图形处理器用于通用计算的技术、现状及其挑战*

吴恩华[1,2+]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[2](澳门大学 科技学院 电脑与资讯科学系,澳门)

State of the Art and Future Challenge on General Purpose Computation by Graphics Processing Unit

WU En-Hua[1,2+]

[1](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau, China)

+ Corresponding author: Phn: +853-3974468, Fax: +853-831694, E-mail: ehwu@umac.mo, http://www.umac.mo

Received and Accepted 2004-08-14

Wu EH. State of the art and future challenge on general purpose computation by graphics processing unit. *Journal of Software*, 2004,15(10):1493~1504.

http://www.jos.org.cn/1000-9825/15/1493.htm

Abstract: Graphics processing unit (GPU) has been developing rapidly in recent years at a speed over Moor's law, and as a result, various applications associated with computer graphics advance greatly. At the same time, the highly processing power, parallelism and programmability available nowadays on the contemporary GPU provide an ideal platform on which the general-purpose computation could be made. Starting from an introduction to the development history and the architecture of GPU, the technical fundamentals of GPU are described in the paper. Then in the main part of the paper, the development of various applications on general purpose computation on GPU is introduced, and among those applications, fluid dynamics, algebraic computation, database operations, and spectrum analysis are introduced in detail. The experience of our work on fluid dynamics has been also given, and the development of software tools in this area is introduced. Finally, a conclusion is made, and the future development and the new challenge on both hardware and software in this subject are discussed.

Key words: graphics processing unit; general purpose computation; programmability; real time computation; stream processing

摘 要: 多年来计算机图形处理器(GPU)以大大超过摩尔定律的速度高速发展.图形处理器的发展极大地提高

* Supported by the National Natural Science Foundation of China under Grant Nos.60033010,60173022,60223005 (国家自然科学基金);the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312102 (国家重点基础研究发展规划(973)); the Research Grant of University of Macau(澳门大学研究基金).

作者简介: 吴恩华(1947－),男,江苏南通人,博士,研究员,博士生导师,主要研究领域为计算机图形学,可视化,虚拟现实.

本文专为《软件学报》创刊 15 周年而撰写,并在"庆祝《软件学报》创刊 15 周年暨学术报告会"上作为大会特邀报告进行宣读.

了计算机图形处理的速度和图形质量,并促进了与计算机图形相关应用领域的快速发展.与此同时,图形处理器绘制流水线的高速度和并行性以及近年来发展起来的可编程功能为图形处理以外的通用计算提供了良好的运行平台,这使得基于GPU的通用计算成为近两三年来人们关注的一个研究热点.从介绍GPU的发展历史及其现代GPU的基本结构开始,阐述GPU用于通用计算的技术原理,以及其用于通用计算的主要领域和最新发展情况,并详细地介绍了GPU在流体模拟和代数计算、数据库应用、频谱分析等领域的应用和技术,包括在流体模拟方面的研究工作.还对GPU应用的软件工具及其最新发展作了较详细的介绍.最后,展望了GPU应用于通用计算的发展前景,并从硬件和软件两方面分析了这一领域未来所面临的挑战.

关键词: 图形处理器(graphics processing unit,简称GPU);通用计算;可编程性;实时计算;流处理机

中图法分类号: TP301 **文献标识码**: A

似乎在一夜之间,计算机图形处理器(graphics processing unit,简称 GPU)用于通用计算的研究就成为一个十分热门的话题.人们感到惊奇的是,在计算机图形处理器多年迅速发展的进程中,几乎没有人认真地预言过这一重大应用.而在今天,由于计算机图形处理器具备了极高的性能,以及具备了雄厚的技术、工业和应用基础,使得人们对 GPU 的这一新的应用前景寄予了前所未有的期望和热情.微机的图形处理器所具有的高性能从下例可见一斑:2004 年,Nvidia GeForce 6800 Ultra 可达到峰值 40 Gigaflops,而 Intel 3GHz Pentium 4 采用 SSE 指令集也只能达到 6 Gigaflops.自从 1993 年以来,GPU 的性能以每年 2.8 倍的速度增长,而且估计还可以维持这一增长速度 5 年左右.现在每隔半年左右,新一代的 GPU 便会诞生.为此,当人们发现图形处理器能够有效地应用于通用计算时,这首先引起了图形界的极大关注.2003 年,世界上唯一的图形学硬件年会(SIGGRAPH/EUROGRAPHICS Graphics Hardware 2003)迅速将其重点转向了计算机图形处理器的非图形应用.甚至有专家认为[1],GPU 将进入计算的主流.此届年会将成为现代计算的一个转折点.也有人提出,GPU 将变成 General Processing Unit 的概念.无论这一论断是否言过其实,一年来新的研究工作又在多方面取得了重要进展.

本文首先从 GPU 发展的历史开始简单地介绍图形处理器用于通用计算的发展过程及其技术原理,然后介绍 GPU 用于通用计算的主要成功领域及发展现状,包括软件领域的发展,最后从硬件结构和软件发展两方面总结和展望未来的发展趋势及其面临的挑战.

1 图形处理器的发展及其可编程功能

当我们惊叹计算机图形处理器近两、三年来不可思议地迅速扩展到通用计算领域时,简单地回顾一下图形处理器的主要发展进程是一件很有意义的事情.

在计算机图形学发展初期的 20 世纪六、七十年代,由于受硬件条件的限制,图形显示只是作为计算机输出的一种手段.为此,在规范化图形应用的设计实现即定义图形用户界面时,人们只是纯粹地从软件实现的角度考虑问题.图形用户界面国际标准 GKS(GKS3D)[2,3],PHIGS[4]的出现是这种规范化的典型代表.它所定义的图形输出流水线(如图 1 所示)曾经在一段时间内引导了图形软件的实现和应用,甚至曾经有人试图使用完整的硬件设计实现其流水线的全部功能,生产出所谓“GKS 工作站”.然而众所周知,这种企图很快便销声匿迹,因为硬件的发展尚未达到这样的能力.其后的发展证明,当时用户界面标准的设计只是理想化的标准而已.

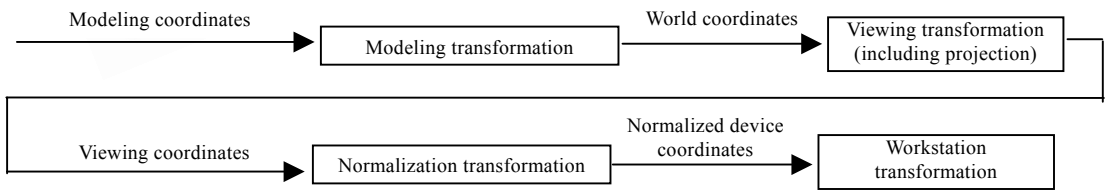

Fig.1 Graphics output pipeline in standard

图 1 图形标准输出流水线

从 20 世纪 80 年代初期开始,以 GE(geometry engine)为标志的图形处理器的出现和不断更新使得计算机图

形的发展进入了由图形处理器引导其发展的年代.20 世纪 80 年代初期,GE 芯片[5]的推出是计算机图形处理器开始进入市场的标志,它对其后的图形发展和变革产生了巨大影响.GE 的核心是实现四维向量的浮点运算功能.它可由一个寄存器的定制码定制出不同功能,分别用于图形输出流水线中的矩阵乘法、裁减计算、投影缩放等操作,从而用 12 个 GE 单元完整地实现三维图形输出流水线的功能.该芯片的设计者 James Clark 以 GE 作为核心技术建立的 SGI 公司在其图形学发展和计算机图形工业应用中产生了巨大影响,这已为人熟知.基于其图形处理器功能的图形界面 GL 其后发展为 OpenGL,至今一直成为图形用户界面事实上的工业标准.

在 20 世纪 80 年代以及 90 年代的大部分时间里,计算机图形特别是三维图形的应用,主要集中在工作站以上的处理机上.此时,图形处理流水线的大部分功能尚由 CPU 完成.GE 及其图形处理器功能的不断增强和完善促使图形处理功能不断从 CPU 向 GPU 转移,而且 GPU 本身亦不断增强了图形处理的各种高级功能,比如遮挡剔除、纹理映射、透明/半透明处理、景深效果等.

现代图形处理的流水线主要集中在顺序处理的两大部分(如图 2 所示):第 1 部分是对图元实施几何变换及对图元属性进行处理,即将几何模型的多边形/三角形顶点数据流从 CPU 交由图形处理部件实现几何变换及属性处理(包括部分光照计算).第 2 部分则是在实现扫描转换进行光栅化以后进行一系列图形绘制处理,包括各种光照效果的合成、纹理映射、遮挡处理、反混淆处理等.图形输出处理本身具有规范的流水线操作,当越来越高级的图形绘制功能需要在光栅一级处理时,SIMD(单指令多数据)结构的处理机是最合适的并行硬件结构.用这一方法实现图形处理加速最典型的代表是 20 世纪 80 年代北卡罗莱那大学的 Pixel Plane 系列[6~8]以及 Pixel Flow[9]图形处理机.Pixel Plane 5[8]作为当时图形处理能力最强的绘制系统,可达到每秒一百万 Phong 多边形的图形绘制能力.它的高度并行性是由像素级的阵列处理单元所完成.其硬件系统包括多达 32 个数字处理器和 16 个绘制部件,每个绘制部件可以对一个 128×128 的像素阵列的每个像素实现二次多项式的并行计算,这种在图素处理上使用 MIMD(多指令多数据)结构并在像素级采用 SIMD 结构的并行处理,第一次为用户提供了可以编程的图形处理能力.

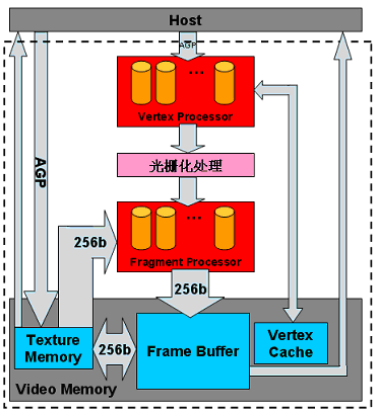

Fig.2 GPU rendering pipeline

图 2 GPU 绘制管道

伴随着虚拟现实、计算机仿真、计算机影视及游戏等应用不断增长的需求,图形处理的计算需求(计算量)一直也在不断增长.从 SGI 的 Geometry Engine 到 Nvidia 的 GeForce3 GPU[10]历经了 20 年,其芯片的线宽从 3μm 提高到 0.18μm(2004 年 GeForce 6800 为 0.13μm,集成度 1.5 亿晶体管),集成电路的逻辑设计能力提高了 300 倍,但是处理带宽仅提高了十几倍.而图形处理计算需求的增长还将继续下去.这样,固定流水线的模式已经很难适应图形处理不断增长的计算需求以及虚拟现实等应用的实时需求,可编程的灵活性是势在必行的.以 Nvidia 和 ATI 为代表的 GPU 的技术和产品则是适应了这种需求.与此同时,这种可编程且具有高速计算功能的图形处理能力则在通用计算领域找到了应用的舞台.

图 2 所示的 GPU 绘制管道的可编程并行处理能力分布在两部分:顶点处理器(vertex processor)和子素处理器(fragment processor).它们都是典型的流处理机(stream processor).这种流处理机与向量处理机的主要区别在于[11],它不具有大容量的快存/存储器可以读写,只是直接在芯片上利用临时寄存器作流数据的操作.对于 GPU 而言,图形流数据分别是顶点图元及光栅化后的像素(理论上多个子素即 Fragment 组成一个像素).根据图形处理的特点,GPU 流处理的元素为 4 个单元的向量,可以用它表示三维齐次坐标、三维空间齐次向量、颜色等.正是这种流处理机的并行结构,为通用计算提供了并行计算的平台.其中,顶点处理器操作在空间的几何点上,因而比较适用于除图形绘制以外的几何操作类的应用上,而子素处理器由于是操作在像素一级的单元上,具有较大容量的“纹理”空间,可以模拟纹理数据及其操作,因而可以更广泛地用于各种通用计算.这是 GPU 用于通用计算的最主要的平台.

自从 1998 年以来,GPU 的功能迅速更新,平均每一年多便有新一代的 GPU 诞生.在现代 GPU 概念出现以前,

特殊的图形硬件只出现在诸如 Silicon Graphics(SGI)等图形工作站上,具有基于硬件的顶点变换和纹理映射功能.第一代现代图形处理器出现在 1998 年后期,主要代表为 Nvidia TNT2,ATI Rage 和 3DFX Voodoo3.这些处理器主要处理光栅化部分,有些芯片支持多纹理,可以在光栅化过程中完成多幅纹理的融合操作.从 1999 年后期开始,第二代 GPU(Nvidia GeForce256,GeForce 2 和 ATI Radeon 7500)可以处理顶点的矩阵变换和进行光照计算,但此时还未出现真正的可编程性.第三代 GPU(Nvidia GeForce3,GeForce4,ATI Radeon 8500,2001 年和 2002 年早期)代表着第一次重要变革,这时可以将图形硬件的流水线作为流处理器来解释.顶点级出现可编程性,而在像素级出现有限的可编程性.在像素级程序中,访问纹理的方式和格式受到一定限制,只有定点数可用.正是这个时候,采用 GPU 作通用计算开始出现.第四代 GPU(Nvidia GeForce FX series,ATI Radeon 9700/9800)的像素和顶点可编程性更通用化,可以包含上千条指令.依赖纹理更为灵活,可以用作索引进行查找.GPU 具备了浮点功能,纹理不再限制在[0,1]范围,从而可以用作任意数组,这一点是对通用计算的一个重要贡献.最新的第五代 GPU 以 Nvidia GeForce 6800 为代表,功能相对以前更为丰富、灵活.顶点程序可以访问纹理,支持动态分支操作;像素程序开始支持分支操作,包括循环、if/else、重复等,支持子函数调用,64 位浮点纹理滤波和融合,多个绘制目标.

第 2 节将介绍到目前为止利用 GPU 作通用计算的最重要和典型的成功应用.由于我们在文献[12]中已经重点介绍了 GPU 在代数计算方面的应用,因而本文除介绍我们在这一方面(流体计算)的工作经验之外,代数计算只作简略介绍.其后将把重点放在介绍其他领域的通用计算,以及软件领域为这一课题所作的研究工作及其最新进展上.最后展望和讨论 GPU 用于通用计算的前景及其面临的问题.

2 基于图形处理器的通用计算

由于图形处理器并行流处理和可编程性的出现,越来越多的人开始用其做一些非绘制方面的计算.这些计算涉及的范围很广,从图形输出流水线以外的非绘制处理,到几何计算、碰撞检测、运动规划、代数运算、优化计算、偏微分方程 PDEs(partial differential equations)数值求解等,不一而足.如在几何处理方面,Govindaraju 等人[13]采用两个 GPU 来做遮挡剔除的运算,外加一个 GPU 用于最终的绘制.同样,Govindaraju 等人[14]利用 GPU 来进行快速碰撞检测.Sud 等人[15]采用 GPU 来加速三维距离场的计算.Tomov 等人[16]则用 GPU 来进行蒙特卡洛仿真.

将 GPU 用来做一些通常意义计算的例子很多,下面我们将就其中几个具有代表性的通用计算方面的应用进行较为详细的介绍,这些应用包括代数计算及流体模拟、数据库操作、频谱变换和滤波等.它们体现了 GPU 应用于通用计算的主要技术.更多的关于 GPU 用于通用计算的详细内容可以从以下网站获取 http://www.gpgpu.org.

2.1 代数计算及流体模拟

很早就有人利用图形卡进行一些简单的代数运算,如 Larsen 等人[17]在 2001 年利用多纹理技术做矩阵运算操作.后来随着顶点可编程的出现,Thompson 等人[18]在顶点级实现了一个代数运算的框架系统,其中包括矢量运算和矩阵乘法等.而像素可编程的出现加速了这方面的研究,如 Krüger 等人[19]利用像素程序做基本代数运算,Hall 等人[20]对于矩阵乘法运算利用像素程序做了若干优化,以充分利用 GPU 的高速缓存.近年来,这方面的研究越来越多,详细内容可以参考文献[12].

流体模拟一直是计算机图形学中的热点问题,其在电影特效和游戏中应用很多,而且在工程上的应用也很广泛.为了能够很好地描述流动现象,许多研究者不得不求解复杂的运动方程,如 Navier-Stokes 方程组 NSEs(navier-stokes equations).然而,NSEs 的求解相当耗时,且对时间步长极为敏感,很容易导致结果发散.近年来,为了加速流体运动问题的求解,人们开始利用 GPU 的并行性和可编程性通过求解有限差分方程解偏微分方程,以使整个模拟实时化.在像素程序出现以前,人们采用硬件本身支持的颜色融合操作和多纹理来计算传热和各向异性扩散有限单元方程[21],从而实现图像处理的功能.Harris 等人[22]通过 Register Combiner 编程求解 CML(coupled map lattice)问题,从而实现交互的对流模拟、反应扩散以及沸腾效果模拟.Li 等人[23]采用 Register

Combiner 结合 LBM(Lattice Boltzmann method)模拟流体和烟的效果.随着硬件可编程性的进一步发展,基于像素程序的流体模拟开始出现. Krüger 等人[19]利用像素程序做基本代数运算,并在此基础上实现了共轭梯度法和高斯-赛德尔迭代法,从而完成流体 PDEs 的求解.Bolz 等人[24]实现了基于像素编程的稀疏非结构化矩阵的共轭梯度法和正交网格的多重网格法,并用于加速几何处理和流体模拟.Goodnight 等人[25]实现了基于像素程序的多重网格算法,用来求解边界值问题(热传导问题、流体力学问题).Harris 等人[26]利用 GPU 求解云彩运动的流体方程.Li 等人[27]在其原来的基础上采用更为灵活的像素程序来求解.但是直到现在,大多数研究者只是集中在二维问题域,而且对于边界条件的处理过于简单,以至于很难满足实际问题的需要.造成这一问题的主要原因是,目前 GPU 缺乏像 CPU 那样的灵活性,编程也不如在 CPU 上那样容易.

Kim 等人[28]将冰晶体生长过程采用 GPU 来实施物理计算,整个物理方程组的计算在 GPU 上执行.Lefohn 等人[29]将 level-set 的等值面数据压缩为一个动态稀疏纹理格式,针对不同边界情况采用不同的像素程序计算 level-set 的 PDEs.

在利用 GPU 求解流体运动方程方面,我们亦开展了研究工作.为了充分利用 GPU 的并行性加速整个算法,以满足实时模拟的需要,我们在 GPU 上采用稳定性好的半拉格朗日方法[30]求解 NSEs,整个方程组形式如下:

$$\nabla\cdot\boldsymbol{u}=0 \tag{1}$$

$$\partial\boldsymbol{u}/\partial t=-(\boldsymbol{u}\cdot\nabla)\boldsymbol{u}+v\nabla^2\boldsymbol{u}-\nabla p+\boldsymbol{f} \tag{2}$$

$$\partial\rho/\partial t=-(\boldsymbol{u}\cdot\nabla)\rho+k_\rho\nabla^2\rho+S_\rho \tag{3}$$

$$\partial T/\partial t=-(\boldsymbol{u}\cdot\nabla)T+k_T\nabla^2T+S_T \tag{4}$$

这里,$\boldsymbol{u}$ 为速度矢量,v 为运动粘性系数,$\boldsymbol{f}$ 是外部作用力,p 是流体压强,而密度ρ和温度 T 是两个标量.整个方程描述了流体中的对流扩散等现象.

我们通过结合模板缓冲器对三维场景进行剪切操作,形成一系列实心的剖切截面以构成整个流动边界.这样,通过修正因子来处理任意复杂的边界条件,整个计算过程与几何场景的复杂度无关.图 3 所示为系统算法流程,整个计算由 4 步构成,先不考虑压强的影响求出一个中间速度分布,然后求解压强泊松方程,以修正速度场满足质量连续性条件.对于中等规模的问题,整个模拟和显示能够实时进行,图 4 给出了一个气流在城市上空流动的计算结果图.我们在二维问题上的 GPU 算法比对应的 CPU 算法效率要高出十几倍,具体内容可以参考文献[31~33].这一工作为我们下一步利用 GPU 做通用计算提供了一个很好的范例.

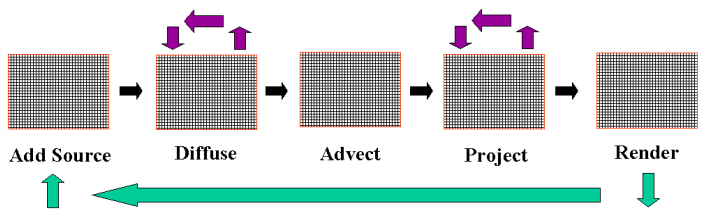

Fig.3 Flow chart in solving NSEs

图 3 求解 Navier-Stokes 方程流程图

Fig.4 Air-Flow dynamics over a city based on GPU computation[33]

图 4 基于 GPU 运行的城市气流流动的计算[33]

具体实现的时候,利用像素程序结合纹理操作完成整个运算.这里以密度(ρ)的扩散过程为例,说明具体的实

现方法.如果采用 7 点格式离散其扩散方程,并采用显式迭代,则整个求解形式如下:

$$\rho'_{i,j,j}=(\rho_{i-1,j,k}+\rho_{i+1,j,k}+\rho_{i,j-1,k}+\rho_{i,j+1,k}+\rho_{i,j,k+1}+\rho_{i,j,k-1}+\alpha\rho_{i,j,k})\beta\ .$$

为了减少绘制的次数,避免绘制句柄切换导致的性能损失,我们采用了平铺三维纹理的方式,即将整个三维计算域平铺到二维空间,如图 5 所示,因此整幅纹理即代表了整个密度分布,一个纹元即代表一个节点的密度值.在像素程序中利用纹理位置偏移即可完成对周围 7 个节点的采样操作,从而完成整个计算.

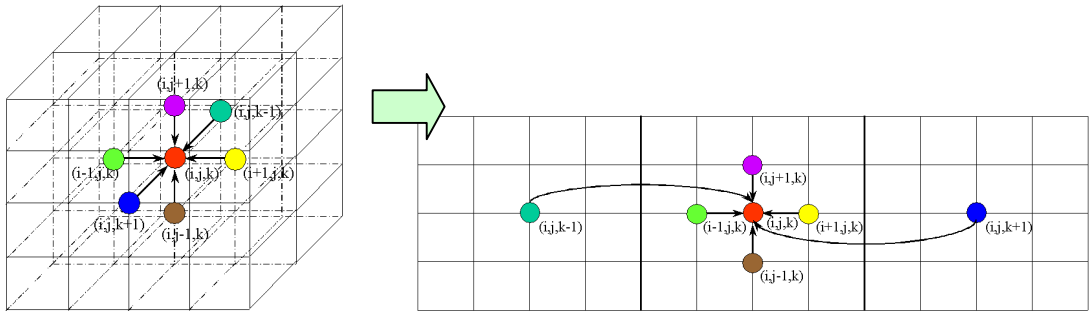

Fig.5 Solving discrete equation in fragment program

图 5 像素级求解离散方程

2.2 数据库操作

将 GPU 运用于数据库的研究工作最近获得一些可喜的结果.美国北卡罗莱那大学在这一研究中用 GPU 实现数据库的某些基本操作取得乐观的效果[34].这些操作主要是对数据库记录实行各类查询,其中包括关系查询(relational query)、合取选择(conjunctive selection)、聚集操作(aggregation)等.GPU 对数据库的操作是将 GPU 本身的纹理存储用作记录数据库的属性,而利用子素处理器对纹理实行并行操作且作检测,并利用深度缓冲区(depth buffer)和模板缓冲区(stencil buffer)的深度检测和模板检测协助其数据库操作的执行.举例来说,对数据库某属性项进行范围查询(range query)的操作过程是首先将纹理存储(数据库项)拷贝到深度缓冲区,然后调用“深度限界检测”的 GPU 硬件功能去“绘制”具有下界深度的长方形,最后在模板缓冲区生成(留下)通过了“深度限界检测”的结果,其结果的每个置了 1 的 Pixel 对应于符合条件的数据库项.

对于数据库各项操作的研究工作是在 Nvidia GeForce 5900 GPU 上进行的,与之对比测试的是具有双 CPU、2.8GH 的 Intel Xeon,并在具有针对该 CPU 的 SIMD 并行的优化编译下进行.测试的数据库记录数量达到百万量级.其测试结果表明,对于大部分类型数据库操作,GPU 相对于 CPU 具有明显的优势,而对于某些少部分操作,则 CPU 占先.其中对于半线性关系查询(semi-linear query)及选择查询(selective query),GPU 比基于 CPU 的实现具有一个数量级的加速效果.这种加速主要来自于几个方面:一是子素处理器的多像素并行处理(GeForce 5900 有 8 个像素的并行,而最新的 GeForce 6800 有 16 个像素的并行)以及每个像素上具有的向量计算——选择查询主要得益于前者,而关系查询则得益于二者;第 2 方面,GPU 的高度流水线结构,使得多个素元可以在流水线上同时得到处理,因此对于多个属性的同时查询特别有好处;第 3 方面,GPU 的深度剔除硬件功能使得部分子素(fragments)可以在流水线中较早地被排除在外,免除进入像素处理和深度检测,从而大大地提高了处理效力;第 4 方面,在 GPU 实现的这些查询操作中,不存在分支,如误预期(misprediction)操作.此操作在 CPU 上具有很大开销,例如,在 Pentium IV 上每个分支误预期操作会消耗 17 个时钟周期[35].

第 2 类数据库操作如“第 k 个最大数查询”等,由于这种操作只能利用 GPU 的部分而不是全部的优化功能,其实验结果仅具有 2~4 倍的性能改进.这部分功能来自于像素处理器的并行性,其 80%的时间花费在像素处理的计算开销上,这意味着其他方面如数据传输等开销很小.

由于 GPU 本身结构的局限性,并不是所有的数据库操作都能在 GPU 上运行而达到良好的效果.实验结果表明,累加算法在 GPU 上运行比基于 CPU 实现的算法要慢.其主要原因是,子素处理器不支持整数算术,因此在子素程序中诸如检测第 i 位的简单工作就需要 5 条指令去完成.这种情况下,CPU 对于 GPU 的主频优势(2.8GH 相

对于 450MH)便可以显现出来,再加上其 CPU(Intel Xeon)本身每个处理器亦具有 4 个 SIMD 的处理器并行工作.

2.3 频谱变换和滤波

频谱变换在一维或二维(特别是二维)的空间/时间域上进行,均可以在其离散的函数上实施统一的计算操作,即使用选定的核作卷积运算.为此,频谱变换和滤波是比较适于作 SIMD 运算和流处理的计算模型.

Moreland 等人[36]报道了他们利用常规算法在 GPU 上实现了快速傅立叶变换(FFT).在其变换过程中,为了避免改变写入中间存储,他们设计了特殊算法和数据结构以避免对数据的排序操作.利用两个纹理存储之间的交换(通过帧存储器),使得不同趟的像素程序相继操作在前趟程序的执行结果上.在求卷积的过程中,FFT 的实部和虚部分别占用纹理存储的一半空间.其运行测试的结果表明,在 Geforce5800 Ultra 上实现的快速傅立叶变换与 GPU(1.7 GHz Intel Zeon)上执行的优化 FFTW 具有相当的效率,对 512×512 的图像可以在 1 秒之内实施正向或反向的傅立叶变换.

小波变换[37]近年来受到了广泛的关注.作为多分辨率表示和分析的重要数学工具,它在时域和空域均具有很好的局部性能,有利于表示有限空间分布的高频信号如对图像作多分辨率分解和表示,在边缘检测、图像分割、图像压缩等方面具有重要应用.早在 1999 年,Hopf 和 Ehl[38]便利用 SGI Octane 工作站的图形处理器 MXE 基于 OpenGL 实现了 Haar 和 Daubechies 小波的变换及反变换用于边缘检测.其测试效率比纯软件实现的小波变换提高了 2~5 倍.

根据最新的报告,香港中文大学王剑清等人在最新发展的 GPU 上实现了完整的小波变换[39,40],包括小波多尺度分解变换和重建逆变换,二者使用类似的实现框架完成.该实现用图像处理作为应用实例,实现了小波变换对图像的多分辨率表示,图像基于频谱变化的变形、图像压缩(JPEG2000 标准)等.此变换同样在子素处理器上完成.在小波变换分解的过程中,对于每个尺度(分辨率)的卷积计算都是在存储于纹理上的前一个尺度计算结果的基础上进行,并生成新的结果输出到另一纹理上,如此反复转换,直到所需要的精度满足为止.对于二维函数的变换,按常规分解到两个方向上轮流进行,其变换所分解成的子带自始至终占据设定分辨率的纹理存储.离散小波变换在算法执行时分别在低趟(low pass)和高趟(high pass)两个层次上交错进行,并有选择地使用源数据,然而,子素处理器不允许对纹理存储进行存储调整(重新安排数据如分类等),因此有效地实现对读(从源纹理)写(帧存储)存储的寻址是变换获得成功的关键.该实现在解决这一问题时利用了子素处理器纹理存储的间接寻址功能,用一个中间纹理作间接寻址存储器,变换时,每次根据当前分辨率及其趟数类别,对每个变换的像素求取对应源数据的地址,存入其中间纹理.离散小波变换(DWT)已用于图像压缩标准 JPEG2000[34]并且是该压缩编码的主要开销部分,当基于 GPU 的 DWT 引擎引入这一标准(GPEG2000 Part-1 标准,即 ISO/IEC 15444-1)的实现时,其测试结果证明编码速度可以得到显著提高.在 Intel Pentium Ⅳ 2GH CPU 和 GeForce5900 Ultra GPU 平台上用 Cg 基于 OpenGL 的运行结果表明,只有当图像大小低于 400×400 时,GPU 的运行性能才低于软件运行的性能,这时,GPU 起始化及数据传输等源于 OpenGL 的开销(不依赖于图像大小,基本为常数)占据了开销的大部分.当源图像尺寸增大时,GPU 运行开销的增加很慢,而软件运行的开销却增长很快.以至于当图像大小为 2048×2048 时,基于 GPU 运行的开销比软件运行的开销具有几乎超过一个数量级的改进.

3 软件研究及其进展

以 Nvidia 和 ATI 为代表的 GPU 产品正处于不断改进的过程当中,正如前面所提及的,每隔半年左右,新一代的产品便会诞生,会增加更多的功能及提供更强的处理能力,但同时亦为用户使用这些新功能带来了诸多困难和问题.

由于 GPU 是以芯片产品提供给微机厂家的,而近年来 GPU 的产品更新速度可以说是大规模集成电路产品发展历史上最快的,由 GPU 芯片直接提供的高级而复杂的功能一下子提到用户的面前,而且频繁更新,使得无论是 PC 厂家,还是用户都显得措手不及.在这种仓促应付的情况下,GPU 的应用途径至今主要是通过图形 API(OpenGL 或 DirectX)扩充新 GPU 的功能,其扩充功能或由 GPU 厂家提供,或由 API 软件开发者提供.利用高级语言实现不依赖于具体硬件(GPU)及计算平台的编程自然是 GPU 使用的努力目标.为此目标努力而在绘制

语言(shading language)和实时绘制语言方面所作的研究一直在进行之中.但针对 GPU 的通用计算,非图形用户最终需要摆脱图形流水线及其绘制的概念,从流处理机的角度利用高级语言编程,这方面的研究工作在 2004 年开始具有新的进展.

GPU 所提供的可编程功能以顶点处理器和像素处理器的操作形式完成,由每个处理器执行用户定义的汇编级的绘制程序(shader program)[10],对流数据(顶点/像素)执行绘制程序的操作.标准图形界面用户在使用这些 GPU 的新功能时一般通过 GPU 设计者或用户界面的设计者以扩充函数库的形式提供给用户.

OpenGL 作为事实上的工业标准已为学术界和工业界所普遍接受,因而绝大部分与图形有关的应用产品一直以 OpenGL 作为实现界面.对于使用 GPU 的扩充函数,OpenGL 中包括了 GPU 设计者(如 Nvidia)以及 OpenGL"架构委员会(ARB)"所扩充的函数.DirectX 作为微软视窗的标准,其图形界面 Direct3D 从新世纪开始亦得到广泛接受和应用,特别是作为游戏软件的实现界面.

为适应 GPU 应用的需求,DirectX 则根据 GPU 新产品功能的扩充与进展及时地定义新的版本以扩充 Vertex Shader 和 Pixel Shader 的新功能,DirectX 软件接口所提供的功能几乎与 GPU 提供的功能同步.对于熟悉 OpenGL 或 DirectX 的用户或软件专业用户来说,直接使用其扩充的接口软件应该是比较好的选择,因为这样可以从底层更灵活地控制和对 GPU 编程.而对于使用 GPU 进行通用计算的用户,当前这一方法似乎是唯一的选择,尽管这一方法对使用者来说繁琐而困难,编程者往往需要考虑硬件结构的许多具体问题.

3.1 高级绘制语言及实时绘制语言

使用类似于 C 的高级语言对 GPU 直接编程一直是图形界努力追求的目标,也是将来应该达到的目标.在这方面,绘制语言(shading language)以及实时绘制语言的研究一直针对此目标在不断改进和完善.使用绘制语言及高级语言进行绘制编程可以方便用户书写各种不同功能的绘制程序以及对其绘制程序提供各种控制,以使 GPU 硬件的具体功能对于用户而言具有透明性.绘制程序(shader)设计的思想源自于早年 Pixar 设计的 RenderMan[41~43]绘制软件.此软件多年来广泛应用于好莱坞电影制作的绘制.关于新型标准绘制语言的研究工作,近年来具有较大影响的是:OpenGL shading language[44,45],斯坦福大学的 RTSL(real-time shading language)[46,47],Microsoft 的 HLSL[48](high-level shading language)以及 Nvidia 的 Cg[49].尽管还未形成统一的绘制语言,这些语言的研究和应用为用户提供了直接基于 API(OpenGL 或 DirectX)编程的较为方便和高层次的工具,尤其是 Nvidia 的 Cg.

3.2 流处理机编程环境及工具

然而,使用绘制语言编程仍然存在着不少缺陷,对使用者来说仍然是一件十分麻烦的工作.首先,用户必须编写控制图形流水线的许多任务,如分配纹理存储、读入绘制程序、构造图形素元等,为此,用户对最新的 API 以及图形处理器硬件的特点与限制需要有详细的了解.此外,用户仍然需要利用纹理、三角形等图形素元表达他们的算法,这就使得 GPU 的通用计算编程工作仍然局限于只能由资深的图形开发者进行.如果这一问题不能很好地解决,GPU 用于通用计算的发展和普及将会受到很大的限制.因为,熟悉 GPU 图形编程的人毕竟太少,而这些资深的图形开发者对通用计算的各个应用领域又比较生疏.为解决这一问题,当前的努力方向主要是将 GPU 的结构纯粹纳入流处理机的模型而以高级语言编程,使得程序说明、运算操作、模块化定义等一系列运算和控制规范化,使用户在实现高效率利用 GPU 的同时,不必考虑 GPU 的具体图形结构.这一研究工作的代表是最近发表于 ACM SIGGRAPH2004 的斯坦福大学的 Brook-for-GPU 系统[50]和加拿大 Waterloo 大学的 Shader Algebra 系统[51].由于 Brook-for-GPU 系统建立在扎实的前期工作基础之上并对于解决以上问题更具有代表性,而且 Brook 是开放源代码的系统,本文在此对该系统作一简单介绍.

Brook 原先是为诸如斯坦福大学的 Merrimac 超级流处理计算机[52]、MIT 的 MITRaw 处理机[53]等流处理机开发的语言.此语言由于是为流处理机所设计,因而具有应用于 GPU 流处理机模型的良好结构与基础,有利于扩充该语言作为通用计算的软件工具.Brook 的设计本身通过流(stream)数据类型定义流数据,作用在数据上的操作或计算称为核(kernel).核作用于流数据结构实现并行操作,以保证较高的计算效率(arithmetic intensity,即计算操作量相对于带宽的比率).Brook 具有很好的可移植性(portability),除了基于 OpenGL,DirectX 针对

Nvidia 和 ATI 等的 GPU 编程以外,亦可对普通 CPU 编程.Brook 的 GPU 系统包括编译器及运行系统.其编译器完成从 Brook 源程序到 Cg Shaders(Cg 绘制程序)源程序的编译工作,然后特定 GPU 厂家提供的 Shader 翻译程序再将 Cg Shaders 编译成 GPU 的汇编程序.Brook 运行系统是一个与 GPU 结构无关的程序库,它允许由编译器产生的编码使用该库函数去运行 Kernels.Brook-for-GPU 的最大特点表现在两方面:首先,作为面向流处理机的语言,它允许用户定义两种截然不同的同构数据集,一种是“输入流(input stream)”类型,另一种是“聚集流(gather stream)”类型.前者允许 Kernel 以规则顺序读入但不可重用,而后者则可以随机读写,并可重用.另一方面,在使用 Kernel 时,对于流数据和向量数据的编程明显地区分开来,对流数据可以有任意的函数计算,而对向量仅有简单的数学操作;向量运算需要大的临时存储用于对大规模向量寄存器文件的读写,而与此相对比,Kernel 用局部寄存器作临时存储,并且只在最后得到结果时才写进内存,以此提高计算效率.

4 结论与展望

在持续高速发展多年之后,GPU 仍然处在高速发展的过程当中.最新的 GPU,如 Nvidia Geforce 6800、ATI Radeon X800,其主频为 500M,纹理达到最大 4096×4096 的容量,具有平均每秒 4.5Gfloats 纹理带宽.其 Pixel Shader 自 Geforce FX 以来即具有了浮点计算能力,这对于 GPU 的通用计算是一个极大的推动,因为仅有定点运算的计算能力会是十分有限的.Pixel Shader 部件可编程指令的数目,从 Geforce3 的 96 条提高到几乎没有限制,大大地提高了程序处理能力.如前所述,通用计算的主要平台是 Pixel Shader 部分.可以预见,Pixel Shader 部分将会不断增加和加强新的功能.

4.1 GPU硬件及结构

对通用计算来说,最关键的功能是浮点的精度.目前 GPU 已经提供 IEEE 单精度浮点标准,这在很大程度上已能满足相当一部分通用计算的需求.但是,如果要真正满足许多应用领域计算的需求,IEEE 双精度浮点运算是必需的条件.很难说 GPU 在近期会达到这一条件,因为我们认为这一发展需要在以下条件满足时才有可能:一是可以充分证明 GPU 确实能在通用计算领域担负起举足轻重的角色;二是软件发展可以使用户(至少是通用的数学库)方便地使用 GPU 作通用计算.就 GPU 当前担任的主要角色——图形绘制,例如计算机游戏等而言,单精度浮点似乎已经足够.

GPU 在结构上限制通用计算的最大问题是缺少大容量局部存储.通用计算的主要平台像素处理器内用于通用计算的存储是纹理存储器,当前最大容量为 256M,这对于大量的通用计算应用来说是一个很大的限制,因其无大容量外存储器可用.虽然局部存储(纹理)还可以继续扩充以满足计算需求,但似乎不可能大量扩充.GPU 的计算处理能力从很大程度上来源于它将主要的集成电路功能贡献于流处理计算.而对 CPU 来说,如 Intel Xeon 微处理器虽具有相当于 ATI R300(1.1 亿晶体管)的集成度(1.08 亿),其中 60%却要用于快存.因此,如果 GPU 局部存储大量扩充,或许再加上 GPU 操作能力的扩充,这无异于向一个独立 CPU 的方向发展,这与 GPU 本身用于图形绘制的初衷是相悖的,亦相应增加了 GPU 用于图形绘制的成本.

解决存储问题的另一方向是增加 GPU 与主机接口的带宽,我们在文献[12]中亦指出了这一想法.自从 1996 年,图形接口由 Intel 的 AGP 代替了 PCI 接口以后,到 2002 年,AGP3.0(AGP 8X)达到 2.1GB/s 的传输带宽,带动了 3D GPU 及其显卡产品的蓬勃发展.但是如果将 GPU 及其主机内外存储器用于通用计算,GPU 与主机之间的图形接口无疑将会成为瓶颈.近年来,Intel 提出了 PCI Express 新型总线标准,传输带宽扩大到 8GB/s(PCI Express X 16),具有双向同等的传输率.Nvidia 和 ATI 即将推出使用 PCI Express X 16 接口的新一代显卡产品.这为 GPU 访问主机内存提供了改善的条件.然而对于 GPU 的通用计算来讲,流数据处理的基本结构将很难有大变化.其原因在于,无论是像素处理器还是顶点处理器,访问主机内存的代价都实在太大.如果频繁访问主机内存,则不符合巧用 GPU 的像素/顶点处理器作通用计算的初衷.

4.2 软件界面及工具

从上一节的介绍可知,基于 GPU 通用计算的软件研究及其开发已经取得很多进展,然而软件方面所存在的问题比硬件更甚.当前 GPU 在通用计算领域研究和应用方面主要有两大障碍:一是缺少对应用算法和图形绘制

均熟悉的研究开发者;二是 GPU 编程仍然过于繁杂.对于前者,如何将通用计算中各种复杂算法转换到顶点/像素处理器可编程算法是关键中的关键.拿主要平台的像素处理器来说,其计算模型以及操作的资源有限,功能均在很低层次,且受到流处理的读写种类等诸多限制,将通用计算的各种算法转换为像素处理编程是一件颇为不易的工作,需要由算法设计者精心考虑、充分利用有限资源和有限操作功能.除非设计者对其应用问题计算算法以及 GPU 图形处理过程有相当深入的了解,否则完成这一工作是很困难的.

在 Cg 或者扩充的 C 语言编程环境下,基于 OpenGL 或 DirectX 编程在相当一段时间内将会是 GPU 编程的主要方式.就 API 而言,OpenGL 作为事实上的工业标准,多年来已广泛为学术界与工业界所接受,而 DirectX 作为广泛用于游戏产业的 API,具有强大的企业背景和技术支持,在支持 GPU 新产品的过程中亦步亦趋,往往具有不亚于甚至优于 OpenGL 支持的运行效果[50].近年来,虽然有众多的生产厂家竞争 GPU 这一具有庞大市场的产业,但目前基本上为 Nvidia 和 ATI 两家所垄断.两家产品在总体结构上基本上保持一致,而具体细节上却有差异,用户在编程时仍会有某些困惑.

如前所述,GPU 的高速发展受到相关应用领域的强大驱动.首先,大数据量的图形环境模型以及虚拟现实、计算机仿真、计算机游戏等实时需求向图形处理提出了越来越高的要求[54,55].单就游戏产业而言,它的庞大市场就足以使 GPU 以及相关软硬件的发展产生强大的动力,微软的 Xbox 使用常规的 GPU(从 Nvidia 到 ATI)便是明证.因此,GPU 无疑将会在近几年内得到进一步的高速发展,这就可为 GPU 用于通用计算提供更丰富的功能和更强的能力.业界有人预言,约 5 年左右,当 GPU 变成以通用计算为主,软件以流处理的管道式写法顺利推广之后,GPU 的集群服务器就可以构成以 GPU 为主的超级计算机.对于这一问题,或许在三五年内人们会作各种各样的尝试和努力,事实上已经有人在用多 GPU 做通用计算并行处理的研究工作[56,57].但我们认为,要真正达到使用 GPU 的集群服务器做超级计算机还有相当长的一段路要走,无论在硬件、软件方面尚需克服许多难题.在技术上,GPU 毕竟是面向 Graphics 的 GPU,如果不在设计上为通用计算提供更多的直接计算功能并为 GPU 访问大容量内外存储提供更高的带宽和结构上的改变,计算的真正通用性是很难实现的.更大的问题或许是在软件方面.在任何情况下,基于 GPU 的通用计算工具必须建立在底层图形软件功能的基础上,而恰恰是这些底层图形功能的统一界面还远未尘埃落定.从通用计算的角度来讲,以斯坦福大学为代表的研究工作,包括绘制语言及流处理语言的研究是技术上最先进和合理的.事实上,在可以预见的一段时间内,人们还只能像现在这样用 C 或 Cg,基于 OpenGL 或 DirectX 对 GPU 编程.研究领域和工业界在选择使用 OpenGL(工业界大部分参加了 OpenGL“架构委员会”,微软却于近期退出)或 DirectX(从 2000 年以后变得越来越强势,但只限于 Windows)以及选择 C 或 Cg 编程时常常感到无奈.这就意味着,基于 GPU 集群的超级计算机是否能在工业界成为真正通用计算的工具将取决于诸多因素:GPU 硬件结构的更新能否为面向通用的计算作进一步的努力;工业界能否为使用 GPU 可编程功能提供软件界面及其工具作一致的努力;高层软件在面向流处理机通用计算方面为用户提供编程环境及调试工具的进展能有多快等.

致谢 柳有权实现了 GPU 上的流体模拟计算并在本文写作过程中进行了有益的讨论;李奎宇、吴金钟帮助打印了中文初稿,在此表示感谢.

References:

[1] Macedonia M. The GPU enters computing's mainstream. IEEE Computer, 2003,36(10):106~108.

[2] Hopgood FRA, Duce DA, Gallop JR, Sutcliffe DC. Introduction to the Graphics Kernel System (GKS). Academic Press, 1983.

[3] Enderle G, Kansy K, Pfaff G. Computer Graphics Programming: GKS—The Graphics Standard. Berlin: Springer-Verlag, 1984.

[4] Howard TLJ, Hewitt WT, Hubbold RJ, Wyrwas KM. A Practical Introduction to PHIGS and PHIGS Plus. Addition-Wesley, 1991.

[5] Clark JH. The geometry engine: A VLSI geometry system for graphics. In: Proc. of the SIGGRAPH'82. 1982. 127~133.

[6] Fuchs H, Poulton J. Pixel-Planes: A VLSI-oriented design for a raster graphics engine. VLSI Design, 1981,2(3):20~28.

[7] Eyles J, Austin J, Fuchs H, Greer T, Poulton J. Pixel-Plane 4: A summary, advances in computer graphics hardware II. In: Eurographics Seminars Tutorials and Perspectives in Computer Graphics. 1988. 183~208.

[8] Fuchs H, Israel L, Poulton J, Eyles J, Greer T, Goldfeather J, Ellsworth D, Molnar S, Turk G, Tebbs B. Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In: Proc. of the SIGGRAPH'89. 1989. 79~88.

[9] Molnar S, Eyles J, Poulton J, Greer T. PixelFlow: High-Speed rendering using image composition. In: Proc. of the SIGGRAPH'92. ACM Press, 1992. 231~240.

[10] Lindholm E, Kilgard MJ, Moreton H. A user-programmable vertex engine. In: Proc. of the SIGGRAPH 2001. Los Angeles, 2001. 149~158.

[11] Owens JD, Dally WJ, Kapasi UJ, Rixner S, Mattson P, Mowery B. Polygon rendering on a stream architecture. In: Proc. of the Eurographics/SIGGRAPH Workshop on Graphics Hardware. 2000. 23~32.

[12] Wu EH, Liu YQ. General purpose computation on GPU. Chinese Journal of Computer Aided Design & Computer Graphics, 2004, 16(5):601~612 (in Chinese with English abstract).

[13] Govindaraju NK, Sud A, Yoon SE, Manocha D. SWITCH: Parallel occlusion culling for interactive walkthroughs using multiple GPUs. Technical Report, TR02-027, UNC-CH, 2002.

[14] Govindaraju NK, Redon S, Lin M, Manocha D. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In: Proc. of the Eurographics/SIGGRAPH Workshop on Graphics Hardware. 2003. 25~32.

[15] Sud A, Otaduy MA, Manocha D. DiFi: Fast 3D distance field computation using graphics hardware. In: Proc. of the Eurographics, 2004.

[16] Tomov S, McGuigan M, Bennett R, Smith G, Spiletic J. Benchmarking and implementation of probability-based simulations on programmable graphics cards. Computers & Graphics, 2005,29(1).

[17] Larsen ES, McAllister D. Fast matrix multiplies using graphics hardware. In: Proc. of the Supercomputing. 2001. 55~60.

[18] Thompson CJ, Hahn S, Oskin M. Using modern graphics architectures for general-purpose computing: A framework and analysis. In: Proc. of the Int'l Symp. on Microarchitecture. 2002. 306~317.

[19] Krüger J, Westermann R. Linear algebra operators for GPU implementation of numerical algorithms. ACM Trans. on Graphics, 2003,22(3):908~916.

[20] Hall JD, Carr NA, Hart JC. Cache and bandwidth aware matrix multiplication on the GPU. UIUCDCS-R-2003-2328, Champaign: University of Illinois at Urbana-Champaign, 2003.

[21] Rumpf M, Strzodka R. Using graphics cards for quantized FEM computations. In: Proc. of the VIIP 2001. 2001. 98~107.

[22] Harris MJ, Coombe G, Scheuermann T, Lastra A. Physically-Based visual simulation on graphics hardware. In: Proc. of the Graphics Hardware 2002. 2002. 109~118.

[23] Li W, Wei XM, Kaufman A. Implementing lattice Boltzmann computation on graphics hardware. The Visual Computer, 2003, 19(7-8):444~456.

[24] Bolz J, Farmer I, Grinspun E, Schröoder P. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. ACM Trans. on Graphics, 2003,22(3):917~924.

[25] Goodnight N, Woolley C, Luebke D, Humphreys G. A multigrid solver for boundary value problems using programmable graphics hardware. In: Proc. of the Graphics Hardware 2003. 2003. 102~111.

[26] Harris MJ, Baxter W-V III, Scheuermann T, Lastra A. Simulation of cloud dynamics on graphics hardware. In: Proc. of the Graphics Hardware 2003. 2003. 92~101.

[27] Li W, Fan Z, Wei XM, Kaufman A. GPU-Based flow simulation with complex boundaries. Technical Report, 031105, Computer Science Department, SUNY at Stony Brook, 2003.

[28] Kim T, Lin MC. Visual simulation of ice crystal growth. In: Proc. of the SIGGRAPH/Eurographics Symp. on Computer Animation. 2003. 86~97.

[29] Lefohn AE, Kniss JM, Hansen CD, Whitaker RT. Interactive deformation and visualization of level set surfaces using graphics Hardware. In: IEEE Visualization. 2003. 75~82.

[30] Stam J. Stable fluids. In: Proc. of the SIGGRAPH'99. 1999. 121~128.

[31] Wu EH, Liu YQ, Liu XH. An improved study of real-time fluid simulation on GPU. Journal of Computer Animation & Virtual World (invited paper of CASA2004), 2004,15(3-4):139~146.

[32] Liu YQ, Wu EH, Liu XH. Real-Time 3D fluid simulation on GPU with complex obstacles. In: Proc. of the Pacific Graphics 2004. 2004.

[33] Liu YQ, Wu EH, Liu XH. Fluid simulations on GPU with complex boundary conditions. In: ACM Workshop on General-Purpose Computing on Graphics Processors (GP2). 2004.

[34] Govindaraju NK, Lloyd B, Wang W, Lin M, Manocha D. Fast computation of database operations using graphics processors. In: Proc. of the SIGMOD. 2004.

[35] http://lava.cs.virgina.edu/bpred.html

[36] Moreland K, Angel A. The FFT on a GPU. In: Proc. of the Graphics Hardware 2003. 2003.

[37] Strang G, Nguyen T. Wavelets and Filter Bands. California: Wellesley-Cambridge Press, 1996.

[38] Hopf M, Ertl T. Hardware accelerated wavelet transformations. In: Proc. of the EG/IEEE TCVG Symp. on Visualization 2000. 2000. 93~103.

[39] Wang JQ, Wong TT, Heng PA, Leung CS. Discrete wavelet transform on GPU. In: ACM Workshop on General-Purpose Computing on Graphics Processors (GP2). 2004.

[40] Wang JQ. Exploiting the GPU power for intensive geometric and imaging data computation [MPhil Thesis]. Chinese University of Hong Kong, 2004.

[41] Hanrahan P, Lawson J. A language for shading and lighting calculations. Computer Graphics, 1990,24(4):289~298.

[42] Upstill S. The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics. Addison-Wesley, 1990.

[43] Apodaca AA, Gritz L. Advanced RenderMan: Creating CGI for Motion Pictures. Morgan Kaufmann Publishers, 2000.

[44] Kessenich JD, Baldwin RR. OpenGL 2.0 Shading Language. 1.051 Ed, 2003.

[45] Rost RJ. OpenGL Shading Language. Addison-Wesley, 2004.

[46] Proudfoot K, Mark WR, Hanrahan P, Tzvetkov S. A real time procedural system for programmable graphics hardware. Computer Graphics, 2001. 159~170.

[47] Mark WR, Proudfoot K. Compiling to a VLIW fragment pipeline. In: Proc. of the Graphics Hardware 2001. 2001.

[48] Peeper C, Mitchell JL. Introduction to the DirectX 9 High-Level Shader Language. 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhlsl/html/shaderx2_introductionto.asp

[49] Mark WR, Glanville S, Akeley K, Kilgard MJ. Cg: A system for programming graphics hardware in a C-like language. ACM Trans. on Graphics, 2003,22(3):896~907.

[50] Buck I, Foley T, Horn D, Sugerman J, Fatahalian K, Houston M, Hanrahan P. Brook for GPUs: Stream computing on graphics hardware. ACM Trans. on Graphics, 2004,23(3):777~786.

[51] McCool M, Toit SD, Popa T, Chan B, Moule K. Shader algebra. ACM Trans. on Graphics, 2004,23(3):787~795.

[52] Dally WJ, Hanrahan P, Erez M, Knight TJ, Labont F, Ahn JH, Jayasena N, Kapasi UJ, Das A, Gummaraju J, Buck I. Merrimac: Supercomputing with streams. In: Proc. of the SC 2003. ACM Press, 2003.

[53] Talor MB, Kim J, Miller J, Wentzlaff D, Ghodrat F, Greenwald B, Hoffmann H, Johnson P, Lee JW, Lee W, Ma A, Saraf A, Seneski M, Shnidman N, Strumpen V, Frank M, Amarasinghe S, Agarwal A. The raw microprocessor: A computational fabric for software circuits and general purpose programs. IEEE Micro, 2002.

[54] Wu EH. Challenge of highly detail object modeling and illumination calculation in virtual reality (Keynote Speech). In: Proc. of the ACM/VRST2002. Hong Kong, 2002.

[55] Wu EH. Modeling and rendering of complex virtual environments (Keynote Speech). In: Proc. of the 3rd National Conf. on Virtual Reality and Visualization. Changsha: National University of Defense Technology, 2003(in Chinese with English abstract).

[56] Fan Z, Qiu F, Kaufman A, Yoakum-Stover S. GPU cluster for high performance computing. In: Proc. of the ACM/IEEE SC2004 Conf. 2004.

[57] Gulde R, Weeks M, Owen S, Pan Y. Parallel computing with multiple GPUs on a single machine to achieve performance gains. In: ACM GP2: Workshop on General Purpose Computing on Graphics Processors. 2004.

附中文参考文献:

[12] 吴恩华,柳有权.基于图形处理器(GPU)的通用计算.计算机辅助设计与图形学学报,2004,16(5):601~612.

[55] 吴恩华.复杂虚拟场景的造型与绘制(大会特邀报告).见:第 3 届全国虚拟现实与可视化学术会议(CCVRV 2003).长沙:国防科学技术大学,2003.