doi:10.3772/j.issn.1002-0470.2022.02.005

一种面向嵌入式图形处理器的访存子系统结构设计①

赵士彭②*,**,*** 张立志*,**,*** 章隆兵*,**,***

(*计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

(**中国科学院计算技术研究所 北京 100190)

(***中国科学院大学 北京 100049)

摘 要 嵌入式图形处理器(GPU)随着访存数据量越来越大,访存子系统在性能、面积及功耗等方面的瓶颈已经日益凸显。针对图形处理器的数据特点及访存需求,考虑到嵌入式图形处理器面积及功耗的约束,结合 Godson GPU 架构平台,提出了一种面向嵌入式图形处理器的访存子系统结构设计。该设计主要针对图形处理流水线的访存特点,对 cache 的结构进行了优化,并提出了一种基于链表方式的结构,提高了访存的效率,减少了面积且降低了功耗。为了使访存子系统适配并行图形流水线,提出了一种屏幕分区方法,可以在消除 cache 的一致性问题的同时,使访存子系统的负载更加均衡。该设计为嵌入式图形处理器的访存子系统设计提供了借鉴。

关键词 图形处理器(GPU);访存子系统;嵌入式处理器;链表设计

0 引 言

图形处理器(graphic processing unit, GPU)[1]是计算机系统[2]中处理 3D 实时图形程序的专用加速芯片,已经成为计算机系统中不可或缺的一部分。图形处理器作为图形加速器,在游戏、通用计算、图像处理等领域都发挥着不可替代的作用。在现代计算机系统中,GPU 已经变得越来越不可或缺,且变得越来越复杂[3-6]。

伴随着图形处理器性能的增长,数据量出现大幅提高[7-9],访存的瓶颈也日益凸显。目前的图形处理器访存子系统已经严重限制了性能的提高,成为图形处理器急需解决的瓶颈之一。

嵌入式图形处理器由于其面积小、功耗低等特点,受到了很多厂商的追捧。在有限的面积上设计出更低功耗的图形处理器也成为业界的研发方向之一。图形处理器由于其访存密集、数据量巨大,访存子系统在嵌入式图形处理器中急需改进。

综上所述,鉴于访存子系统对 GPU 性能的严重制约,访存性能也由于技术难度无法和计算速度匹配,嵌入式图形处理器由于其面积和功耗等方面的限制,对于访存子系统的制约更为严重,对访存子系统的改进需求也更为迫切。

传统的图形处理器访存子系统通常采用通用的 cache 结构,采用增加二级 cache 或三级 cache 的方式,提升 cache 命中率,提升访存子系统性能。又或者选择增加 cache 容量的方式,减少 cache 的缺失,提高访存子系统性能。这些通用的提升访存子系统性能的方式,均没有针对图形流水线的数据及访存特点,针对性地定制化重新设计 cache 结构而提高访存子系统的性能。

本文针对图形处理器访存的特点,对 cache 的结构进行了优化,并提出了一种基于链表方式的访存子系统结构设计。通过对 cache 结构的优化,有

① 国家自然科学基金(61521092, 61432016)和中国科学院重点部署(ZDRW-XH-2017-1)资助项目。

② 男,1993 年生,博士生;研究方向:计算机体系结构,处理器设计;联系人,E-mail: zhaoshipeng@ict.ac.cn。

(收稿日期:2020-12-25)

效地减少了面积,并降低了功耗。通过链表的设计方法,可以有效地合并具有相关性的访存请求,消除访存之间的相关性,不堵塞图形流水线。同时,链表方式的设计,可以保证访存请求的乱序执行,更进一步提高访存子系统的性能及效率。

本文通过分析图形应用的行为方式,适配并行图形处理流水线的访存需求,提出了一种屏幕分区方法。通过将屏幕进行有效的分区,将访存请求均匀地分散在 cache 的 4 个 bank 上,在消除 cache 一致性问题的同时,避免了访存请求过于集中于某一访存流水线,导致负载不均衡。

本文在第 1 节介绍了目前最具代表性的图形处理器厂商访存子系统结构设计及参数,分析了各大图形处理器厂商对于访存子系统的发展趋势。第 2 节介绍了本文提出的面向嵌入式图形处理器的访存子系统设计。第 3 节介绍了基于本设计的实验结果及分析。第 4 节对本设计进行了总结。

1 背景介绍

在当下处理器计算性能大幅提升的前提下,存储结构渐渐成为了处理器性能提升的主要瓶颈。图形处理器的计算单元无论是数量还是速度上近几年都得到了大幅度的提升,访存的瓶颈也越来越凸显。近几年,Nvidia 和 AMD 两大图形处理器厂商的白皮书都拿出相当一部分的篇幅讲述访存子系统结构的改进以及带宽的优化。这两家图形处理器巨头厂商对于访存子系统的改进方案各有不同。AMD 选择将 cache 层级做得更加深入,增加了多级的统一 cache。Nvidia 则选择增大 cache 容量,提高访存性能。

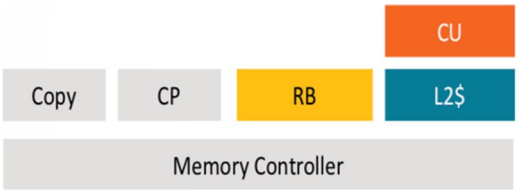
Nvidia 和 AMD 是目前行业领先的图形处理器研发厂商,在行业内具有引领作用。由于商业模式,二者均没有公开其具体的访存子系统结构设计,但通过其发布的白皮书也可分析出当前行业趋势。Nvidia 自 2012 年起共发布了 4 种图形处理器架构的白皮书[10-14],分别是 2012 年的 Kepler 架构、2014 年的 Maxwell 架构、2016 年的 Pascal 架构以及 2018 年的 Turing 架构。AMD 的访存子系统结构主要是 GCN(graphic core next)系列架构以及 2019 年新发布的 RDNA 架构[15-19]。

表 1 是 Nvidia 各架构白皮书中的访存子系统结构各项数据。由表 1 中的数据可以看出,Nvidia 自 Kepler 架构开始,访存子系统的结构层次没有发生很大的变化,但 cache 容量出现大幅增加。图 1 是 AMD 在白皮书中披露的访存子系统结构层次。从图 1 中可以看出,AMD 主要是改变了访存子系统的结构层次,cache 的级数越做越深。图 2 是 AMD 的 Vega 架构与 RDNA 架构的 cache 容量的比较。从图 2 可以看出,Icache 和 Kcache 的总容量及行容量均有了大幅提高。RDNA 架构中的 L0 cache 相当于 Vega 架构的 L1 cache。新增加的 L1 cache 与 ROP(raster operation processors)单元共享。L2 cache 将 copy engine 即 DMA(direct memory access)进行了集成共享。

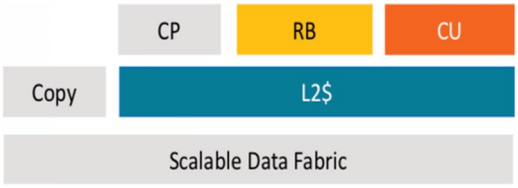
通过比较 AMD 与 Nvidia 两家行业领先厂商的访存子系统结构,可以看出 AMD 更倾向于将访存子系统的结构层次不断做深,并逐渐采用统一的 cache 结构。AMD 的 GCN4 访存子系统结构没有 Nvidia 的统一 L2 cache,AMD 的 RDNA 的访存子系统结构比 Nvidia 做的更深更统一。Nvidia 的访存子系统结构层级始终选择统一的 L2cache,SP(stream processor)拥有独立的 L1 cache,这和 AMD 的 Vega 架构存储结构层次是一致的。

表 1 Nvidia 4 种架构访存子系统及 ROP 数据

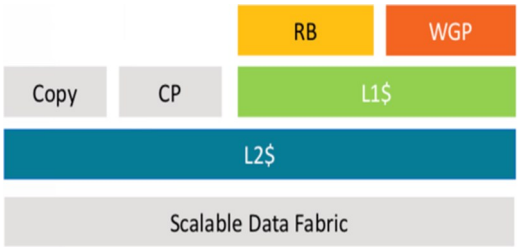
架构	发布时间	L2 cache	L1 cache	GPC	ROP 部分	ROP 单元
Kepler	2012	512 kB	64 kB/SM	4	4	32(8×4)
Maxwell	2014	2 MB	96 kB/2SM	4	4	64(16×4)
Pascal	2016	4 MB	64 kB/SM	6	-	-
Turing	2018	3 MB×2	(32 kB+64 kB)/SM	6	12	96(8×12)



（a）GCN4 访存子系统结构



（b）Vega 访存子系统结构



（c）RDNA 访存子系统结构

图 1　AMD 3 种访存子系统层次图

RX Vega 64	Size	Cache Line Size	Read/Write
Instruction Cache (I$)	32KB per 4 CUs	32B	Read-only
Scalar Cache (K$)	16KB per 4 CUs	32B	Read-only
L1 Cache	16KB per CU	64B	Read-only
L2 Cache	4MB	64B	Read/Write

（a）Vega 访存子系统 cache 容量

RX 5700 XT	Size	Cache Line Size	Read/Write
Instruction Cache (I$)	32KB per WGP (~2CUs)	64B	Read-only
Scalar Cache (K$)	16KB per WGP (~2CUs)	64B	Read-only
L0 Cache	2x 16KB per WGP (~2CUs)	128B	Read-only
L1 Cache	128KB per shader array	128B	Read-only
L2 Cache	4MB	128B	Read/Write

（b）RDNA 访存子系统 cache 容量

图 2　AMD 最新访存子系统 cache 容量

在 AMD 在 GCN4 及其之前的架构访存子系统层次中，SP 单元拥有独立的存储结构层级，拥有独立的 L1 和 L2 cache，不与其他单元共享。ROP 单元通过独立的 depth/stencil cache 和 color cache 直接访问显存。命令处理器的指令等信息也直接访问显存。Vega 架构则将 L2 cache 做成统一的 cache 结构，ROP 单元与命令处理器均采用该统一 cache。AMD 在最新的 RDNA 架构中则又进一步做深访存子系统层次。SP 内部采用 L0 cache、Icache 和 Kcache，并与 ROP 单元采用统一的 L1 cache。为了适配 SP 的只读访存特性，与 ROP 统一的 L1 cache 选择做成只读 cache，ROP 单元的写操作直接操作 L2 cache。随着 AMD 的访存子系统结构越做越深，Vega 架构采用统一的 L2 cache，并使 L2 cache 容量增加至4 MB。RDNA 架构的 L0 cache 相当于 Vega 架构的 L1 cache。新增加的统一 L1 cache 容量大小为128 kB。越来越多的统一 cache 出现，所有的 cache 行容量均出现不同幅度的增加。另一 GPU 厂商 Nvidia 的访存子系统层次则从 2012 年发布的 Kepler 起没有再进一步做深，始终采用了与 AMD 的 Vega 系列相同的访存子系统结构层次。但 Nvidia 的 L2 cache 容量却不断增加。L1 cache 从 Maxwell 架构开始便采用 1 个 SM 内部共享 1 个 L1 cache。Turing 架构则为了通用计算更进一步将 96 kB 的 L1 cache 拆分成 64 kB 的 share cache 加上 32 kB 的 L1 cache 或 32 kB 的 share cache 加上 64 kB 的 L1 cache 2 种结构。

综上所述，AMD 与 Nvidia 提高访存子系统性能的方法均是采用提高 cache 容量以及增加 cache 的层级等通用的方式。这种方式的好处是方法更加通用，也可以对访存子系统的性能起到提升的作用。但它未能针对图形处理流水线的特点进行特定的优化，也未能对面积、功耗等方面进行权衡。

本文采用针对图形处理器访存子系统的访存特点，提出一种基于链表方式的访存子系统结构设计和负载均衡设计。在提升访存子系统性能的同时，该设计具有更小的面积，更低的功耗，更加符合嵌入式图形处理器的访存子系统要求。

2　访存子系统结构设计

2.1　基于链表方式的 cache 结构设计

图形处理流水线的访存请求具有一定的相关性。在图元处理过程中，具有相关性的访存请求必须要按序处理，并需要避免写后读（read after write，RAW）相关。这样，访存流水线将会堵塞等待，最终会导致图形流水线的堵塞。

当进行 cache 查找后，cache 缺失的请求会进入

访存的 miss 队列,并发出访存总线请求。当前后 2 个访存请求均需要读取相同的像素块时,2 个访存请求就具有了相关性。为保证功能的正确性,后一个访存请求必须要等待前一个访存请求处理完之后才可以进行处理。这会堵塞图形处理流水线,导致性能损失。

嵌入式图形处理器对于面积和功耗的要求也更加严格,追求面积更小、功耗更低。通常的图形处理器访存子系统设计,采用了通用的 cache 作为片上缓存。分析图形处理流水线可知,为了优化效率,不堵塞图形流水线,访存子系统需要在同一拍同时处理读和写请求,这样才能达到完全流水。通常的访存子系统设计需要在命中情况下达到全流水。传统的图形处理器访存子系统,采用通用 cache 作为片上缓存,当发生读或写请求时,均需要查询 TagRAM,读回填和写请求都需要写 DataRAM。这样为了满足图形流水线需求,则需要两读两写的四端口 RAM 作为 Tag 和 Data 的片上存储,这样做将会浪费大量的面积及功耗。

本文提出了一种基于链表方式的 cache 结构设计。本设计不仅取消了访存的相关性,同时支持乱序访存,提高了访存效率的同时并减少了面积,降低了功耗。

本文提出的链表方式的设计,结合访存 miss 的队列,可以有效消除图形处理流水线中访存请求之间的相关性。图 3 是本文提出的链表方式设计,在访存 miss 队列中加入像素块的首地址以及指向下一个同地址请求的队列项指针。当队列中没有相关的访存请求时,则以正常按序发出访存总线请求。

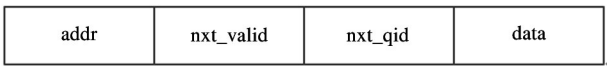


图 3　访存子系统中的链表设计

当队列中出现相关性的访存请求时,则将指针指向当前的队列项,更新为当前的队列项号。每一次查找相关性请求均可通过链表的有效队列进行并行查找,提高相关性请求的查找效率。为避免具有相关性的访存请求多次发出总线请求,导致读出显存中的旧值引起访存功能性错误,链表方式设计在查找出相关访存请求后,后续的相关访存请求不会再进行总线请求。

本文提出的链表方式设计,不仅可以消除访存中的相关性,更可以进行乱序访存,进一步提高访存子系统效率。图形处理流水线中,虽然要求每一级流水级都需要严格按序处理,以防乱序导致图形在深度测试及颜色混合时出现错误,但严格按序只需保证具有相关性的访存请求进行按序处理;其余不具有相关性的访存可以进行乱序处理,以提高访存效率。本文提出的链表方式设计,可以通过有效队列指针,保证具有相关性的访存请求顺序。同时,对不具相关性的访存请求进行乱序访存发射,更进一步优化了访存总线效率,提高访存子系统的访存效率。

通过链表方式设计,还可以支持无效写合并(store fill buffer)优化。当数据具有相关性时,根据图形流水线的特点,后续数据的写会对前一数据的写进行覆盖,导致前级流水的 cache 回填(Refill)操作不需要关心、不需要使用。通过链表方式设计,可以等所有链表的指针写操作均完成时,再进行合并的写操作。

图 4 是本文提出的基于链表方式的片上缓存 cache 的结构图。根据图形处理流水线的访存特点,针对性地设计了一种片上缓存 cache 的结构。当图形流水线发出读请求时,会进行读 cache 的 TagRAM

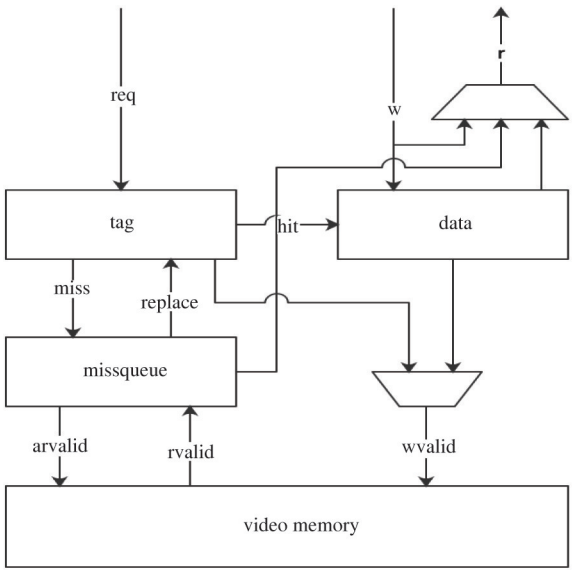


图 4　片上缓存 cache 的结构设计框图

进行查询,若命中,则读 DataRAM。若未命中,当数据返回时,无需更新 TagRAM,数据无需回填 cache 的 DataRAM,直接送回图形流水线,更新在链表之中。在 cache 缺失进行回填 TagRAM 时,需要先堵塞读 Tag 和 Data 流水级将数据送入执行级进行处理,待处理后将结果写回 DataRAM,并同时更新 TagRAM。链表中,如果访存请求存在相关性,则写操作时直接前递给下一流水级,待链表中所有相关性请求执行后再进行写请求更新 cache。由于本设计是基于 OpenGL 2.0 的图形处理流水线,深度和模版测试以及颜色混合无需浮点操作,无需考虑计算延迟,所以可以满足流水性能。

2.2 负载均衡设计

为了提高图形流水线的处理效率,同时支持 4 倍多重采样抗锯齿(multi sampling anti-aliasing, MSAA)操作,图形流水线通常采用 4 条流水线并行处理。为了保证访存子系统不会成为处理瓶颈,匹配图形流水线,访存子系统流水线也必然需要采用 4 条并行处理。为了不引入一致性问题,同时平衡访存子系统的负载,本文提出了一种访存子系统的负载均衡设计。

在进行访存子系统流水线时,本文提出了一种对屏幕进行分区处理的方法。图形流水线是根据图元进行光栅化后产生的像素点为基本单位进行处理,访存子系统通常采用 4 ×4 个像素点组成 tile 或 2 ×2 个像素点组成的 quad 为基本单位进行处理。由于访存的像素点都是通过图元光栅化得到的,所以访存请求具有一定的空间局部性。当这些访存请求被发射到 4 条并行的访存流水线时,需要保证 cache 的一致性,即如果显存数据同时在多条访存流水线中拥有更新数据,最终会导致深度测试和颜色混合出现错误,同时也会引起相关性等致命错误。本文提出的负载均衡设计是将屏幕按照 4 ×4 的 tile 或 2 ×2 的 quad(MSAA 的情况下)为基本单位,均匀分散在 4 条访存流水线中,保证每一个 tile 或 quad 四周的访存数据不会落在同一访存流水线中,充分地避免了 cache 一致性和相关性带来的错误。图 5 是本设计采用的屏幕分区方式,每一个 tile 或 quad 与四周的访存数据均不在同一访存分区,四周访存数据不会落在同一访存流水线,对图形流水线处理的图元来说,分布更加均匀,不会出现某一访存流水线堵塞而其他流水线空闲的情况,负载更加均衡。

0	1	2	3	0	1	2	3
3	2	1	0	3	2	1	0
0	1	2	3	0	1	2	3
3	2	1	0	3	2	1	0

图 5 负载均衡的屏幕分区设计

3 实验结果及分析

为了评估本设计的访存子系统的性能、面积及功耗,本文采用现有的高性能图形处理器,结合高性能图形处理器的驱动设计,采用了 Linux 图形测试集,进行了本访存子系统结构设计的测试。

3.1 图形测试集

本文实验测试采用的基准测试集是 GL _ MARK[20]。GL _ MARK 是一款图形基准测试集,使用 OpenGL-ES 进行开发,提供了一系列丰富的图形测试。涉及图形单元性能的各个方面,涵盖光照、阴影、超多图元、简单 2D 等多种类型的测试,是目前 Linux 操作系统上较为全面的测试集之一。

本文节选了 GL _ MARK 测试集中的几个典型例子,分别是 2D 渲染测试 Effect2D,反射及阴影渲染测试 Shadow,折射及超多图元渲染测试 Refract 以及复杂 shader 处理测试 Jellyfish。通过搭载本访存子系统的 GSGPU(Godson graphic processing unit)图形处理器[21-22]得到最终渲染出的结果。

3.2 GSGPU 图形处理器平台

本文使用的 GSGPU 图形处理器主要由命令处理器(command processor, CP)、全局任务调度器(global task scheduler, GTS)、图形处理集群(graphics processing cluster, GPC)、二级静态缓存(L2 SCache)、内存控制器(memory controller, MC)5 部分组成。其中图形处理集群又由计算处理引擎(compute engine, CE)、几何处理引擎(geometry engine, GE)、图元处理引擎(primitive engine, PE)、局部任

务调度器(local task scheduler,LTS)、流处理器集群(stream processor cluster,SPC)、输出合并单元(output merge unit,OMU)6 部分组成。整体结构如图 6 所示。

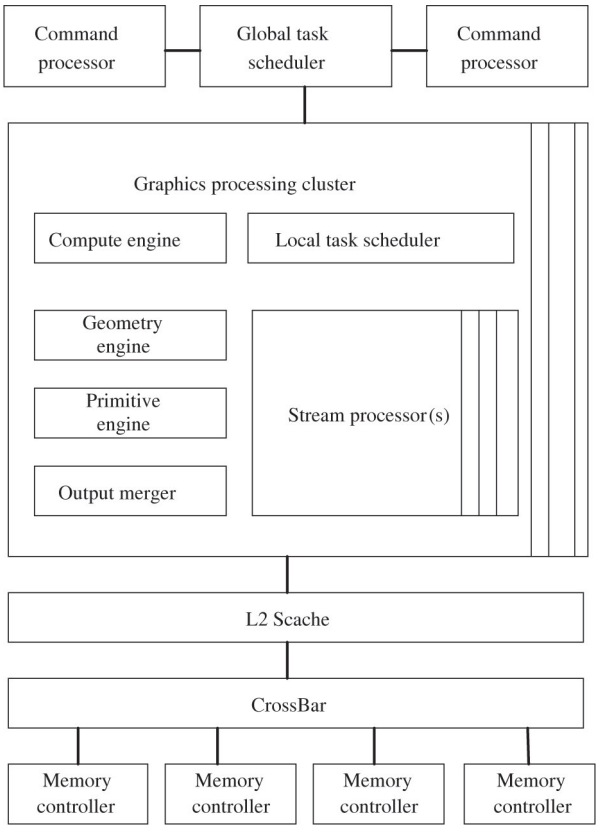


图 6　GSGPU 图形处理器的结构框图

3.3　访存子系统性能分析

搭载本设计的 GSGPU 图形处理器光栅化后的访存基本单位是 4 ×4 的像素块,所以在设计 cache 行时选择了 4 ×4 的像素块作为 cache 行宽度。经过性能与面积功耗权衡后,选取 4 路组相连 cache,容量大小选择 32 kB。

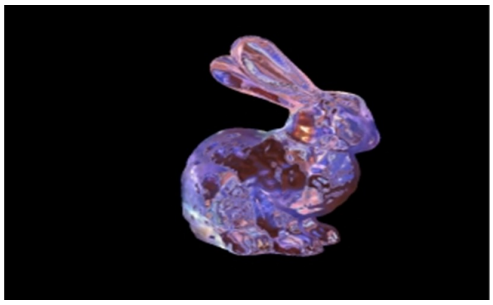
表 2 是搭载本设计的 GSGPU 图形处理器访存子系统的访存请求及 cache 命中率。本文选取了 GL _ MARK 基准测试集中最典型的 4 个测试用例,分别是 2D 渲染测试的 Effect 2D、阴影及反射测试的 Shadow、折射及超多图元测试 Refract 以及复杂 shader 程序的 Jellyfish 测试。图 7 是搭载本设计访存子系统的 GSGPU 图形处理器渲染效果图。

表 2　访存子系统访存请求及命中率

Test example	Vertices number	Memory request	Cache hit
Effect 2D	4	8070	80.86%
Shadow	21 516	120 402	56.67%
Refract	209 889	647 680	11.91%
Jellyfish	13 200	316 088	35.00%

(a) Effect 2D 效果图

(b) Shadow 效果图



(c) Refract 效果图

(d) Jellyfish 效果图

图 7　搭载本设计的 GSGPU 图形处理器渲染效果图

从图 7 可以看出,本文提出的访存子系统结构设计在功能上完全正确,访存结果无误。在针对 2D 渲染场景时,顶点数量虽然只有 4 个,但访存请求是根据像素点为基本单位进行计算。经过光栅化成像素点后,访存请求的数量也达到了 8070 次。通过 GL _ MARK 测试集的测试用例可以看出,即使再简单的测试用例,渲染一帧的图像,图形处理器的访存请求数量依然很多,访存子系统的压力依然巨大。所以,针对访存子系统的优化对于图形处理器的性能提升具有很大的帮助。

通过表 2 可以看出,搭载本设计的访存子系统在少至 8070 个的访存请求、多至 647 680 个的大量访存请求下,均可以正常处理。在如此巨大的访存请求下,链式逻辑设计可以完全消除访存相关性,并乱序访存提高性能,这将对访存子系统和图形处理器整体性能提升起到巨大的作用。由于图形处理器的访存请求通常不具备良好的时间局部性,所以在普通的图形处理程序中,cache 的命中率有限,后续也可通过预取等方式提升命中率。

3.4 访存子系统功耗及面积分析

本文采用 Synopsys Design Compiler 对提出的访存子系统的 RTL 代码进行综合,工艺库选择 C28SOI _ SC _ 12 _ CORHP _ LL。表 3 和表 4 分别是功耗综合报告以及面积综合报告。从综合报告可以看出,本访存子系统总功耗仅需 1.45 W。当前,新发布的商用高性能嵌入式图形处理器总功耗约为 40 W 左右[23]。与之相比,在满足性能需求的情况下,本设计完全满足嵌入式图形处理器对于功耗方面的要求。本访存子系统面积也满足当前嵌入式图形处理器的设计指标。

表 3 Synopsys Design Compiler 功耗综合结果

功率/mW	Internal	Switching	Leakage	Total
Register	1.1131e +03	0.2925	144.6265	1.2580e +03
Combinational	6.3392	5.6609	185.29408	197.2938
Total	1.1194e +03	7.9533	329.9205	1.4553e +03

表 4 Synopsys Design Compiler 面积综合结果

Combinational area	Buf/Inv area	Noncombinational area	Total
810 031.459 124 8	103 569.073 056 8	708 985.182 212	1 519 016.641 336 8

4 结 论

本文通过分析图形处理流水线访存请求的特点,提出了面向嵌入式图形处理器的访存子系统结构设计。本设计针对图形处理流水线的访存特点,对 cache 的结构进行了优化设计,大幅减少了访存子系统的面积并降低了功耗,更适合嵌入式图形处理器设计。本设计提出的基于链表方式的访存子系统结构设计,有效地消除了图形流水线的访存请求的相关性,当相关性访存请求时不再堵塞流水线,还可以通过乱序访存提升访存性能和效率。本文提出了一种屏幕分区方式,适配并行图形处理流水线,消除 cache 的一致性问题,使访存负载更均衡。实验通过搭载在 GSGPU 图形处理器,跑通 GL _ MARK 图形基准测试集,完成了正确性验证。并通过 Design Compiler 进行了面积及功耗方面的评估,达到了嵌入式图形处理器的设计指标。本设计为提高嵌入式图形处理器访存子系统的性能设计提出了一个新的方向,也为嵌入式图形处理器的访存子系统设计提供了重要的参考。

参考文献

[1] McClanahan C. History and Evolution of GPU Architecture[R]. Atlanta: College of Computing, Georgia Institute of Technology, 2010

[2] AKENINE-MOLLER T, HAINES E, HOFFMAN N. Real-Time Rendering[M]. Natick/Boca Raton: A K Peters/CRC Press, 2019

[3] AMD Corporation. RDNA architecture presentation[EB/OL]. https://gpuopen. com/wpcontent/uploads/2019/08/RD NA _ Architecture _ public. pdf: AMD,(2019-05-23),[2020-12-01]

[4] AMD Corporation. RDNA 1.0 instruction set architecture [EB/OL]. https://gpuopen. com/wp-content/uploads/2019/08/RD NA _ Shader _ ISA _ 5August2019. pdf: AMD,(2019-07-07),[2020-12-01]

[5] AMD Corporation. RDNA Architecture Whitepaper[EB/OL]. https://www. amd. com/system/files/documents/rdna _ whitepaper. pdf: AMD, (2019-05-23),[2020-12-01]

[6] LINDHOLM E. NVIDIA Tesla: a unified graphics and computing architecture[C]// IEEE Micro, LakeComo, Italy, 2008:39-55

[7] HESTNESS J, KECKLER S W, WOOD D A. A comparative analysis of microarchitecture effects on CPU and GPU memory system behavior[C]// IEEE International Symposium on Workload Characterization, Raleigh, USA, 2014: 150-160

[8] KIM G, LEE M, JEONG J Y, et al. Multi-GPU system design with memory networks[C]// IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK, 2014:484 - 495

[9] 卢俊,颜哲,田泽. 一种高效 GPU 存储系统体系架构设计[J]. 计算机技术与发展,2015,25(4):6-9

[10] WITTENBRINK C M, KILGARIFF E, PRABHU A S. Fermi GF100 GPU architecture[C]// IEEE Micro, Porto Alegre, Brazil, 2011:50-59

[11] NVIDIA Corporation. NVIDIA PascalWhitepaper[EB/OL]. https://www. nvidia. com/en-us/data-center/pascal-gpu-architect ure/,(2016-01-01),[2020-12-01]

[12] NVIDIA Corporation. NVIDIA Turing Whitepaper[EB/OL]. https://download. csdn. net/download/weixin _ 4087868 4/10682852, (2018-07-07),[2020-12-01]

[13] NVIDIA Corporation. NVIDIA Maxwell Whitepaper[EB/OL]. http://www. docin. com/p-1099423122. html, (2014-01-04),[2020-12-01]

[14] NVIDIA Corporation. NVIDIA Tegra K1 Whitepaper [EB/OL]. https://www. mendeley. com/catalogue/45d78968-f62 1-3535-85cd-a21c3fcc6b7d/, (2016-05-12),[2020-12-01]

[15] AMD Corporation. AMD RS880 Databook device specification for the RS880[EB/OL], http://support. amd. com/te chdocs/46112. pdf,[2020-12-01]

[16] AMD Corporation. RADEONPolaris Whitepaper[EB/OL]. https://www. amd. com/system/files/documents/polaris-whitepap er. pdf,[2020-12-01]

[17] AMD Corporation. AMD ISA Documentation[EB/OL]. https://gpuopen. com/documentation/amd-isa-documentation/,[2020-12-01]

[18] AMD Corporation. AMD VEGA ISA Documentation[EB/OL]. https://developer. amd. com/wpcontent/resources/Vega _ 7nm _ Shader _ ISA. pdf,[2020-12-01]

[19] AMD Corporation. AMD GCN3 ISA Documentation[EB/OL]. http://developer. amd. com/wordpress/media/2013/12/AMD _ GCN3 _ Instruction _ Set _ Architecture _ rev1. 1. pdf,[2020-12-01]

[20] glmark2 [EB/OL], https://github. com/glmark2/glmark2,[2020-12-01]

[21] 赵士彭, 张立志, 赵皓宇, 等. 高性能 GPU 模拟器驱动设计研究[J]. 高技术通讯, 2020,30(5): 435-442

[22] 张立志, 赵士彭, 赵皓宇, 等. 高性能 GPU 模拟器的实现[J]. 高技术通讯, 2020,30(6): 553-560

[23] AMD Corporation. AMD 发布全新嵌入式 GPU[EB/OL]. https://www. amd. com/zh-hant/node/9396, (2017-10-09),[2020-12-01]

An architecture design of memory access subsystem for embedded graphics processor

ZHAO Shipeng[*,**,***], ZHANG Lizhi[*,**,***], ZHANG Longbing[*,**,***]

([*] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([***] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

With the increasing amount of memory access for embedded graphics processoring units (GPU), the bottleneck of the memory access subsystem in terms of performance, area and power consumption has become increasingly prominent. In view of the data characteristics and memory access requirements of the graphics processor, considering the constraints of the area and power consumption of the embedded graphics processors, combined with Godson graphic processing unit (GSGPU) architecture platform, a subsystem architecture design of memory access oriented to the embedded graphics processor is proposed. The design mainly aims at the memory access characteristics of the graphics processing pipeline, optimizes the architecture of the cache, and proposes a architecture based on the chainform method, which improves the efficiency of memory access, reduces area and power consumption. In order to adapt the memory fetching subsystem to the parallel graphics pipeline, a screen partition method is proposed, which can eliminate the cache consistency problem while making the load of the memory fetching subsystem more balanced. This design provides a reference for the design of the memory access subsystem of the embedded graphics processor.

Key words: graphic processoing unit (GPU), memory access subsystem, embedded processor, chain form