图形处理器低功耗设计技术研究

田 泽 张 骏 许宏杰 郭 亮 黎小玉
(中国航空工业西安航空计算技术研究所 西安 710077)

摘 要 图形处理器(GPU)以其强大的图形加速性能以及在通用计算领域的出色表现正在被越来越广泛地应用。但随着芯片规模和集成度的不断提升,单个GPU芯片的功耗已经高达376W,是高端通用处理器的2~3倍。高功耗带来的可靠性、稳定性以及芯片成本问题使"功耗墙"已经成为未来GPU设计过程中需要突破的关键问题之一。立足于体系结构层次,结合图形处理器的渲染流水线的结构特点,从深度测试和消隐、染色器数据通路、纹理映射和压缩、渲染策略、寄存器文件和片上Cache等角度描述了图形处理器的低功耗设计技术,并指出了GPU低功耗设计技术的进一步研究方向。

关键词 图形处理器,低功耗,渲染,Cache
中图法分类号 TP303 **文献标识码** A

Low-power Design Techniques for GPU

TIAN Ze ZHANG Jun XU Hong-jie GUO Liang LI Xiao-yu
(Aeronautics Computing Technique Research Institute, Aviation Industry Corporation of China, Xi'an 710077, China)

Abstract Graphic Processing Unit(GPU) is extensively used to accelerate the graphic processing and general purpose computing. But unfortunately, along with the increasing of chip scale and density, power consumption of single GPU reaches 376W, which is two or three times higher than a typical high-end general purpose CPU. Stability, reliability and cost problems brought by high power consumption had already made the "Power Wall" be a key obstacle which must be broken through. On architecture level, and combined with the structure characteristics of GPU rendering pipeline, this paper demonstrates the GPU low-power design techniques, such as depth test and elimination, shader data path, texture mapping and compression, rendering strategy, register file and Cache, and then indicates the further research content for GPU low-power design.

Keywords Graphic processing unit(GPU), Low-power, Rendering, Cache

1 引言

20世纪末,在不断进步的半导体工艺和多元化应用需求的双重推动下,具备快速构建2D/3D图形、场景和渲染能力的图形处理器GPU(Graphics Processing Unit)应运而生,并迅速成为各种嵌入式系统、PC、工作站、个人数字媒体产品(如智能手机、游戏机和平板电脑)中不可缺少的核心部件[28-30]。近几年来,通用图形处理器GPGPU(General Propose computing on Graphics Processing Unit)利用其计算密集型的硬件架构优势,在高性能计算领域异军突起。有人甚至断言,GPU将在不久的将来替代传统CPU。如Nvidia Tesla C106[19]的理论峰值浮点性能为933GFLOP/s,而Intel高端服务器至强系列中E5450处理器的理论峰值浮点性能只有48GFLOP/s,相差近20倍;C1060最高数据带宽可达102 GB/s,几乎比E5450的10.6GB/s高一个数量级。Nvidia最新推出的采用40纳米工艺的新一代GPU Fermi[16]集成了512个CUDA内核,约30亿晶体管,单/双精度浮点计算性能分别达到1.31TFLOPS/768GFLOPS[15],功耗达到376W;AMD(ATi)也推出了最新代号为RV870的GPU[18],其更是集成了1600个流处理内核,输出2.72TFLOPS的单精度浮点性能,功耗为276W。

性能的提升固然令人兴奋,但同时出现的"功耗墙"问题对设计人员提出了更大的挑战。随着芯片集成度越来越高,芯片表面温度也变得越来越高并呈指数增长,每三年芯片的功耗密度就能翻一番,功耗在GPU设计中已经成为与性能和面积同等重要的设计指标。虽然研究表明GPU的效能(每瓦性能)是CPU的5倍左右,但GPU的总体功耗仍然是CPU的2~3倍。而目前阵列式的众核GPU体系结构更是加剧了解决功耗问题的紧迫性。高功耗不仅仅意味着大量的能源消耗,而且热堆积和不断增加的功耗密度将造成系统稳

本文受2012总装预研基金(9140A08010712HK61095),中国航空工业集团公司创新基金(2010BD63111)资助。

田 泽(1965-),男,博士,研究员,主要研究领域为SoC设计方法学、航空专用集成电路等;张 骏(1978-),男,博士后,高级工程师,主要研究领域为计算机体系结构、微处理器设计、SOC设计和航空专用集成电路,E-mail:zhangjun2008@mail.nwpu.edu.cn;许宏杰(1981-),男,工程师,主要研究领域为航空总线、图形图像专用集成电路;郭 亮(1982-),男,工程师,主要研究领域为航空总线、图形图像专用集成电路;黎小玉(1983-),女,工程师,主要研究领域为航空总线、图形图像专用集成电路软件及验证。

定性问题，研究表明工作温度每增加 10 度，芯片的失效率将增加一倍。另外，为了缓解热堆积，不得不使用散热能力更加强大的封装材料、额外的散热装置和发热保护电路，这无疑增加了 GPU 的制造成本。过高功耗限制了 GPU 性能的提升，如果要进一步提高内核频率或增大片上缓存容量，会使 GPU 的功耗继续攀升，进而走入恶性循环。更糟糕的是，芯片对这些功耗的有效利用率低于 1%，其余功耗要么浪费在闲置的电路上，要么直接转化为热量。面对高功耗压力，低功耗设计技术已经成为未来 GPU 设计中的核心问题。

2 背景技术

2.1 低功耗设计技术

CMOS 数字集成电路的功耗主要来自动态功耗、短路功耗、静态功耗和漏电流功耗[17]。动态功耗是整个电路功耗的主要部分，但随着 GPU 集成晶体管数目的不断增加，降低短路功耗、静态功耗和漏电流功耗也成为微处理器低功耗设计中不可缺少的部分。低功耗设计可以从 GPU 设计的各个阶段入手，不同的阶段可以降低功耗的因素有所不同，在设计时考虑的重点也不同。

图 1 是不同抽象层次功耗优化的概率。按照抽象层次的不同，可以分为：系统级、结构级、寄存器传输级、逻辑门级、版图级和电路级。每个级别可以达到的低功耗设计效果也完全不同，抽象层次越高，表明在数字系统的设计中越早地进行低功耗设计，因此在系统级和结构级进行低功耗设计效果较为明显。系统级低功耗设计和热管理方法主要考虑算法、软硬件分工等方面，研究的重点是如何控制芯片进行工作，从而达到降低功耗的目的。结构级主要考虑调度、硬件资源分配、操作数交换等方面。对 GPU 来说，在系统级和结构级主要考虑在深度测试和消隐、染色器数据通路、纹理映射和压缩、渲染策略、寄存器文件和片上 Cache 等方面进行低功耗设计。

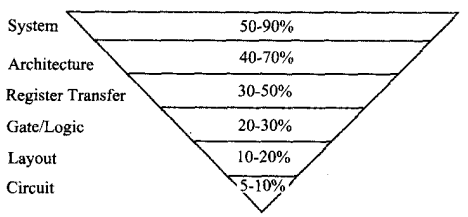

图 1 不同抽象层次功耗优化概率

2.2 图形处理器的渲染流水线

GPU 通常由主机接口、2D 加速模块、3D 加速模块、存储控制模块和显示控制模块 5 部分构成，其中最关键、功耗最大的是 3D 加速模块，它通常占到整个 GPU 芯片面积的 90%以上。因此，GPU 的低功耗设计主要是针对 3D 加速模块的低功耗设计。

3D 加速模块包含一条或多条渲染流水线，用来完成几何图形的构建、坐标变换、光栅化、像素染色、纹理映射和片段处理等一系列 3D 图形加速处理过程，图 2 是一个典型的 GPU 渲染流水线的流程图。

如图 2 所示，典型的渲染流水线包括 10 个阶段。顶点染色级接收来自主机接口的顶点信息进行顶点染色，包括完成顶点、纹理坐标、光栅位置、法向量、光源位置和聚光灯方向向量的旋转、平移和缩放等几何操作；完成顶点坐标处点的光照计算以及模型视图矩阵和纹理矩阵的变换和堆栈操作。图元装配级接收经过变换的顶点流，将基本图元通过一定的算法进行装配，得到点、线和三角形等各种图元。平面剪裁级使用客户指定的剪裁平面对世界空间中的图元进行剪裁，以决定哪些图元被保留、哪些图元被舍弃。投影变换级定义一个视景体，并进行投影变换，视景体决定了图元是以透视投影还是正投影的方式映射到屏幕上。三维剪裁阶段使用视景体定义的 6 个平面将位于视景体之外的所有图元进行裁剪。视窗变换是将位置性光源的坐标、光栅位置坐标、图元顶点坐标进行齐次坐标变换后，再将所得的坐标与视窗变换矩阵相乘，将窗口中的图形信息送到视口中，即将视景体内的物体显示在二维的视口平面。消隐主要实现的功能是设置三角形面的正面和反面，以及剔除设置的面，从而减轻隐藏面的计算量。光栅化阶段完成由连续的直线和三角形到屏幕上离散的像素点之间的转换功能，即完成连接直线的两个顶点间像素点和三角形内部像素点的扫描和填充。像素染色阶段完成像素值的放缩、偏移、映射、截断，以及像素的光照计算、纹理贴图及雾处理操作。最后还要进行片段处理，选择要最终写入帧缓存中的片段，并根据条件改变帧缓存中的值。这些操作主要包括片段测试、混合、抖动、逻辑操作、累积缓冲区以及帧缓存清除和屏蔽等。

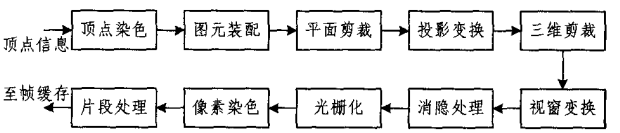

图 2 典型的 GPU 渲染流水线的流程图

相对于控制电路来说，低功耗设计应用于数据通路、片上存储器的设计和访问策略以及处理流程优化的效果更好。因此，GPU 的低功耗设计应重点着眼于计算和数据密集的顶点、像素染色器以及整个渲染流程的优化。以下从深度测试和消隐、染色器数据通路、纹理映射和压缩、渲染策略、寄存器文件和片上 Cache 等角度对图形处理器的低功耗设计技术进行描述。

3 相关研究工作

在最近的一些低功耗 GPU 设计中，DVFS 是一项被广泛使用的技术。文献[20]提出监视顶点和片段引擎间的片段队列，从而在两个引擎上使用 DVFS 策略。当片段队列满时，前端的顶点引擎就可以降低工作电压和频率，因为此时片段队列已经不能缓存更多的片段数据，从而降低了顶点引擎的功耗。文献[21]描述了在几何阶段、三角形建立和染色阶段的不平衡性，并提出了基于签名的工作负载预测机制，该机制能够根据帧历史和当前帧属性对后续帧的工作负载情况做出估计，以便根据预测的结果应用 DVFS 策略，对需要的工作电压和频率做出调整，从而降低 GPU 的整体功耗。文献[22]针对手持设备提出了一种低功耗 GPU 体系结构，该结构具备 3 个功耗域，并能够根据历史预测结果对 3 个功耗域中的电压和功耗独立地做出调整。

另外，有一些研究致力于通过减少外部存储器访问来降

低功耗。文献[23]提出了一种具有预取功能的纹理 Cache 用来进行纹理映射，从而减少片外存储器的访问数量，降低功耗。文献[17,20]提出了双线性平均 mipmap 映射算法作为一个低开销的纹理过滤机制来节省存储器带宽，从而降低功耗。相比之下，高端图形系统上使用的三线性 mipmap 映射需要访问两个 mipmap 层次上的 232 相邻单元，它需要访问更高分辨率 mipmap 层次上的 4 个纹素。利用这些信息，可以计算 n 层次上的一个双线性过滤和一个 $n+1$ 层次上的最近相邻过滤，还有介于这两个层次之间的一个线性过滤。文献[13]提出了一种叫做 FlipQuad 的低开销凡走样机制。这种机制基于旋转网格超采样模式(RGSS)对像素边界位置进行采样，采样模式是相邻翻转、相隔相同的。这种机制简化了相邻像素的采样共享，并且这种采样模式对每个像素来说只需要两个采样。这种方法虽然需要增加额外的片上存储器，但是能够大量减少电容性 I/O 引脚上传输数据的总量和对片外存储器的请求数量，总体来说仍然能够大幅地降低功耗。

还有许多研究集中在通过门控的方式来降低功耗。文献[24]在一个容量可变的 I-Cache 上应用了一种叫做门控电压的低功耗策略，即 DRI-i_Cache。DRI-i_Cache 挖掘了不同应用内和之间 I-Cache 利用率的变化，进而选择需要的 I_Cache 容量。在此基础上，不需要使用的 Cache Cell 上的电压就可以通过门控策略进行控制，从而消除漏电流、降低功耗。文献[25]注意到当 Cache 数据行刚进入 Cache 后会被频繁地使用，然后在该 Cache 行被替换出去前会有一段“死时间”，在“死时间”期间内该数据既没有被使用，也没有被替换 。文献[26]提出在 Cache 保存数据期间周期性地将 Cache 行切换为低电压模式。这种策略在小幅性能影响的情况下能够大幅减少 Cache 的漏电流。文献[27]提出一种预测超时机制来动态预测空闲周期的长度，以便在空闲周期较短时控制电路不要进入门控方式下的低功耗模式。

4 图形处理器的低功耗设计技术

4.1 深度测试提前

通常来说，按照 OpenGL 流程，3D 图形处理器在段处理阶段进行深度测试。因此，像素是否可见要等到段处理结束后才能够确定。消隐只是标识出空间中图元的正反面，然后根据需要进行剔除，而并不会对空间中有多个图元相互重叠遮挡的情况进行处理，这意味着 GPU 将要计算并染色一些最终不可见的像素，导致了不必要的计算和功耗。

如图 3 所示，矩形 B 遮挡了矩形 A，导致矩形 A 中的部分像素最终不可见。为了改变这一缺点，提前深度测试已经被广泛使用[1]，例如在 Nvidia 的 Tesla[19] 体系结构中就使用了这种技术。该技术的思想是将深度测试提前到渲染流水线的中间进行，深度测试在光栅化操作后立即进行。

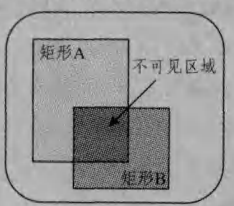

图 3 矩形 B 遮挡矩形 A 导致部分像素不可见

如图 4 所示，在深度比较逻辑中，将已经存储在深度缓冲器中的像素深度值与新产生的同一像素位置的深度值作比较，如果新产生的像素点是不可见的，则通过控制掩码屏蔽对深度缓冲器的写入操作；否则更新该像素点的深度值。同时，使用提前深度测试的结果，可以在纹理单元和片段操作模块使用时钟门控策略，深度缓冲器的写入控制掩码当作渲染流水线中的流水线寄存器门控时钟条件，这样可以避免这些片段的染色和纹理操作，同时还消除了不必要的纹理存储器和帧缓冲存储器的访问请求，从而减少了渲染处理器的动态功耗。对于典型的图形应用来说，提前进行深度测试能够使功耗降低 25%。

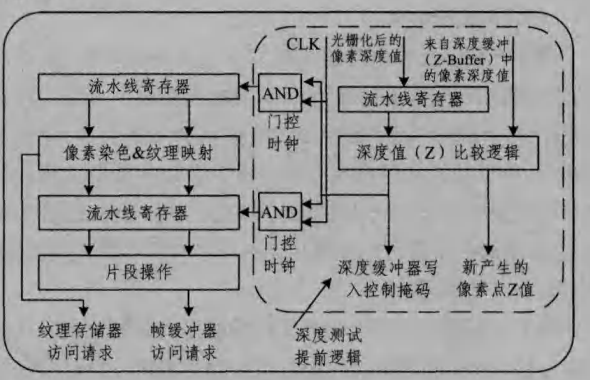

图 4 深度测试提前结构图

4.2 低功耗数据通路

在 GPU 渲染流水线中，剪裁和变换阶段涉及到的顶点、纹理、光栅位置、法向量和光源的平移、旋转、缩放等操作，在染色阶段涉及到的 Flat Shading、Gouraud Shading、Phong Shading 算法，以及在光栅化阶段涉及到的 Bresenham、Cohen-Sutherland 算法需要用到大量的复杂数学运算。其中执行较多的数学运算包括除法、倒数、平方根、平方等，这些功耗较大的操作使得 GPU 的功耗要比通用处理器大很多。根据文献[2]统计，在 GPU 渲染流水线占用的时钟周期中，有 80.95%用来执行除法操作、5.11%用来执行乘法操作、2.65%用来执行平方根操作。也就是说，处理这些数学函数的功耗占据了渲染流水线总功耗的绝大部分。因此，要降低渲染流水线的功耗，应该尽可能地减少执行这些复杂函数需要的时钟周期数。一个可行的方式是使用对数数据通路简化流水线中的数学运算过程。

对数数字系统简化了数学运算的过程，许多研究者试图将其应用到通用和专用处理器上。如表 1 所列，对数系统的最大优势在于功耗低、运算速度快和规模小，它能够将复杂的乘、除运算转换为加、减运算；将指数运算转换为与常数的乘法运算，在二进制系统中可以通过移位操作来实现。

表 1 对数系统的数学运算表达

数学运算类型	算术表达	对数系统的算术表达
乘法	$x \cdot y$	$X+Y$
除法	x/y	$X-Y$
倒数	$1/x$	$-X$
平方根	$\sqrt{x}$	$(1/2) \cdot X$
平方根倒数	$1/\sqrt{x}$	$-(1/2) \cdot X$
平方	x^2	$2 \cdot X$

然而，正如许多研究者指出的一样，对数系统的一个关键

问题是正常数据与对数数据之间相互转换时的数据精度偏差问题。自从 Mitchell 提出了二进制对数转换算法以来，对数运算的线性逼近算法已经被广泛研究，从而解决了如何在硬件实现中最小化精度偏差的问题[10-12]。

文献[6]开发了一个具有精确的对数-反对数转换算法的数学运算单元，如图 5 所示。首先，正常的输入数据信号被转换为对数值，然后按照表 1 中的数学运算操作进行计算，最后，对数运算结果通过反对数转换器转换为正常数据。为了最小化精度偏差，使用了一个 8 区域分段线性差值逼近算法来进行对数和反对数的转换。

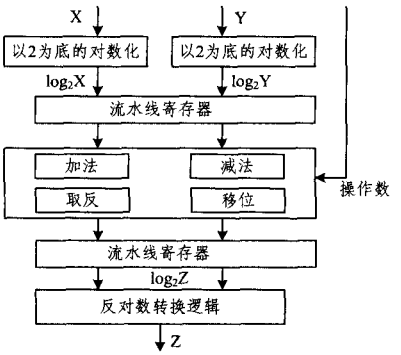

图 5　对数数据通路

在渲染流水线中，除了加法和减法操作，所有的定点变换、光照、光栅化和纹理映射都能够在对数数据通路中完成运算。研究表明，基于传统数据通路和对数数据通路渲染同一个 3D 图形，通过裸眼几乎分辨不出任何差距。正如前边提到的一样，对数的运算方法对于降低计算复杂度、提升运算速度和减少电路规模来说是很好的选择。相对于传统数据通路来说，对数数据通路能够使管芯面积减少到原先的 1/4，频率提升 3～4 倍，使功耗减少一半。渲染过程中需要进行大量的除法操作，如三角形建立、光栅化和纹理地址产生等，因此除法操作已经成为渲染系统的瓶颈，而使用对数数据通路是提升性能和降低功耗的较好选择。

4.3　低功耗纹理单元

要达到 3D 图形和场景的实时构建和渲染，GPU 通常需要很大的存储带宽。为了达到这一目的，嵌入式存储系统需要很宽的数据总线和高速时钟频率。直接通过 I/O 从片外存储器中取数会消耗大量的功耗，这是因为大量并行数据在电容性 I/O 上传输和存储器响应请求所导致的。因此，减少存储器请求的数量能够有效地降低功耗。而减少存储器请求的有效方法是利用访存的局部性，包括空间局部性和时间局部性。对于纹理数据，文献[1]提出了地址对齐逻辑(AAL)来挖据纹理空间的访问模式。

在双线性 MIPMAP 过滤中[7]，上层图像的一个像素需要使用图像同一细节水平的 4 个纹素，因此在过滤过程中发出的多条纹理数据请求有可能在部分数据上产生重叠现象。图 6 说明了 AAL 的模块图。纹理数据请求地址重叠检测逻辑发现并消除这些有数据重叠的请求，从而减少了纹理存储器请求的总数。纹理数据请求地址局部性检测逻辑将当前请求的纹理地址与先前的纹理地址做比较，只留下不同的纹理地址。它将最近使用到的纹素存储在流水线锁存器和比较器中。纹理数据请求地址局部性检测逻辑在结构上包含 6～8 个入口较小的纹理缓存器，需要注意的是纹素是存储在流水线中的锁存器里，而不是功耗很大的 SRAM 中。当纹素在空间上的重叠性和时间上的局部性被移除后，剩余请求的平均数量能够被减少至使用 AAL 前的 1/4 左右。

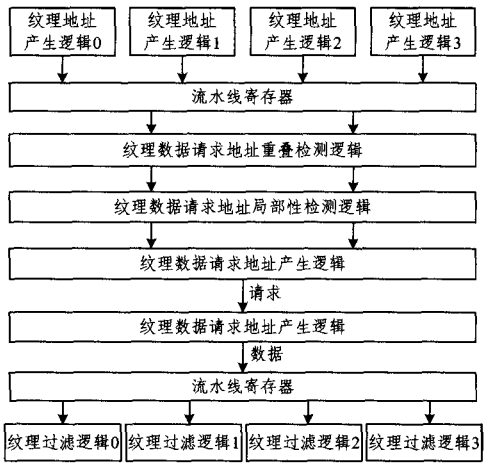

图 6　AAL 低功耗策略结构图

AAL 通过减少纹理存储器的访问数量降低了图形绘制的功耗需求。在集成 AAL 的情况下，由于存储器冲突，访问纹理存储器的平均时钟周期将会增加 10%左右。纹理存储器的功耗与每时钟周期激活的纹理存储器数量是成正比的。在 AAL 机制的帮助下，访存请求减少到原先的 25%，访问纹理存储器所需要的功耗能够被减少 68%。

4.4　低功耗分区渲染技术

传统的 GPU 一次性渲染所有场景，而不去考虑是否场景中的所有部分都需要被渲染。基于分片渲染(TBR-Tile Based Rendering)的思想是将一个场景分解为许多小的片段，并且分别对这些片段进行渲染，这样做除了能够提升渲染效率外，另一个重要的原因就是能够节省功耗。

如图 7 所示，TBR 的操作对象是屏幕上的多个小片段，基于片段渲染的 GPU 只需要部分小片段的数据，而不需要整个屏幕的数据。因此，基于片段渲染的 GPU 能够大大减少数据的传输交互。另外，基于片段的 GPU 只需要容量相对较小的片上存储器就能够满足存储渲染小片段所用的颜色值和深度值的需求，而不需要大容量复杂的片上 Cache 系统和数据压缩算法，这些都有助于降低功耗。

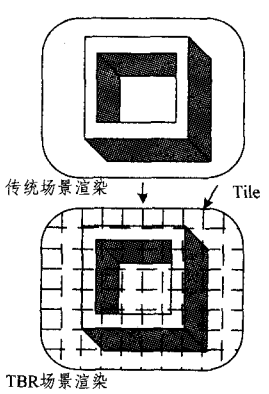

图 7　TBR 场景渲染原理

TBR 的另一个优势是隐藏面消除。传统的 GPU 首先填充所有多边形，而不考虑哪些多边形需要在后续处理中被隐藏。相比之下，TBR 技术首先验证是否所有的多边形都需要被填充。通常，每个像素点上只有距离屏幕最近的对象需要被处理。经典的 3D 游戏需要频繁地访问深度存储器和加载纹理数据，这样会完全占用存储器带宽，从而导致图形系统性能的下降。因此，TBR 技术只加载可见对象需要的纹理数据，而隐藏对象的相关颜色、纹理、亮度和深度属性的相关数据则不需要加载，从而大大降低了功耗。

4.5 纹理压缩技术

纹理压缩技术的主要目的是降低对存储器带宽的需求。基本的思想是对纹理图像进行压缩，减少纹理数据的总量，以在不大幅降低图像质量的情况下节省存储器带宽。对纹理数据进行压缩有一些要求：由于需要进行直接地址计算，需要有固定的数据压缩率；另外，压缩过程中间接查找的数量也有限制。这样在硬件的实现过程中就能够较为简单和快速，从而保证流水线的延迟较小。

目前主要有两种纹理压缩算法。Palettized 算法[17,18]使用一个颜色表来存储高精度颜色值，每个纹素使用一个索引值来访问表中的颜色值。另一种压缩策略是 iPackman[11]，该策略基于人眼对亮度的敏感性要高于对色度的敏感性的事实，在改变多个纹素色度的同时也改变每个纹素的亮度。Palettized 算法适合于压缩颜色数量较少的图像，而 iPackman 适合于压缩那些纹素亮度差别大而色度相对统一的图像。

4.6 GPU 的片上 Cache

由于图元的顶点属性信息是通过主机接口从外部主存进入 GPU 内部，因此在渲染流水线中构建片上 Cache 有助于减轻 GPU 对外部主存的压力，如图 8 所示。

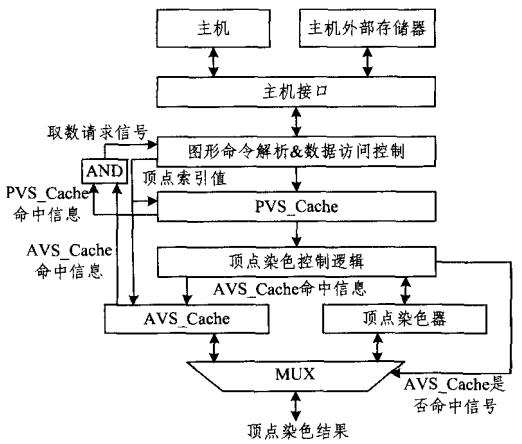

图 8 面向顶点染色的片上 Cache 系统

首先，可以在顶点染色阶段（包括变换和光照计算）前设置 PVS-Cache，该 Cache 主要用来减少对外部主存储器的数据访问。PVS_Cache（PVS-Pre Vertex Shading）对顶点数据的预取可以充分地利用主存具备的突发数据传送效率，这通常比使用多次分离的访问效率更高。研究表明，当 PVS-Cache 有 32 个入口时，81%的顶点信息可以被重用[14]。

图 9 中的顶点存在共用情况，公用的顶点被重复处理。例如，顶点 5 就被处理了 4 次。AVS_Cache（AVS-After Vertex Shading）的作用是减少这类重复操作，从而在提升计算效率的同时降低功耗。AVS_Cache 可以暂时保留顶点处理的结果，并且在对该顶点进行新的操作前，AVS_Cache 中的内容不会改变。只有当 PVS_Cache 和 AVS_Cache 同时发生缺失时才需要从外部主存中取数。研究表明，当图形系统中配置 32 入口的 PVS_Cache 和 8 入口的 AVS_Cache 时，大约 65%的 GPU 与主机间的数据带宽可以被节省，这大大降低了从主机存储器中取数而产生的功耗。

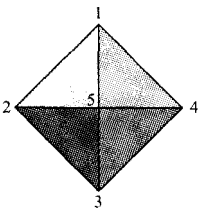

图 9 顶点共用示例图

4.7 低功耗寄存器文件

目前，高性能 GPU 均采用阵列式的流处理内核[27,30]对像素和顶点数据进行染色处理，因此多种用途的大容量寄存器文件必不可少。通常来说，寄存器文件分为输入寄存器文件、输出寄存器文件、临时寄存器文件和常数寄存器文件如用来保存顶点坐标、归一化向量、像素位置、颜色和纹理坐标的寄存器文件属于输入寄存器文件，而用来保存顶点和像素最终染色计算结果的寄存器文件是输出寄存器文件，常数寄存器文件存储 3D 图形操作中用到的系数，临时寄存器文件用来存储在顶点程序和像素程序执行过程中的一些临时计算结果。这些寄存器文件的动态和静态功耗通常占到芯片总功耗的很大比例。常规的设计方法是将这些片上存储器作为一个整体来访问。为了降低功耗，可以考虑使用分区访问技术。

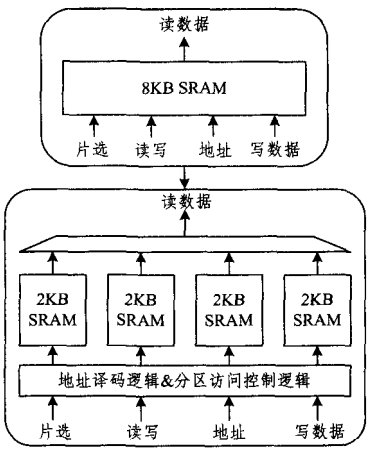

图 10 寄存器文件的分区访问机制

假设访问一个 8192 入口的寄存器文件，需要使用 13bits 地址。每次访问过程中，一个 13bits 输入 8192 输出的译码电路均将工作。如果将上述寄存器文件分为 8 个分区，每个分区 1024 行，则访问每个分区的索引地址将减少到 10bits。此外，需要在前端增加一个 3-8 译码器。每次访问寄存器文件时，高 3bits 地址驱动 3-8 译码器，然后根据译码结果使用低 10bits 地址访问特定寄存器文件分区。这样，每次访问只有一个分区处于活动状态，并且 10bits 地址有助于降低分区译

码器的功耗。同时，因为存储器深度减小，寄存器文件中数据位线所串联存储单元的数目也将减少到原先的 1/8，连线的负载电容大幅减少。因此，寄存器文件本身的功耗也会降低，如图 10 所示。

对于 GPU 来说，寄存器文件通道数通常与 SIMD 数据通路的通道数目相同，根据执行的指令情况，并不是所有的通道都会在同一时刻被激活。选择性的通道激活机制能够有效降低未被使用通道的功耗。通过指令的译码和写掩码，相应的通道可以被激活，没有被选中的通道将被门控从而阻止无谓的功耗浪费。

5 进一步研究

目前，高性能 GPU 已经能够集成上千个流处理内核，这些流处理器阵列占据了 GPU 功耗的绝大部分，而是否所有的应用程序都需要全部的流处理内核全速工作是一个值得研究的问题。根据应用程序的规模和行为特征动态控制需要运行的内核数量并应用 DVFS 策略是一个既能保证程序性能又能降低功耗的有效手段。

复杂 3D 图形通常由大量三角形构成，在硬件渲染过程中会对多个三角形共用的顶点进行多次光照和颜色计算，这不但浪费计算资源、增加延迟，更重要的是增加了不必要的能量消耗。已经有文献研究通过顶点 Cache 的方法来避免对顶点属性的重复计算，但代价是增加片上 Cache，而 Cache 也需要消耗能量并占用芯片面积。可以考虑在图形 API 中或程序编译时为共用顶点加入相关标志位，使该顶点在染色时只属于一个三角形，从而指示硬件染色器达到不进行重复计算的目的。由于不会改变顶点的存在性，因此不会影响到三角形的建立、消隐和光栅化等阶段。这种方法不使用硬件进行处理，不会增加芯片面积，并能够有效提升染色效率，降低 GPU 功耗。

另外，GPU 除了包含大量执行资源外，存储资源是低功耗设计需要关注的另一个重要内容。大量的片上寄存器文件和 Cache 不但动态功耗巨大，漏电流也不可忽视。可以根据应用程序对存储资源的需求变化和行为特征，结合适当的预测器，在保证性能不受影响的情况下，采用 DVFS 和门控策略相结合的方式减少不必要的能量消耗。

结束语 功耗性能是衡量 GPU 经济性、可用性、稳定性和可靠性的重要指标。文章首先分析了 GPU 的发展现状和背景技术，描述了面向 GPU 的功耗设计相关研究工作。接着，在体系结构级结合图形处理器渲染流水线结构，给出了针对深度测试、数据通路、纹理采样和压缩、渲染策略以及片上寄存器文件和 Cache 的低功耗设计技术。最后，指出了 GPU 低功耗设计的进一步研究方向。

参 考 文 献

[1] Woo R. A 210-mW graphics LSI implementing full 3D pipeline with 264-Mtexels/s texturing for mobile multimedia applications [J]. IEEE, J. Solid-St. Circ., 2004, 39(2): 358-367

[2] Yosida K, Sakamoto T, Hase T. A 3D graphics library for 32-bit mocroprocessors for embedded systems[J]. IEEE Trans. Consum. Electron., 1998, 44(4): 1107-1114

[3] Jr M J N. Computer multiplication and division using binary logarithms[J]. IEEE Trans. Electron. Comput., 1962, 11: 512-517

[4] SanGregory S L, Siferd R E, Brother C, et al. A fast, low-power logarithm approximation with CMOS VLSI implementation[C]// Proc. of IEEE Midwest Symposium on Circuits and Systems. 1999: 388-391

[5] Combet M, Zonneveld H, Verbeek L. Computation of the base-two logarithm of binary numbers[J]. IEEE Trans. Electron. Comput., 1965, 14: 863-867

[6] Kim H. A 231-MHz, 2. 18-mW 32-bit logarithmic arithmetic unit for fixed-point 3D graphics system[J]. IEEE J. Solid-St. Circ., 2006, 41(11): 2373-2381

[7] Williams L. Pyramidal parametrics[C]// Proc. of SIGGRAPH. 1983: 1-11

[8] Hakura Z S, Gupta A. The design and analysis of a cache architecture for texture mapping[C]// Proc. of 24th International Symposium on Computer Architecture. 1997: 108-120

[9] Park Y-H, Han S-H, Kim J-S. A 7. 1-GB/s low -power 3D rendering engine in 2D array embedded memory logic CMOS[C]// Digest of Technical Papers of IEEE International Solid-State Circuits Conference. 2000

[10] Knittel G, Schilling A, Kugler A, et al. Hardware for superior texture performance[J]. Comput. Graph., 1996, 20(4): 475-481

[11] Strom J, Akenine-Moller T. iPACKMAN: high-quality, low-complexity texture compression for mobile phones[C]// HWWS'05: Proceedings of ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. 2005: 63-70

[12] Akenine-Moller T, Strom J. Graphics for the masses: a hardware rasterization architecture for mobile phones[C]// Proceedings of the ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'03). ACM, New York, NY, 2003: 801-808

[13] Woo J-H. A 195/152-mW mobile multimedia SoC with fully programmable 3D graphics and MPEG4/H. 264/JPEG[J]. IEEE J. Solid-St. Circ., 2008, 43(9): 2047-2056

[14] ARM Corporation. AMBA 2. 0 Specification[S]. Revision 2. 0

[15] NVIDIA's Next Generation CUDA Compute Architecture: Fermi, v1. 1. White Paper[OL]. http://www. nvidia. com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper. pdf, 2009-09

[16] Wittenbrink C M, Kilgariff E, Prabhu A. Fermi GF100 GPU Architecture[J]. IEEE Micro, March/April, 2011, 31(2): 50-59

[17] 梁宇，韩奇，魏同力. 低功耗数字系统设计方法[J]. 东南大学学报：自然科学版，2000(5)：30

[18] GOODHEAD. Matrix hd 5870 power consumption and thermals [OL]. http://www. bittech. net/hardware/graphics/2010 /07/15/asus-matrix-hd-5870-graphics-card- review/7

[19] Lindholm E, Nickolls J, Oberman S, et al. NVIDIA TESLA: A Unified Graphics and Computing Architecture[J]. IEEE Micro, March/April, 2008, 28(2): 39-55

[20] Sheaffer J W, Luebke D, Skadron K. A flexible simulation framework for graphics architectures[C]// Proceedings of the

ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware(HWWS'04). ACM, New York, NY, 2004: 85-94

[21] Mochocki B, Lahiri K, Cadambi S, et al. Signature-based workload estimation for mobile 3d graphics[C]// Proceedings of the 43rd Annual Conference on Design Automation (DAC' 06). ACM, New York, NY, 2006: 592-597

[22] Lee N B-G, et al. A low-power handheld GPU using logarithmic arithmetic and tripleDVFS power domains[C]// Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics hardware(GH'07). 2007: 73-80

[23] Igehy H, Eldridge M, Proudfoot K. Prefetching in a texture cache architecture[C]// HWWS' 98: Proceedings of the ACM SIGGRAPH/ EUROGRAPHICS Workshop on Graphics Hardware. ACM, New York, NY, 1998: 133-142

[24] Powell M, Yang S-H, Falsafi B, et al. Gated-vdd: A circuit technique to reduce leakage in deep-sub micron cache memories[C]// Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED' 00). ACM, New York, NY, 2000: 90-95

[25] Kaxiras S, Hu Z, Martonosi M. Cache decay: exploiting generational behavior to reduce cache leakage power[C]// Proceedings of the 28th Annual International Symposium on Computer Architecture(ISCA'01). ACM, New York, NY, 2001: 240-251

[26] Flautner K, Kim N S, Martin S, et al. Drowsy caches: Simple techniques for reducing leakage power[C]// Proceedings of the 29th Annual International Symposium on Computer Architecture(ISCA'02). IEEE Computer Society, 2002: 148-157

[27] Youssef A, Anis M, Elmasry M. Dynamic standby prediction for leakage tolerant microprocessor functional units[C]// Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2006: 371-384

[28] 杨毅，郭立，史鸿声，等. 面向移动设备的 3D 图形处理器设计[J]. 小型微型计算机系统，2009，30(8)：1668-1672

[29] 韩俊刚，蒋林，杜慧敏，等. 一种图形加速器和着色器的体系结构[J]. 计算机辅助设计与图形学学报，2010，23(3)：363-372

[30] 韩俊刚，刘有耀，张晓. 图形处理器的历史现状与发展趋势[J]. 西安邮电学院学报，2011，16(3)：61-64

(上接第 195 页)

由结果可见，实验结果受到几个因素的影响。如漫画风格、图片质量。同时，过分和少分相互影响。如果把 3 个分割比值调大些，对分割线的要求将更严格，出现过分的情况会随之减少，但少分情况由于分割线要求的升高会增加，反之亦然。所以根据大多数漫画文法综合考虑，选取过分和少分总和较少的值。

结束语 本文分析漫画后提出一种基于二叉树的漫画帧分割排序法和基于分割比的分割线选取方法对漫画帧进行分割排序。其虽尚存在不足之处，但总体分割成功率较高，有以下几点：

(1)本方法对漫画帧的排序没有基于分割线角度来判断，而是基于分割线与边缘线交点的特点，对漫画帧形状唯一的限制是四边形，与漫画文法相契合。

(2)通过设定分割比，对分割线不完整的情况放松条件，使情景画面和对话框覆盖分割线的情况得到解决。

(3)直线检测是基于细化后的轮廓线上进行的，提高了原本非常耗费时间的直线检测的效率。

(4)对分割线与前景图像相交情况做了判断，从而出现了很少的错分与过分图片。

本文只对一般漫画进行分割，对于嵌套或重叠漫画帧内容的无损分割是本文以后的研究内容。

参 考 文 献

[1] Yamada T, Watanabe T. Extraction of Person Objects from Japanese. Four-Scenes Comics [J]. IEIC Technical Report, 1999, 99(47): 152-159

[2] Hoashi K, Ono C, Ishii D, et al. Automatic Preview Generation of Comic Episodes for Digitized Comic Search [C]// Proceedings of the 19th ACM international conference on Multimedia. 2011: 1489-1492

[3] Kurlander D, Skelly T, Salesin D. Comic Chat [C]// Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. 1996: 225-236

[4] Yamada M, Budiarto R, Miyazaki S. Comic image decomposition for reading comics on cellular phones [J]. IEICE transactions on information and systems, 2004, 87(6): 1370-1376

[5] Tanaka T T, Shoji K, Toyama F, et al. Layout analysis of tree-structured scene frames in comic images [C]// Proceedings of IJCAI. 2007: 2885-2890

[6] Chan C, Leung H, Komura T. Automatic panel extraction of color comic images [J]. Advances in Multimedia Information Processing-PCM 2007, 2007, 4810: 775-784

[7] Arai K, Herman T. Method for automatic e-comic scene frame extraction for reading comic on mobile devices[C]// Information Technology: New Generations (ITNG), 2010 Seventh International Conference on. IEEE, 2010: 370-375

[8] In Y, Oie T, Higuchi M, et al. Fast frame decomposition and sorting by contour tracing for mobile phone comic images [J]. International journal of systems applications, engineering and development. 2011, 5(2): 216-223

[9] Nobuyuki O. A threshold selection method from gray-level histograms [J]. IEEE transactions on systems, 1979, 9(1): 62-66

[10] Fernandes L A F, Oliveira M M. Real-time line detection through an improved Hough transform voting scheme[J]. Pattern Recognition, 2008, 41(1): 299-314

[11] 郭斯，霍文娟，唐求，等. 结合 Hough 变化与改进最小二乘法的直线检测[J]. 计算机科学，2012，39(4)：196-200

[12] Illingworth J, Kittler J. A survey of Hough transform [J]. Computer Vision Graphics Image Processing, 1988, 44(1): 87-116