

龙芯 UNCACHE 加速原理及其在系统图形性能优化中的应用^①

张爽爽^② * * * * * 孟小甫 * * * 汪文祥 * * * 高 翔 * * * * *

(* 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)

(**** 龙芯中科技术有限公司 北京 100095)

摘 要 针对当前采用独立显卡的桌面计算机系统架构普遍存在的 CPU(中心处理单元)访问 GPU(图形处理单元)存储空间数据传输延迟较大的瓶颈,采用了龙芯 GS464 处理器核心实现的 UA(uncache acceleration)机制对 GPU 驱动程序中的 GPU 存储空间访问接口进行优化,极大地提高了处理器向 GPU 等 IO 存储空间连续数据写入的速度。详细分析了龙芯处理器 UA 机制的原理及其相对于 uncache 方式 IO 写所能带来的性能提升。通过 UA 机制优化了龙芯 3A+2H 平台的 GPU 驱动性能, x11perf 测试结果显示,采用 UA 优化 GPU 驱动后, Xserver 的一些接口性能提升达 5%~230%。将龙芯处理器的 UA 机制封装到了标准 MMAP 系统调用,并通过该扩展后的系统调用优化了 Xserver 的 Xvideo 扩展接口,实验结果显示,播放常见较高分辨率视频时该接口性能能够有 6~12 倍的性能提升。

关键词 处理器核, UA, 图形, 图像

0 引 言

现代主流桌面计算机普遍采用了“CPU(中心处理单元)+主存”与“GPU(图形处理单元)+GPU 存储”通过总线互联的系统架构^[1,2],现在正在兴起的异构计算也大都基于这种结构。在这种结构下, CPU 和 GPU 各自地址空间之间会发生大量的数据拷贝操作,主流的 GPU 都实现了直接内存访问(direct memory access, DMA)硬件单元来加速 CPU 和 GPU 之间的数据拷贝。从很大程度上优化了 CPU 与 GPU 之间的数据拷贝,但现代操作系统架构下, 应用程序直接访问 GPU 存储空间的操作是无法直接通过 DMA 进行加速的,一方面这是由于现代操作系统不允许应用程序直接操作硬件,另一方面,即

使将硬件 DMA 抽象为一个应用程序空间的接口, 它所接收到的拷贝操作的源地址和目的地址也都是应用程序空间的虚拟地址,需要查找页表进行虚实地址转换,这些额外的开销可能导致最终拷贝效率下降。

为了解决处理器向 IO 地址空间写入速度慢的问题, MIPS R10000^[3,4] 处理器最早引入了一种叫做 UA(uncache acceleration)的机制来加速处理器向 uncache 的 IO 地址空间的连续地址写入, 龙芯 GS464 处理器核的设计中也实现了一个类似的 UA 机制, 龙芯 3A 处理器^[5] 采用的处理器核心就是 GS464。本文详细分析了龙芯 GS464 处理器核 UA 机制的实现及其理论加速比,并在龙芯 3A+2H^[6] 平台验证了理论分析结果。分析和实验结果显示在 IO 总线性能足够的情况下, UA 方式的写入速度相

① 国家“核高基”科技重大专项课题(2009ZX01028-002-003, 2009ZX01029-001-003, 2012ZX01029-001-002-002, 2014ZX01020201)和国家自然科学基金(61221062, 61232009, 61222204, 61432016)资助项目。
② 男,1988 年生,博士生;研究方向:计算机体系结构;联系人, E-mail: zhangshuangshuang@ict.ac.cn
(收稿日期:2014-11-13)

对于 uncache 方式有很高的加速比,将 UA 应用于龙芯 3A + 2H 平台的图形驱动和图像接口优化也取得了显著的性能提升。

本文对如下内容进行了描述和分析:龙芯 GS464 处理器核心 UA 硬件的实现机制;UA 的加速机制,以及 UA 和 uncache 两种机制下 GPU 存储写入速度差异的量化方式对比;对龙芯平台 Linux 核的存储映射(memory map, MMAP)系统的调用及 UA 功能扩展;实验用龙芯 3A + 2H 互联平台;采用 UA 扩展的 MMAP 系统调用后龙芯 3A + 2H 平台 GPU 存储空间写入与 uncache 方式的性能差异;3A + 2H 平台 GPU 驱动进行 UA 改造前后的图形应用程序编程接口(API)性能提升;采用 UA 扩展的

MMAP 接口对 XServer 的 Xvideo 接口进行优化后的性能改进。

1 龙芯 GS464 处理器核 UA 机制

龙芯 GS464 处理器核心实现了类似于 MIPS R10000 处理器的 UA 机制,该机制的逻辑结构如图 1 所示。GS464 处理器核实现了两个与 MIPS R10000 相同的 32 字节的 UA 缓冲器入口(uncache accelerate buffer entry, UABE),与 MIPS R10000 处理器 4 项 FIFO 等待队列不同,GS464 处理器核实现了 8 项 FIFO 等待队列。

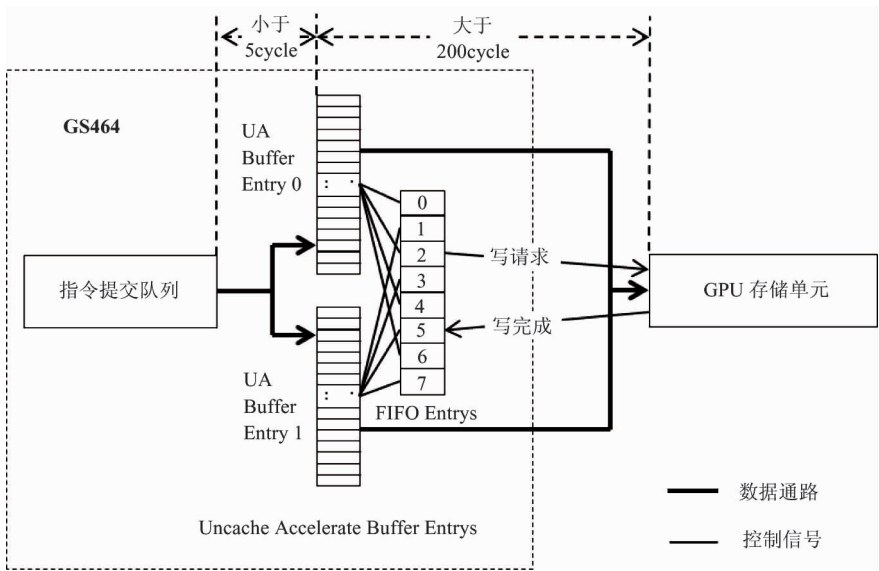


图 1 龙芯 GS464 处理器核心 UA 逻辑结构

MIPS 兼容的龙芯处理器平台中地址访问分为无映射(unmapped)和有映射(mapped)两种,这两种地址的区别在于前者不需要经过转换后备缓冲器(translation lookaside buffer, TLB)^[7,8]的转换,后者则需经过 TLB 转换,其中 unmapped 地址空间访问的高速缓冲存储器(cache)属性位是 64 位地址的[61,63]三位,mapped 地址空间访问的 cache 属性位是该地址所对应数据转换后备缓冲器(data translation lookaside buffer, DTLB)表项中的 cache 属性位,如果一个 unmapped 地址的 cache 属性位或一个 mapped 地址对应 DTLB 页表项的 cache 属性位的值

为 0b111,则该地址的访问方式为 UA 方式。
当处理器核发出一个 UA 的写请求时,该请求将命中 GS464 处理器核心中的非高速缓冲写访问队列(uncache queue, ucqueue)模块,也即命中图 1 中的一个 UABE。

在下面两种情况下,ucqueue 中的写逻辑会将缓冲区中的数据写出处理器核心:

(1) 一个 UA 写请求命中时,如果此次写操作完成后,ucqueue 中的一个 UABE 的 32 项数据缓冲区被填满,则 UA 机制会启动 UABE 的写回逻辑,将当前 UABE 中的数据写出。

(2) 处理器发出一个 UA 的读请求,则该请求会被推迟到 ucqueue 中两项 UABE 都被写回后执行,其执行结果与普通的 uncache 读请求一样,UA 的读请求返回的数据不会被缓存到 ucqueue 中的 UABE 中,也就是说一个 UA 读请求相当于一个 UABE 强制写回操作加一个 uncache 读操作,该机制可以用于 UA 的同步操作。

2 UA 机制的加速原理

在普通的 uncache 访问模式下,如果处理器对一段连续的 GPU 存储空间进行写访问,则每条位于处理器核指令提交队列中的存储(store)指令都需要等到写操作完成的提交(commit)信号才能提交,uncache 写入较长的延迟就导致了 store 指令在处理器核指令提交队列中的阻塞,对于 GS464 处理器核来说,一条 uncache store 指令后面的 uncache store 指令必须等到前面 uncache store 指令提交后才能执行,也就是说连续的 uncache store 指令操作开销等于一条 uncache store 的开销乘以 uncache store 操作指令数,很明显连续的 uncache 写入开销会很大。如果采用 UA 方式的写,则每条 uncache store 指令只需要将数据存入 UABE 中就可以完成当前 store

操作,继续下一条 uncache store 指令的执行了,直到当前正在写入的 UABE 存满数据时才由硬件将数据以数据块的形式写出处理器核,此时 ucqueue 的另外一个 UABE 可以继续接收来自指令提交队列写出的数据,这样两个 UABE 可以交替向处理器核外写出数据。另外由于 UABE 写出数据也是需要等待提交信号的,因此 ucqueue 中实现了一个 8 项的 FIFO Entry,其中 FIFO Entry 0,2,4,6 和 1,3,5,7 分别用于循环等待 UABE 0 和 UABE 1 成功写入 GPU 存储空间的 commit 信号。

如图 2 所示,如果不考虑 commit 信号的传输时间及数据传输总线的带宽限制的话,数据总线上最多可以同时存在 8 块 UABE 的数据写出(W0 ~ W7),第 9 个 UABE 数据块 W8 的发出需要等待 W0 写操作完成的 commit 信号,也就是说阻塞在 UABE 的 W8 数据写出需要等待的是在其之前的第 8 个数据块 W0 的 commit 信号。由上面的分析可以看出,不考虑访存的输出速度以及外部带宽的情况下,能够同时在总线上传输的 UABE 数目由 FIFO Entry 的数目决定,也就是在使用有 4 项 FIFO Entry 的 MIPS R10000 处理器的 UA 机制时,数据总线上最多能同时存在 4 块 UABE 的数据写出。

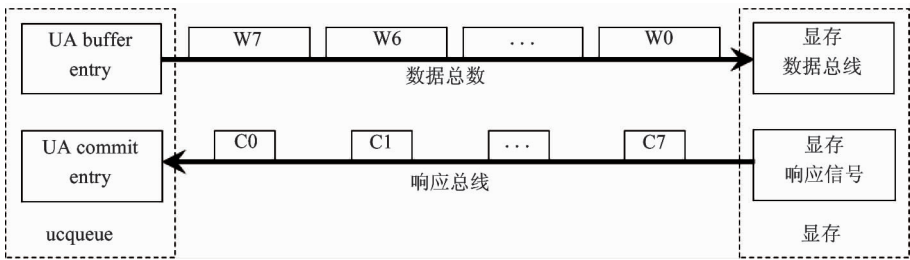


图 2 理想情况下的 UA 数据传输

假设一条 store 指令命中 UABE 的开销为 1 个处理器时钟周期(CPU cycle),而写入 GPU 存储的开销为 300 个时钟周期,由于处理器对 GPU 存储空间的写访问开销主要花费在 commit 信号的等待上,为简化计算,这里假设一项 UABE 写入 GPU 存储的开销也为 300 个时钟周期,设 $T_u(x, size)$, $T_{ua_R10000}(x, size)$ 和 $T_{ua_LS3A}(x, size)$ 分别表示在 uncache,有 4 项 FIFO Entry 的 MIPS R10000 的 UA 机

制,以及有 8 项 FIFO Entry 的龙芯 3A 处理器的 UA 机制下,使用每次写入 x 字节的 store 指令连续写入 $size$ 大小的空间的时间开销。则在不考虑带宽限制的理想情况下,我们能得到如下结论:

$$T_u(x, size) = \frac{size}{x} \times 300$$
$$T_{ua_R10000}(x, size) = \frac{size}{x} \times 1 + \frac{size}{32 \times 4} \times 300$$

$$T_{ua_LS3A}(x,size) = \frac{size}{x} \times 1 + \frac{size}{32 \times 8} \times 300$$

uncache 模式与 UA 模式分别使用 MIPS 的单字节存储指令(SB)、双字节存储指令(SH)、4 字节存储指令(SW)、8 字节存储指令(SD)指令连续写入 1M 字节的情况下,开销见表 1(单位为 CPU 时钟周期数)。

表 1 不同 store 指令 uncache 和 UA 写开销

	uncache (Cycle)	UA_R10000 (Cycle)	UA_LS3A (Cycle)
SB	300M	3.34M	2.17M
SH	150M	2.84M	1.67M
SW	75M	2.59M	1.42M
SD	37.5M	2.47M	1.30M

根据表 1 的理论分析结果可以得出两方面的结论,一方面是采用 SB、SH、SW、SD 指令在 uncache 的访问方式下写入开销远大于 UA,但会以折半的速度递减,而在 UA 方式下则开销递减,但开销降幅较小且逐渐趋缓。另一方面是在 UA 方式下 FIFO Entry 的数目确实决定了理论上能够同时在总线上传输的 UABE 数目,因此理论上拥有 8 项 FIFO Entry 的龙芯 3A 处理器 UA 加速效果要优于拥有 4 项 FIFO Entry 的 MIPS R10000 处理器。

3 MMAP 系统调用 UA 扩展

MIPS 指令集兼容的龙芯处理器的内核态代码全都运行在 unmapped 地址空间中,因此在内核态空间使用 UA 机制是比较简单的,只需要修改 64 位访存地址的[63,61]三位即可,而龙芯处理器用户空间的地址都是 mapped,因此用户空间的 UA 机制必须通过设置页表 cache 属性位^[9]来实现,直观的做法是通过添加一个修改页表 cache 属性位的系统调用来实现 UA 的支持,但这样会带来同步的开销,因为不确定当前要修改的页表是否已经在 TLB 中。经过分析发现,Linux 系统图形库中对 GPU 配置寄存器及存储空间进行读写,以及那些提供给应用程序对 GPU 存储空间进行读写的 API 都是通过

MMAP 系统调用将 GPU 存储空间映射到应用程序的用户地址空间,然后图形库或应用程序通过映射后的虚拟地址对 GPU 配置寄存器空间及存储空间进行读写,MMAP 系统调用映射的过程实际上就是应用程序构建 GPU 存储空间页表项的过程,这时构建的页表一定不在 TLB 中,事实上页表 cache 属性也正是在这个时候设定的,因此可以通过对 Linux 系统标准的 MMAP 系统调用进行简单的属性扩展即可实现对 UA 机制的支持。

图 3 是对 Linux Kernel MMAP 系统调用进行 UA 扩展后 MMAP 执行中与 UA 有关部分的执行流程图。首先在 MMAP 系统调用中获取用户程序页表后,计算用户请求页属性时加入 UA 方式映射与否的判断,如果用户请求了 UA 的映射,则置位虚拟页映射标识 VM_UA,之后在进行页映射操作时遇到 VM_UA 标识则将当前进程的虚拟内存区域(virtual memory area, VMA)结构 cache 属性置为 UA(0x111),最后由于 GPU 存储空间映射属于 IO

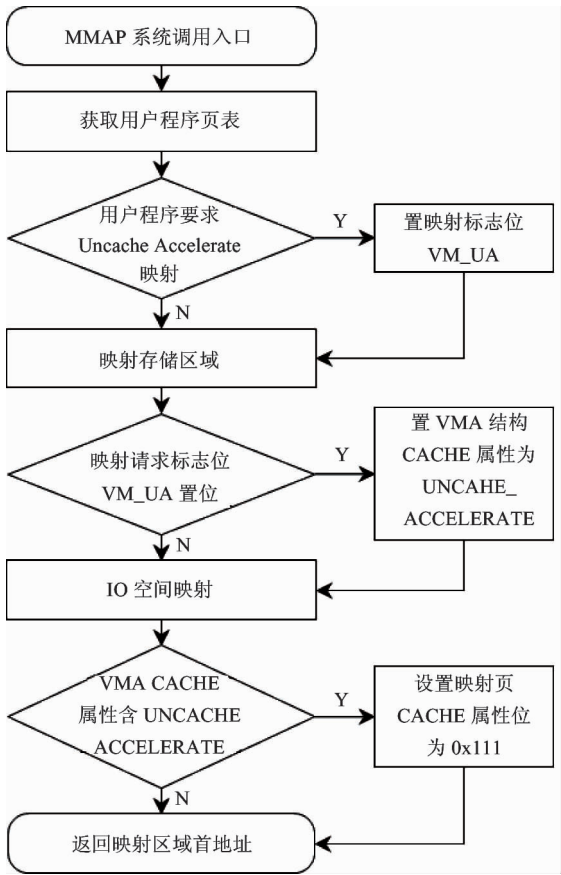


图 3 MMAP UA 扩展

空间映射,因此需要强制设置 uncache 属性,这里修改为遇到 UA 属性则不修改当前映射的 cache 属性。扩展 MMAP 系统调用属性后,只需要简单地之前通过 MMAP 映射 GPU 存储空间代码修改为 UA 属性的,另外做一个简单的软件同步就可以对原有接口进行 UA 优化。

4 UA 与 uncache 对比实验平台

本文的 UA 性能优化实验采用了龙芯 3A + 2H 开发平台,该平台的逻辑结构如图 4 所示。其中龙

芯 3A 芯片是该平台的主处理器,龙芯 2H 芯片^[10]在该平台上作为北桥芯片,两个芯片通过 HyperTransport 总线互联形成一个完整的硬件平台。

龙芯 3A 处理器内部主要由两级的交叉总线组成,其中一级交叉总线 AXI 连接了 4 个作为主设备的 GS464 处理器核,作为主从设备的 HT 控制器以及 4 个作为从设备的二级 CACHE 模块,二级交叉开关则主要连接了作为主设备的 4 个二级 CACHE 模块与作为从设备的内存控制器 (memory controller, MC) 模块。龙芯 3A 的 HT 控制器通过板级总线与龙芯 2H 的 HT 控制器连接形成数据通路。

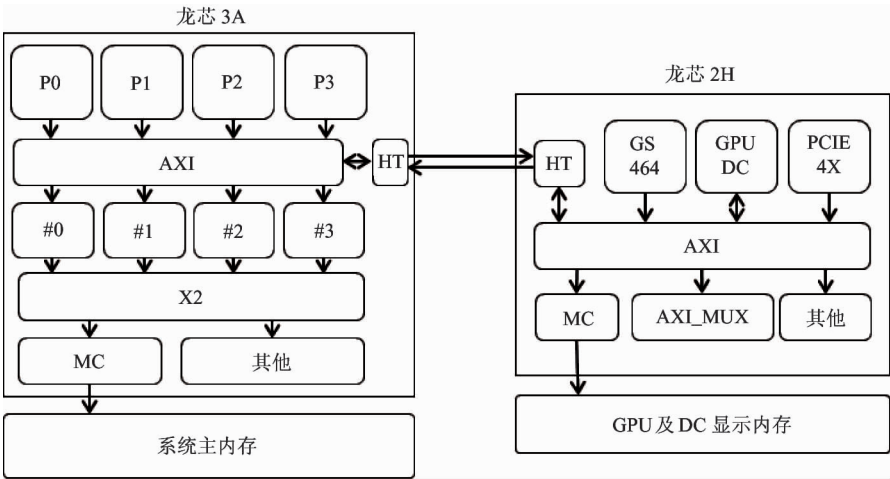


图 4 龙芯 3A + 2H UA 优化对比实验平台

龙芯 2H 芯片内部包含了一个 VIVANTE GC1000 GPU 模块,下文所描述的 GPU 驱动即表示该 GPU 的驱动程序。该 GPU 与 HT 控制器以及显存的内存控制器是通过片内的 AXI 总线互联在一起,其中 HT 和 GPU 是主从设备,内存控制器 MC 是从设备。

由图 4 可以看出,无论是龙芯 3A 处理器芯片内部的 GS464 处理器核心访问本地内存还是龙芯 2H 处理器芯片访问本地显存,都只需要经过片内的总线,开销都比较小。但如果是龙芯 3A 的 GS464 处理器核心通过 HT 访问显存与 GPU 配置寄存器或者是龙芯 2H 的 GPU 通过 HT 访问系统内存,则该访问不仅要通过龙芯 3A 与龙芯 2H 片内的 AXI 与 HT 总线,还需经过板级的 HT 互联总线,多次的协议转换与较长的数据通路导致了跨芯片访问的开

销远大于本地存储空间访问的开销。

5 GPU 存储空间 UA 与 uncache 写性能实际对比测试

第 2 节从理论上分析了各种位宽的访存指令在 UA 模式下相对于 uncache 模式的连续 GPU 存储空间写入性能差异,下面采用第 3 节介绍的 UA 扩展后的 MMAP 系统调用在 3A + 2H 平台上进行实际的对比测试。

图 5 显示了在龙芯 3A + 2H 平台上采用 uncache 和 UA 两种 MMAP 属性对 GPU 存储空间进行映射后,分别采用 SB、SH、SW 和 SD 4 种指令对连续的 256M 连续 GPU 存储空间进行写操作的耗时,由图 5 可以看出,在 uncache 模式下,采用 4 种指令

对同样大小的 GPU 存储空间进行写操作的时间基本符合之前分析的依次折半;但在 UA 模式下,采用 4 种指令对同样大小的 GPU 存储空间进行写操作的时间开销几乎相同,这是由于之前理想情况下的分析没有考虑 CPU 指令执行速度和外部总线带宽的极限,这里 4 种指令在 UA 模式下连续写入 256M 数据的耗时都是接近 1s,这是由于在 UA 模式的写入接近了该平台处理器访问 GPU 存储空间的数据链路的写入带宽上限,因此实验结果表现为 UA 模式下不同指令连续地址写入理论上就比较小的性能差异更小了。

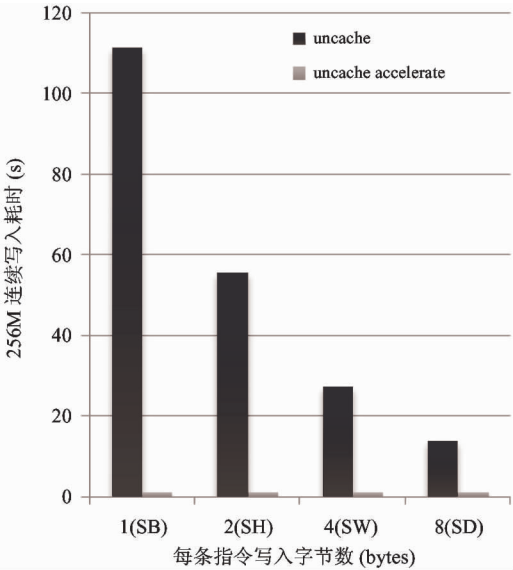


图 5 不同的 stroe 指令 uncache 与 UA 写入耗时对比

由该实验结果可以初步确定在一般的 uncached 连续访问操作中(连续的 SW,SD 指令)采用 UA 对龙芯 3A+2H 处理器平台上的 GPU 存储空间访问进行优化可以对连续 GPU 存储空间写入带来超过 10 倍的性能提升。

6 GPU 驱动的 UA 优化

由上文的分析和初步实验可以确定,通过 UA 方式可以极大地提高处理器访问 GPU 配置空间与存储空间的写入性能,因此我们可以推断如果将 GPU 驱动中 CPU 向 GPU 发送命令、传递消息的方式修改为 UA 方式的,那么那些需要向 GPU 配置/存储空间写入较长数据的图形 API 性能将会有较明

显的提高。

龙芯处理器平台的操作系统内核完全工作在 unmapped 地址空间,因此处于内核态的 GPU 驱动中所产生的 GPU 存储空间访问的地址都是 unmapped 的,对于 unmapped 地址的 UA 访问在本文第 1 节已经介绍了,这里对 GPU 驱动中 GPU 配置/存储空间的 UA 改造只需要做下面两个操作即可:

(1) 将驱动中写入 GPU 配置寄存器或存储空间所用的地址与常数 0X3800_0000_0000_0000 进行逻辑或,也即将 64 位 unmapped 的物理地址 cache 属性位置为 0x111。

(2) 在对当前 GPU 配置寄存器或者存储空间的写入操作完成后读取一个 UA 地址,强制将 UA buffer 中缓存的数据写出以保证一致性即可。

在 VIVANTE GC1000 GPU 的驱动中,有大量操作都是通过向 GPU 发送命令完成的,这些命令的长度从几个字节到几百个字节不等。对 GPU 驱动进行 UA 改造后,使用 x11perf 工具在龙芯 3A+2H 平台 XServer 接口性能做了一次全面对比测试,图 6 显示了部分改造前后性能提升较大的接口,由于各接口 x11perf 分值差异较大,这里做了归一化处理,即对部分分值较低的接口测试结果做了乘 10 处理,分值较高的接口测试结果做了除 10 处理,已在图中标出。

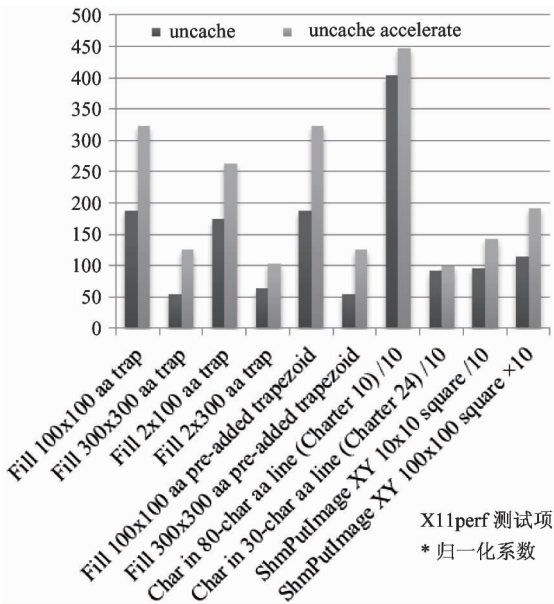


图 6 UA 优化 GPU 驱动后部分 x11perf 性能提升项

经分析发现,这些测试项所执行的操作均需频繁地调用驱动程序中的命令发送及 GPU 配置操作,因此对 GPU 驱动中的配置及命令发送的 UA 改造给这些接口带来了较大的性能提升。

7 Xserver 中 Xvideo 接口的 UA 优化

Xserver 的 Xvideo 扩展接口是 Xserver 提供给播放器软件使用的一个加速视频播放时每一帧图像的 YUV(一种颜色表示方式,Y 表示亮度,U 和 V 表示色度)格式到 RGB 格式转换以及转换后图像显示的接口,该接口是一个典型的应用程序从用户空间拷贝大量数据到 GPU 存储空间的应用程序接口,MPlayer 播放视频时的 xv 输出就是采用 Xserver 的 Xvideo 扩展输出,MPlayer 在初始化时会将 GPU 存储空间用于存放视频解码 YUV 格式数据的空间通过 MMAP 映射出来,之后每次需要显示一帧图像时,MPlayer 会向 Xserver 发送一个显示一帧数据的请求,Xserver 收到该请求后会调用 ProcXvShmPutImage 处理该请求,其中主要处理流程就是将 MPlayer 图像数据缓冲区中的 YUV 数据拷贝到 GPU 存储中,然后由 GPU 将该 YUV 数据转换为 RGB 格式并且完成窗口融合计算显示出来,由于目前视频分辨率普遍较高,因此这里数据拷贝的开销是整个 ProcXvShmPutImage 接口的主要开销。

图7显示了使用MPlyaer采用Xvideo输出播放

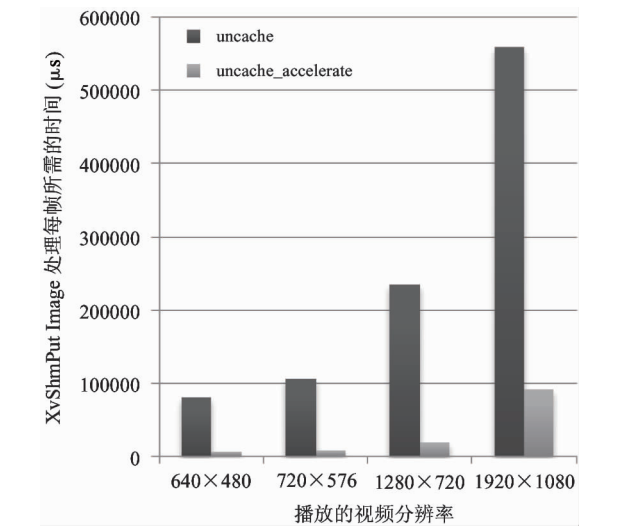


图7 UA 优化 Xvideo 接口前后性能对比

4 种常见分辨率视频时,龙芯 3A + 2H 平台分别采用 uncache 和 UA 两种方式映射 GPU 存储空间进行 YUV 数据拷贝的情况下,ProcXvShmPutImage 接口处理每一帧数据所需时间的差异。其中分辨率低于 1080P 的视频输出采用 UA 接口有 12 倍的加速比,而 1080P 的视频输出加速比为 6,这是由于 GPU 的 YUV 到 RGB 硬件转换单元转换 1080P 分辨率视频时需要进行分段操作,导致 YUV 到 RGB 格式转换在视频输出中耗时占比较高导致的。

8 结 论

在传统的主存和 GPU 存储分离的系统结构下,处理器与 GPU 之间的通信和数据拷贝往往是系统性能尤其是系统图形性能的瓶颈。虽然显卡通过 DMA 的机制提高了 CPU 与 GPU 之间的大块数据拷贝的性能,但 CPU 到 GPU 仍然存在大量较小规模的数据和命令交互,尤其是很多数据交互是由应用程序使用 MMAP 将 GPU 存储空间映射到应用程序地址空间直接操作的,而这些情况并不适合 DMA 机制。龙芯处理器实现了 UA 机制来加速处理器的 GPU 存储写入速度,本文一方面通过该机制优化了 3A + 2H 平台的 GPU 驱动中对 GPU 配置空间的写入配置操作,x11perf 测试结果显示提高了 Xserver 部分接口性能,另一方面通过将龙芯平台 UA 机制集成到标准的 MMAP 系统调用中,通过扩展后的 MMAP 优化了一个典型的需要向 GPU 拷贝大量数据块的应用程序接口 ProcXvShmPutImage,实际测试该接口的性能也有大幅度的提高。

参考文献

[1] Fujii Y, Azumi T, Nishio N, et al. Data transfer matters for GPU computing. In: Proceedings of the 2013 International Conference on Parallel and Distributed Systems, Seoul, Korea, 2013. 275-282

[2] Van Werkhoven B, Maassen J, Seinstra F J, et al. Performance models for CPU-GPU data transfers. In: Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, USA, 2014. 11-20

[3] Yang X J, Tang T, Wang G B, et al. Mptostream: An Openmp compiler for CPU-GPU heterogeneous parallel systems. *Science China Information Sciences*, 2012, 09: 1961-197

[4] Yeager K C. The Mips R10000 superscalar microprocessor. *IEEE Micro*, 1996, 16(2): 28-41

[5] Silicon Graphics, Inc. MIPS R10000 Microprocessor User's Manual. Mountain View: MIPS Technologies, Inc, 1997

[6] Hu W W, Wang J, Gao X, et al. Godson-3: A scalable multicore Risc processor with X86 emulation. *IEEE Micro*, 2009, 29(2): 17-29

[7] Xiao B, Zhang Y F, Gao Y P, et al. Godson-2h: A complex low power Soc in 65nm Cmos. In: Proceedings of the 2012 IEEE 55th International Midwest Symposium on Circuits and Systems, Boise, USA, 2012. 398-401

[8] 林伟. Linux 内存管理子系统在龙芯 2 号上的优化: [硕士学位论文]. 北京:中国科学院计算技术研究所, 2005. 8-10

[9] 伍鸣, 张福新, 林伟等. 龙芯 2 号处理器系统优化关键技术. *计算机研究与发展*, 2006, 06: 980-986

[10] Zhang X H, Cong M, Chen G Q. Software and hardware co-designed multi-level Tlbs for chip multiprocessors. In: Proceedings of the 2011 IEEE 11th International Conference on Computer and Information Technology, Paphos, Cyprus, 2011. 609-614

Graphics and video performance optimization with Godson UA mechanism

Zhang Shuangshuang^{* ** ** *}, Meng Xiaofu^{* **}, Wang Wenxiang^{* **}, Gao Xiang^{****}

(^{*} 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)

(^{****} Loongson Technology Corporation Limited, Beijing 100095)

Abstract

The study aimed to deal with the bottleneck of memory data transmission latency between CPU and GPU in the widely used CPU + GPU architecture. The uncache acceleration (UA) mechanism realized in the Godson GS464 processor core was adopted to optimize the GPU memory access interface, and the performance of store operation from CPU to GPU was significantly improved. The mechanism of UA realized by GS464 processor core was described in detail, and a theoretical analysis of the possible performance improvement due to the UA mechanism compared to normal uncache operations was conducted. The GPU driver performance of the platform of Godson 3A + 2H was optimized with UA by altering the access mode of GPU configuration and memory space in the GPU driver from uncache to UA, and the x11perf testing result showed that the performance of some interfaces of the Xserver was improved by 5% ~ 230% after the optimization of GPU driving by UA. By extending the standard Linux system call interface MMAP with UA, the Xvideo extension in Xserver was optimized, and the test result showed that the optimization made the Xvideo interface 6 ~ 12 times faster.

Key words: CPU core, uncache acceleration, graphics, video