的加速部件集成到同一个芯片内部。一方面说,CPU 核心的融入增强了 GPGPU 的标量处理能力,更适合通用计算的要求,同时融合的结构也大大缓解了 GPGPU 和 CPU 之间的通信带宽受限问题。

4 GPGPU 编程模型

GPU 的编程模型的发展和变迁和 GPU 的硬件水平的发展紧密相关。早期的 GPU 流水线功能简单,只能执行比较固定的几类操作,并且都和图形处理密切相关,很难将非图形领域的应用映射到 GPU 上执行。即使在图形处理应用中,程序员也只能通过简单的图形 API(DirectX[10]、OpenGL[11])进行程序映射,编程效率低下。随着 GPU 流水线功能的进一步完善,可编程性不断提高,出现了比图形 API 更高级的着色语言(HLSL[12]、GLSL[13]、Cg[14])。着色语言是从 C 语言的基础上扩充而来的,简化了编程的复杂性,提高了 GPU 的可编程性。但是,它们仍然没有有效屏蔽 GPU 的硬件细节,带有非常明显的图形处理痕迹,难以用于开发通用计算程序。

随着 GPGPU 的火热兴起,各主流 GPGPU 供主要的 GPGPU 编程模型,包括 AMD 的 Brook+、NVIDIA 的 CUDA 以及业界正在制定的覆盖 CPU、GPU 等多种计算平台的统一编程标准 OpenCL。下面先对这几种业界提出的编程模型进行介绍,再介绍学术界相关的研究进展。

4.1 Brook+

Brook+是 AMD 公司在 Brook for GPU[15]的基础上修订而来的,其最初的原型来自 Stanford 大学流处理器研究组提出的流编程语言 Brook[16]。Brook 在底层的编程语言和运行库如 OpenGL 的基础上进一步抽象,屏蔽了底层实现细节,提供了一种高级的编程接口。它基于 C 语言进行扩展,引入了流(Stream)和核心(Kernel)两个重要的概念。流是指一组结构相同的可以被并行操作的数据对象,核心函数则是指由程序员自定义的施加在流元素上的操作。核心函数隐式地并行施加于所有的流元素之上。Brook for GPU 是 Brook 的 GPU 版本,引入了一些针对 GPU 硬件的特性描述,其编译器和运行环境,将 GPU 抽象为一个流处理器,从而将流编程模型引入 GPU,有效地支持了 GPU 的通用计算。AMD 在 Brook for GPU 的基础上针对其高性能 GPU 做了很多修订,推出了 Brook+编程模型。

图 7 所示为 Brook+软件结构框图。Brook+源码经过分离,产生 CPU 部分代码和 GPU 部分代码,其中 CPU 代码交由本地 C++编译器编译执行,而 GPU 部分则交由 Kernel 编译器编译产生在 GPU 上运行的可执行代码。CPU 代码以函数调用的形式调用核心函数完成计算任务。

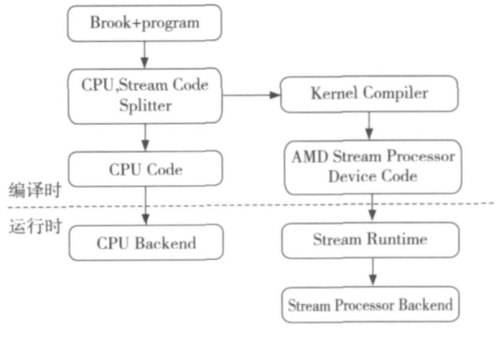

图 7 Brook+编程环境框架

在 GPU 内,核心函数以 SIMD 方式并行执行于所有的流元素之上,执行的范围称为执行区间(Domain),GPU 为执行区间内每个点都派生出一个线程,并调度到某一个线程处理器上执行,如图 8 所示。线程执行的内容就是核心函数。程序员

间内每个点所执行的操作,而诸如线程派生,线程分配和调度,存储访问等操作都是由 GPU 硬件自动完成的,这样屏蔽了 GPU 硬件细节,极大地简化了 GPU 通用计算的开发。

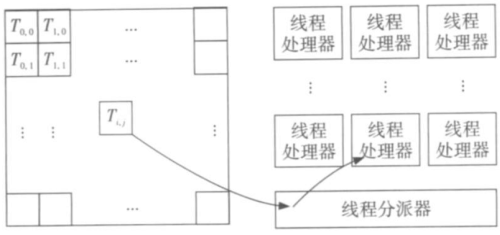

图 8 Brook + 并行计算模型

4.2 CUDA

CUDA 全称统一计算设备框架(Compute Unified Device Architecture,简称 CUDA),是 NVIDIA 公司针对其 GPU 设计提出的 GPGPU 编程模型,它也以 C 语言为基础,降低了编程难度。在 CUDA 计算模型中,应用程序分为 host 端和 device 端,前者是指运行于 CPU 上的部分,后者则是运行于 GPU 上的部分,和 Brook + 一样,这部分代码称为核心(Kernel)函数。CPU 代码准备好数据后,复制到显卡的存储器中,再调用核心函数进行执行,执行完毕后再由 CPU 代码将计算结果拷贝回主存。

在 CUDA 计算模型下,GPU 执行的最小单位是线程,多个线程组成一个线程块,线程块中的线程可以共享一片存储器,并以任意顺序执行,在硬件资源受限的情况下甚至可以串行执行。一个核心程序由一个或多个线程块组成,一个应用则由若干个核心程序构成。图 9 给出各计算实体的对应关系。

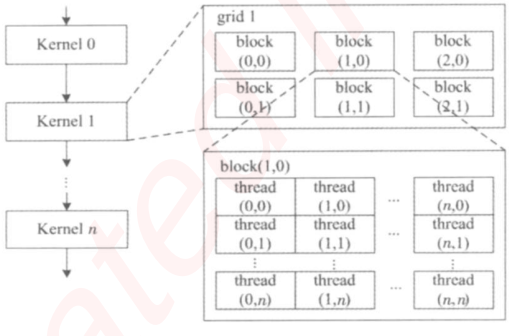

图 9 CUDA 计算模型示意图

4.3 OpenCL

OpenCL 是一个业界正在制定的统一编程标准,它可以覆盖通用 CPU、GPU、DSP 芯片、手持一个统一的框架中,向程序员提供一组 API 和运行库,使得程序员可以透明地使用这些设备。

和 CUDA 类似,OpenCL 的计算平台由一个主控设备(Host Device)和一组计算设备(Compute Device)组成,OpenCL 主控程序运行于主控设备之上,并向各计算设备提交计算命令;计算设备上运行核心(Kernel)程序,完成计算后将结果返回给 OpenCL 主控程序。通常情况下,OpenCL 的计算设备都由一个或多个计算单元(Compute Unit)组成,而每个计算单元又包含多个处理单元(Processing Unit),它们组成了一个三层的嵌套关系,提供了大量潜在的并行性。主控设备在向计算设备提交一次核心计算时,会指定一个执行的区间,计算设备会为这个区间内所有的点都派生出一个线程来执行。每一个线程称为一个工作单元(Work-item),而多个工作单元可以被分组,称为一个工作组(Work-group)。一个工作组上的所有工作单元可以共享一块局部存储器。

计算设备上运行的核心程序可以访问四类存储对象,包括全局存储器、常数存储器、局部存储器和私有存储器。全局存储器对于所有工作组内的所有工作单元都可以进行读写访问,并且可以访问存储对象的任意位置。常数存储器存放核心程序执行所需要的常数参数,由主控程序负责分配和初始化。局部存储器则隶属于一个工作组,只对组内的工作单元可见。私有存储器是属于一个工作单元的存储单元,不对其他工作单元可见。各类存储对象在计算设备上的分布和使用情况如图 10 所示。

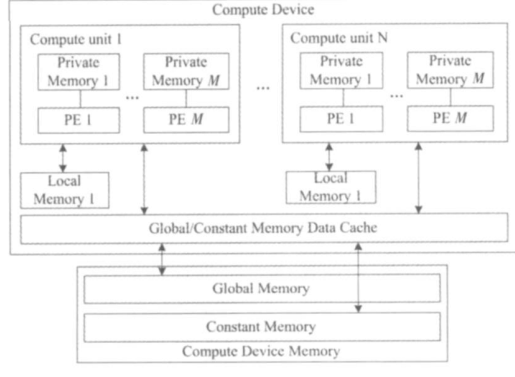

图 10 OpenCL 存储模型

4.4 编程模型对比分析

以上介绍的三种 GPGPU 编程模型中,Brook + 和 CUDA 属于设备相关的编程模型,仅支持相关厂商推出的 GPGPU,而 OpenCL 则属于设备无

抽象。

从计算与数据的关系来说,Brook + 属于一种典型的流编程模型,相对 CUDA 和 OpenCL 具有更明显的以数据为中心的计算模型。Brook + 程序中的数据被定义为流,计算空间是按照输出流的空间派生出来的,计算空间和流空间一一对应,一个计算点一般只计算流中的一个数据。对数据流的随机访问需要将流设置成专门的访问类型。与之相对,CUDA 和 OpenCL 则不存在这类限制,它们直接将计算空间的组织交给程序员完成,更加贴近于传统的多线程编程模型,由程序员指定线程空间以及每个线程完成的操作,而数据默认都可以通过下标进行随机访问。

4.5 学术界的最新研究

目前,学术界大部分关于 GPGPU 的研究集中在 GPGPU 上应用程序的开发与优化方法上,但也有一部分研究者致力于改善 GPGPU 的编程界面和软件支撑环境。如 Lee S 等人[17]提出了 OpenMP-to-GPGPU 编译框架,把 OpenMP 程序自动地源到源转换成 CUDA 程序。这是一种通过编译指导的方式将传统程序自动映射到 GPGPU 平台上的方法。这种方法已经被业界公司所采用,PGI 公司在其推出的 PGI8.0 编译器中,就通过一组自定义的类 OpenMP 的编译指导命令来支持 GPU 通用编程[18]。Dolbeau R 等人提出了一个异构多核的并行编程环境 HMPP[19],它也是通过添加编译指导命令的方式指定一部分代码在加速芯片,如 GPU 上运行。HMPP 与 OpenMP-to-GPGPU 以及 PGI8.0 的区别在于它没有将原始代码做源到源的转化,它仅仅是提供了一种异构平台下的混合编程和运行机制,编译指导命令所指定的代码段要用目标平台的语言单独进行编写和编译,最后链接入主程序。EXOCHI[20]是 Wang P H 等人提出的另一种异构多核多线程系统的编程环境。EXOCHI 将 GPU 抽象成一个 MIMD 的系统资源,而不仅仅是一个设备,这种资源可以和 CPU 通过共享虚存的多线程模型进行紧耦合。很多研究者认为 CPU 和 GPU 将最终融合在一起,这种共享虚存的多线程模型将是最适合这种融合体系结构的编程模型。Saha B 等人[21]沿用这种思想提出了异构 X86 平台(Intel CPU + Larrabee)的编程模型,进一步优化了共享虚存的实现技术并给出了完整的解决方案。

5 结束语

GPU 以其异常强大的计算性能吸引了诸多非图形领域研究者的关注。目前国内外关于 GPU 的通用计算技术研究如火如荼,但大多集中在 GPU 应用程序的开发与优化方法上。相对而言,关于 GPU 编程、调试、容错、低功耗等方向的研究比较少,这些方面都是 GPU 面向通用计算的薄弱环节,极大地限制了 GPU 面向通用计算的潜力。此外,一些经典如存储墙一类的问题,在 GPU 上依然存在,还需要更加深入系统的研究。

参考文献:

[1] http://ati.amd.com/technology/streamcomputing/product_firestream_9270.html.

[2] Kirk D. Nvidia Cuda Software and GPU Parallel Computing Architecture[C] //Proc of the 6th International Symposium on Memory Management, 2007:103-104.

[3] http://ati.amd.com/technology/streamcomputing/AMDBrookplus.pdf.

[4] Open Computing Language[EB/OL]. [2009-11-16]. http://www.khronos.org/.

[5] Luebke D, Harris M, Krüger J, et al. Gpgpu: General Purpose Computation on Graphics Hardware[C] //Proc of SIGGRAPH '04, 2004:33.

[6] Harris M. Mapping Computational Concepts to GPUs[C] //Proc of SIGGRAPH '05, 2005:50.

[7] The AMD Fusion Family of APUs[EB/OL]. [2009-11-15]. http://www.fusion.amd.com/.

[8] Glaskowsky P N. NVIDIAs Fermi: The First Complete GPU Computing Architecture[R]. A White Paper Prepared Under Contract with NVIDIA Corporation, 2009:1-26.

[9] Seiler L, Carmean D, Sprangle E, et al. Larrabee: A Many-Core x86 Architecture for Visual Computing[J]. ACM Transcation on Graphics, 2008, 27(3):11-15.

[10] Pike A. DirectX 8 Tutorial[M]. 2006.

[11] Shreiner D, Woo M, Neider J, et al. OpenGL Programming Manual[M]. 5 edition. OpenGL ARB, Boston: Addison-Wesley, 2005.

[12] Microsoft. High-Level Shader Language[EB/OL]. [2003-05-16]. http://en.wikipedia.org/wiki/high-level-Shader-language.

[13] Kessenich J, Baldwin D, Rost R. The OpenGL Shading Language[EB/OL]. [2003-05-16]. http://www.opengl.org/documentation/g/sl/.

[14] Mark W R, Glanville R S, Akeley K, et al. Cg: A System for Programming Graphics Hardware in a Clike Language [C] //Proc of SIGGRAPH '03, 2003:896-907.

[15] Buck I, Foley T, Horn D, et al. Brook for GPUs: Stream Computing on Graphics Hardware[J]. ACM Transactions

[16] Buck I. Brook Spec v0.2[R]. Technical Report, Stanford University, 2003.

[17] Lee S, Min S, Eigenmann R. OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization [C]//Proc of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2009: 101-110.

[18] Wolfe M, Engineer C. The Portland Group [EB/OL]. [2003-06-15]. http://www.hpcwire.com/specialfeatures/sc08/features/Compilers_and_More_A_GPU_and_Accelerator_Programming_Model.html.

[19] Dolbeau R, Bihan S, Bodin F. HMPP: A Hybrid Multi-Core Parallel Programming Environment [C]//Proc of Workshop on General Processing Using GPUs, 2006.

[20] Wang P H, Collins J D, Chinya G N, et al. Exochi: Architecture and Programming Environment for a Heterogeneous Multi-Core Multithreaded System [C]//Proc of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2007: 156-166.

[21] Saha B, Zhou X, Chen H, et al. Programming Model for a Heterogeneous x86 Platform [C]//Proc of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2009: 431-440.

士生,研究方向为计算机体系结构与低功耗优化。E-mail:linyisong@live.cn

LIN Yi-song, born in 1983, PhD candidate, his research interests include computer architecture, and low-power optimization.

唐玉华(1962-),**女,江苏南京人,硕士,研究员,研究方向为计算机体系结构和计算机网络。E-mail:**

TANG Yu-hua, born in 1962, MS, research fellow, her research interests include computer architecture, and computer networks.

唐滔(1984-),**男,安徽肥西人,博士生,研究方向为计算机体系结构与编译优化。E-mail:**taotang84@nudt.edu.cn

TANG Tao, born in 1984, PhD candidate, his research interests include computer architecture, and compiler optimization.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植

15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究

14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发

6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计

18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究
16. 基于 UEFI 的 Application 和 Driver 的分析与开发
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法

4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论